EMANUEL BERNDL

EMBEDDING A MULTIMEDIA METADATA MODEL INTO A WORKFLOW-DRIVEN ENVIRONMENT USING IDIOMATIC SEMANTIC WEB TECHNOLOGIES

EMBEDDING A MULTIMEDIA METADATA MODEL INTO A WORKFLOW-DRIVEN ENVIRONMENT USING IDIOMATIC SEMANTIC WEB TECHNOLOGIES

EMANUEL BERNDL
MASTER OF SCIENCE (M.SC.)

Doctoral Thesis

December 2018

University of Passau
Faculty of Computer Science and Mathematics
Chair of Distributed Informations Systems

In many ways, people growing up with the Web and now the Semantic Web take the power at their fingertips for granted.

— Tim Berners-Lee, 2005

ABSTRACT

The Semantic Web exists for about 20 years by now, but its applicability as well as its presence does not live up to the standards of its original idea. Incorporated Semantic Web Technologies do have an initial barrier to learn and apply, which can discourage many potential users. This leads to less available data overall in addition to decreased data quality.

This work solves parts of the aforementioned problem by supporting idiomatic entry to those Semantic Web Technologies, allowing for "easier" accessibility and usability. Anno4j is a Java library that implements a form of Object-Relational Mapping for RDF data. With its application, RDF data can be created via a mapping by simply instantiating Java objects - an object-oriented programming concept the user is familiar with. On the other side, requesting persisted data is supported by a path-based querying possibility, while other features like transactional behaviour, code generation, and automated validation of input contribute to a more effective, comprehensive, and straightforward usage.

A use-case is provided by the MICO Platform, a centralised software instance that connects autonomous multimedia extractors in a workflow-driven fashion. This leads to a rich metadata background for the inserted multimedia files, enabling them to be used in diverse scenarios as well as unlocking yet hidden semantics. For this task it was necessary to design and implement a metadata model that is able to aggregate and merge the varying extractor results under a common “headline”: the MICO Metadata Model.

The results of this work allow the use case to incorporate idiomatic Semantic Web Technologies which are then usable natively by non-Semantic Web experts. Additionally, an increase has been achieved in forms of data integration, synchronisation, integrity and validity, as well as an overall more comprehensive and rich implementation of the multimedia extractors.

ZUSAMMENFASSUNG

Das Semantic Web existiert nun seit rund 20 Jahren, jedoch ist dessen Anwendbarkeit und Präsenz nicht auf dem Stand, der in der ursprünglichen Idee angestrebt wurde. Die verwendeten semantischen Technologien besitzen eine Einstiegshürde die potenzielle Nutzer abschrecken kann. Dies führt sowohl zu weniger verfügbaren Daten sowie einer schlechteren Datenqualität.

Diese Arbeit löst einen Teil dieses Problems, indem diese Technologien ëinfacheränwendbar für den Benutzer gemacht werden. Anno4j ist eine Java Bibliothek die eine Form einer objektrelationalen Abbildung für RDF Daten umsetzt. Dadurch können RDF Daten über ein Mapping erstellt werden, indem lediglich einfache Java Objekte erstellt werden müssen - ein objektorientiertes Konzept das dem Benutzer bekannt ist. Eine Abfrage der Daten wird durch Pfad-basiertes Querying vereinfacht, während andere Eigenschaften wie transaktionales Verhalten, Code-Generierung und automatisierte Validierung weitere Beiträge zur effektiveren, umfassenderen und unkomplizierteren Benutzung liefern.

Den Anwendungsfall stellt die MICO Platform dar, eine zentrale Softwareinstanz die über Workflow-orientierte Arbeitsweise autonome Multimedia Extraktoren miteinander verbindet, um eingegebene Multimediainhalte auf ihren Metadatenhintergrund zu untersuchen. Dies erlaubt die Inhalte in einem breiterem Spektrum sowie eventuell unvorhergesehenen Szenarien zu erschließen. Ein eigens entwickeltes RDF Metadatenmodell, das MICO Metadata Model, bildet dafür ein uniformes Datenformat, um alle erstellten Ergebnisse unter einen gemeinsamen Hut zu bringen.

Die Ergebnisse dieser Arbeit ermöglichen in diesem Anwendungsfall eine Vereinfachung der semantischen Technologien für Semantic Web-fremde Benutzer, eine Verbesserung in Form von Datenintegration, Synchronisation und Integrität, sowie eine funktional reichere Implementierung der Multimediaextraktoren.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

PREFACE

1 THE SEMANTIC WEB

In the life of today, many "worldly workflows" are maintained utilising the possibilities supported by the modern World Wide Web. Calendars with various appointments are managed over several end devices like computers, laptops, and mobile phones. Oftentimes this planning and scheduling of events is even interwoven with other peoples' time schedules. Among these appointments, meetings, parties, and other social events are also organised online, with many different possible peers participating in the organisation. Next to this, activities like shopping and the discovery of information for various activities have become a daily routine for almost everyone.

Despite this fact of daily repetition and the efforts that have been spent in order to make them as fluent and functional as possible, there still exist perceived boundaries in terms of inconvenient behaviour or missing automatism. The mentioned workflows still require at least a baseline of human interaction and decision taking, as there is no applied "intelligence" or "knowledge" on the side of the executing computer, in contrast to what humans can do. For example, a computer (in most common cases) can help to schedule temporal matters, warning a user that two appointments occupy the same time slot, but rarely it can involve further information: imagine two consecutive lessons with a break of ten minutes in between. Regarding only the time, this seems feasible, but there may be other factors like the geological position of the two lessons, which would require a 30 minute ride to get from one place to the other, rendering the situation to be impossible to fulfil. Hence, the user still has to do the task of minding such "external" factors and assess the impact of them on his given activities.

In [27], Berners-Lee, Hendler, Lassila, et al. reinforce this by stating that Web content of today is in general designed for humans to read and interpret. A computer on the other hand can only parse these pages for their layout and underlying data structure, but in most cases they cannot manipulate them meaningfully or have a way of interpreting their semantics. But they envision a solution or an enhancement to the situation, an interaction between human and computer that is more fluently and beneficiary by being supported with a component that can access certain data in order to support the user in a seemingly intelligent way: in 2001 Berners-Lee, Hendler, Lassila, et al. claimed that computers will step by step increase their capabilities to process and especially "understand" the data that they merely

display at current times. The idea of the **Semantic Web** was born, originally coined by Berners-Lee et al. [28].

1.1 THE VISION AND IDEA OF THE SEMANTIC WEB

This section will not give a definition or deeper insights about the concept of the Semantic Web, but will rather try to further convey the general vision or the idea from a more high-level point of view. A more thorough and technical definition of it is given in Section 2.1.

For the familiarisation with the Semantic Web, Section 1.1.1 will illustrate a real-world example in extension to the one supported by Berners-Lee, Hendler, Lassila, et al. in [27]. Afterwards, Section 1.1.2 will support a statistical view about the acceptance of Semantic Web technologies to this date and will compare the current technical development in contrast to what was envisioned. Then, potential and assumed issues are enlisted that lead to the numbers shown before, which will constitute the perceived "issues" about the vision of the Semantic Web in Section 1.1.3 and Section 1.1.4.

1.1.1 *A Semantic Web Use Case*

At the beginning of their article, the authors of [27] picture an ideal scenario for the application of the envisioned Semantic Web. In the scenario, two people are intending to solve a problem with the support of the concept. Therefore, they use their so-called **Semantic Web agents** - devices that are not only able to use the "normal" Web for their purposes, but rather the information supported by the Semantic Web. The difference between the two concepts of the Web is made clear, as the semantic agent seems to support the protagonists in their problem-solving process to a way higher degree than currently possible. The agent makes use of various "information peers" in order to suggest solutions that fit the defined task the most. These peers can be of different kind, but they always support some sort of information in the form of metadata - data that further describes or characterises other data or information - that can be seen or interpreted as knowledge. Examples are homepages, personal applications like calendars or address books, sources that support medical or transportation information, and so on. The task to solve is generally decomposed of multiple diverse requirements. These requirements may originate from different natures, like descriptions, characteristics of people, temporal and spatial factors, and/or multiple people taking part. A scenario that is similar to the one in [27] is depicted in example 1.

Example 1. *Two people (Liz and Arnold) are trying to find a fitting destination and point of time for a trip to a zoo. The solution they are looking for needs to fit the following requirements:*

- *The zoo needs to **feature two certain animals**, as Liz and Arnold are especially interested in seeing giraffes and pandas.*
- *That specific zoo needs to be in a **certain defined spatial distance** and needs to be **within reach of public transportation**.*
- *Furthermore, the zoo should have at least a decent **public rating**.*
- *As both Liz and Arnold's vacation time is short and there are already other appointments, the point of time of the trip needs to fit their current **timely constraints** defined by their calendars.*
- *In order to gain more detailed information, the two would want the zoo to support **media representations** of their attractions - at best about the desired animals, in good quality, and available for smartphone display.*
- *Liz's agent is defined to **trust** Arnold's agent - and vice versa - which means that both agents incorporate and trust information that is passed by the other agent to a full extent.*

With this input, the semantic agent prompts them with possible solution options. As one of them is not quite happy with those, they do some readjustments of the requirements and make the request more strict. After this has been passed to the agent, an adapted solution is then given - including some warnings that the proposed plan, including the stricter requirements, is in need of some temporal adjustments of one of the person's previously planned time schedule.

Every presented solution of the agent can be further differentiated in order to see the information about the decisions made that lead to the respective solution. At last, they find a solution they are both comfortable with. As the task and its solution is then accepted by both peers, the respective resources are then automatically updated by the semantic agent to the requirements supported by this task. In this case, both the calendars of Liz and Arnold are updated for future tasks to consider and take into account.

This small example does already highlight the complexity that such a "semantic task" can attain, even with merely few requirements. The requirements arise from different origins, which means that the supported information must not necessarily be supported in a uniform syntax. Next to this fact, also the semantics conveyed by the information are in general not understandable right away or in a uniform way. Also, they can have varying impact on the success of the task and hence the possibilities of achieving it. Pareto optimal solutions are possible, however this also induces the agent to communicate this circumstance accordingly. In addition, next to proposing solutions to the defined tasks, the semantic agent is also supposed to help in the process *after* the (possibly first) solution that has been prompted to its user. By fine-tuning the previously posed requirements, other adapted solutions are to be computed, even in excess

of the boundaries of the defined requirements - in Arnold's example 1, the agent proposed a solution that needed the adjustment of *other already planned* activities.

Nevertheless, although this problematic poses a broad spectrum of difficult tasks and sub-tasks that need to be achieved in order to support a problem solving process with at least satisfying success, the Semantic Web vision is not only composed of "how it should or could be"-thoughts, but also ideas, proposals, specifications, standards, and milestones about the way of how to get to its fulfilment.

Different insights and useful ideas from already existing standardisations and concepts are applied and extended in order to align with the existing Web architecture. Among those, the HTML protocol, URL and URI identifiers, as well as the notion of resources in the same domain are adopted and lifted to the Semantic Web concept. The Resource Description Framework RDF [113] supports a way of syntactically representing the informational data that is contained in various data silos, which is straightforward to use. These data silos incorporate information of different domains and can interlink each other in order to further multiply the usefulness of every single information peer. In order to support convenient usage of a silo's data, descriptive schemata can be supported alongside the data that give a description about the overall structure of the respective data, called an ontology (see Section 2.2.3). The two most commonly used schema modelling languages [173] are the Resource Description Framework Schema RDFS [37] [72] and the more detailed Web Ontology Language OWL [76] [70], which both conveniently implement their schema information in RDF. This is very similar to the commonly known XML standard [111] and their respective XSD schemata [112]. Last but not least, a comprehensive querying language is given by the SPARQL Protocol and Querying Language SPARQL [71], which allows for rich and diverse querying capabilities in order to query the information pieces from the information silos. All of these standardisations and concepts will be referred to as the **Semantic Web technologies** in the remainder of this work and will be covered in detail in Chapter 2.

Around these formal cornerstones, Berners-Lee [26] also defines best practises that should be followed whenever one creates Semantic Web or Linked Data. Linked Data is a concept closely related to the Semantic Web, which will be compared also in Section 2.1. These best practises define how the data should be structured and linked, and also qualitative requirements that should be met. When these rules are followed, data silos can be accessed and its data can be used in the most standardised and efficient way.

With all these contributions, a thorough and comprehensive setting is given that supports developers and content creators to lift their "general" Web applications to the standards of the Semantic Web in order to profit from all its benefits. Nevertheless, the Semantic Web

concept and its picking up in the Web world of today is not as far as it was envisioned in 1998. The following section will give an overview about the Semantic Web's perceived current standing.

1.1.2 *A Look at the Current State of the Semantic Web*

At the time of writing of this part of the thesis[1] many Semantic Web datasets and databases are published online. One of the best known statistics about these datasets is the **Linking Open Data Cloud Diagram**[2] (which in combination of its evolution will be covered in Section 2.1), which shows various datasets that have been published in Linked Data format by contributing to the Linked Open Data community project and other individuals or organisations. With the last statistical counting in August 2017 it featured 1,163 different sets of data[3] originating from various domains like Life Sciences, Geography, Media, or Government.

Next to the LOD Diagram, one of the best known datasets is **DBpedia** [119], a collection of data that at this moment contains 4,58 million different semantic "things", from which more than 92% (4,22 million) are classified by a consistent ontology [53]. These have been extracted and structurally conditioned from various Wikimedia[4] projects.

Both of these data accumulations show that acceptance of the Semantic Web concept is definitely existing. A survey conducted by Cardoso [41] in 2007, some years after the initial vision, showed a numerical breakdown of some statistics about the usage of the Semantic Web. The author did not see the vision fully accomplished at the point of time when he undertook the evaluation, but he clearly accentuated the contributions that have been made in the domain. He also stated his assumption that a full mainstream adoption of the Semantic Web is five to ten years away, but also that surveys like his are important in order to depict the road ahead and reason insights from their results. One of his results that is interesting the most for this thesis is shown in Figure 1, which shows a survey conducted with Semantic Web appliers and the distribution of their specified employment role.

The numbers of Figure 1 are backed up by a more generalised evaluation, grouping the participants to work either in the academic or industrial domain, or both. These numbers showed that around 66% of the participants classified their working field to the academic domain, while only about 18% stated industrial work, and the rest characterised a mixture. This result does comply with the numbers of Figure 1, conveying that the application of the Semantic Web concept is tilted towards the scientific sectors rather than industry. This indicates

1 October 2017
2 http://lod-cloud.net/ (last visited 03/12/2018)
3 An updated version of the LOD Cloud features 1,231 datasets in November 2018
4 https://www.wikimedia.org/ (last visited 03/12/2018)

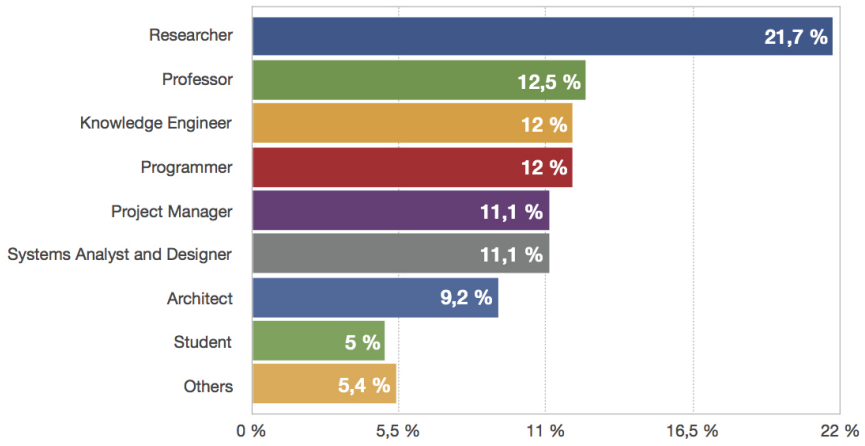


Figure 1: The Role of Semantic Web Users that Participated in the Survey of Cardoso [41] (From Which the Numbers and the Table Are Adopted).

that the industry does not yet incorporate Semantic Web technologies to benefit from its advantages, but it is rather used for research purposes. Cardoso estimates that this tilted circumstance does show a direct reflection of the current market in 2007 and sees a big potential in a technology transfer. Nevertheless, in this day and age, more than ten years later, this situation does still seem to remain.

A more present and more technical evaluation is done by Assaf, Troncy, and Senart [10], who conducted an experiment that asserts the quality of the datasets contained in the LOD Diagram mentioned above. They did this experiment with their application called Roomba [9], which automatically checks the defined datasets for different potential existing metadata, incorporating not only content metadata but also metadata that gives information about the usage of the respective database. The categories they check for are categorised as *general*, *access*, *ownership*, and *provenance* information. At some occasions, when Roomba detects faulty metadata, it is able to correct it automatically.

This project has been developed with the importance of metadata provisioning in mind. Under this task the authors understand the assignment of models and descriptive information next to the actual dataset itself, which contains in particular information that shows how the data can and should be used and how it can be accessed, as well as the nature and content of its resources.

The results of Assaf, Troncy, and Senart [10] are clear, as they claim that many of the datasets need attention. Many of the datasets lack informative access information, while the resources of many others suffer of low quality, exhibiting undefined or no values at all. The problematic here is that especially these two metrics represent a deciding factor for industry or enterprises to pick up the respectively supported data.

1.1.3 *Open Issues of the Semantic Web*

Both of the given evaluations indicate the same tendency of the Semantic Web concept being more accepted in the scientific domain rather than industry. This is strange, as research sees high potential in the application of Semantic Web technologies and Semantic data in intelligent systems [19]. Also, by having less contributing developers and users, this does kind of counteract the overall idea of the Semantic Web: having many collaborating contributors to the overall databases increases the knowledge of every participant, as there is not just more data, but for example also possible peer reviews for better data quality and better linkage between databases for more sophisticated and continuative data.

As a reason for this circumstance, several open issues of the Semantic Web technologies are assumed, which must be addressed in order to increase the number of potential users. These issues are the following:

INSUFFICIENT KNOWLEDGE In contrast to relational databases that are more commonly known and applied in industry as well as taught in basic studies, the Semantic Web concept and technologies as an extension to the common Web represent a more specialised and new way of data representation and data handling. This comes with an own syntactical structure of the core data, own querying languages, and further more specialised specifications and standards. Solely the fact that learning these technologies requires timely effort could discourage potential users, as they are also potentially not aware of the benefits of the Semantic Web beforehand. An exemplary study in 2009 at the University of Potsdam shows that out of 8 selected students in the studies of both Bachelor and Master in Computer Science, only three issued that they have little experience in the Semantic Web or rather heard of the concept at all, while the other five never came in contact with it in any way. On a scale from 0 (no knowledge whatsoever) to 10 (expert in the Semantic Web domain), the interviewees scored a mean of 0,9 on that topic [132].

COMPLEXITY Next to the number of Semantic Web technologies that need to be learnt, the technologies do also impose technical complexity that does further complicate the process of applying the technologies in order to eventually take part in the Semantic Web. This can for example be seen in the formalism that RDF data and their semantics are built on, which allow automated inferencing based on the given data in order to conclude new information. Another example of a perceived high complexity can be seen in the earlier mentioned Web Ontology Language OWL, which is based on so-called description logics [14] [85] that are

a subset of the first order predicate logic [126] [64]. Hence, in order to make proper use of a database that is OWL conform, the user has to have well-grounded knowledge of these logics and formalisms.

INCONSISTENT DATA It has already been shown in the research of Assaf, Troncy, and Senart [10], that many of the existing databases do have shortcomings when it comes to overall data quality, being expressed by undefined, missing, or incorrect values. This is even aggravated by the fact that in many cases open databases tend to be filled in uncoordinated fashion by various heterogenous parties with varying data quality standards, which eventually creates very inconsistent data [57]. Oftentimes this data is even contradictory to existing schemata that would actually suggest structure [84]. This can lead to miscellaneous problems when trying to work with the respective database, making it hard to work with the supported information in a technical or even automated way, especially in the industrial domain, where incorrect or false data is even more grave. Next to the evaluations shown above, Hogan et al. [84] also supports the claims that the problem of inconsistent data exists in many knowledge databases. Hogan et al. crawled and gathered a dataset consisting of 12,534,481 RDF statements which have been analysed for various highlighted errors and shortcomings like dereferencability and accessibility, syntax errors, or data noise and inconsistence errors. Next to the quality of the metadata, another shortcoming that has been mentioned already is the poor accessibility of some existing databases in terms of bad querying possibilities or the querying endpoint not being accessible at all, which further aggravates this problem of the Semantic Web.

1.1.4 *The Semantic Gap - A Metadata Problem in the Domain of Multimedia*

After giving a first introduction about the Semantic Web and enlisting its perceived issues, this section will complete the first look at the concept by showing a representative issue that eventually arises from the lack or bad quality of supported descriptive metadata. In reverse conclusion, these issues can be solved, or at least alleviated, with more people taking part in the Semantic Web as well as give them technical support to do so. The issue at hand is called the **Semantic Gap** [157] [75], which is mostly encountered in the multimedia domain. A visualisation of the problem can be seen in Figure 2.

As described above, the intention of the Semantic Web is to support descriptive data and information that is not only useable for humans, but also for computers. This definition of "usable" goes even further, as with metadata of good quality, computers are envisioned to partly

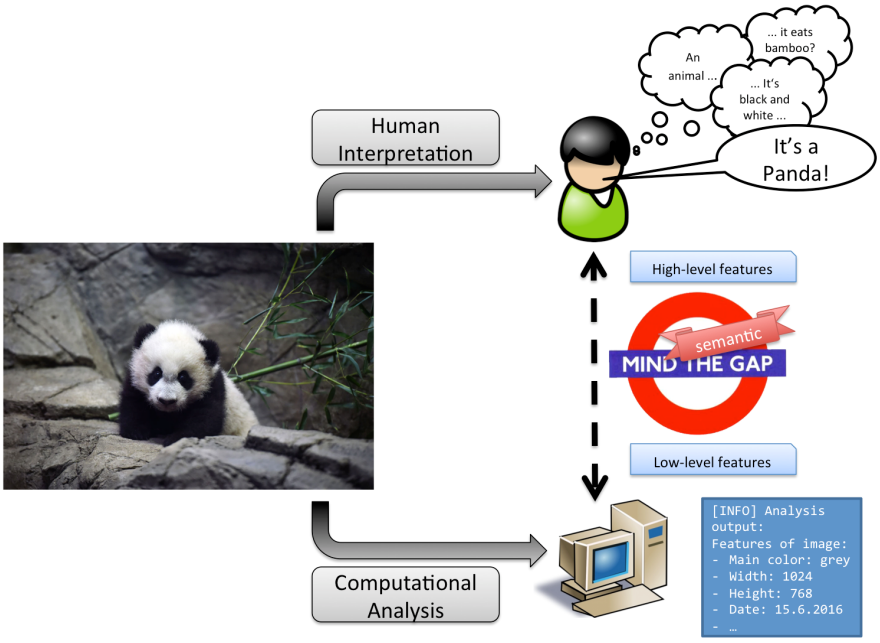


Figure 2: Visualisation of the Semantic Gap Problem.

even be able to "understand" the data in order to fuel further intelligent processes or systems [83].

The Semantic Gap depicts the circumstance of divagating interpretation capabilities of the same object between humans and computers. A human being can use its whole "inner knowledge base" - its past experiences and actions - which can be recombined and applied at an instant in order to fulfil complex chains of thought, eventually determining the information that is contained in a given object he sees - in the case of Figure 2 a panda animal that is depicted on a picture. The bits of information that a human uses in order to come to a conclusion are of a very complex and intellectually high nature and can therefore be called **high-level features**. As an example, a human can infer to see an animal by recognising a set of eyes with a muzzle relatively close to it. From that point on, he can interpret a bear-like shape and the black and white colouring to complete his interpretation.

By contrast, computers in most cases can not perform such an interpretation. They have to rely on technical information and more basic features that are more or less dependant on the given (multimedia) file. This is in general structural and technical information - so called **low-level features**. These features can oftentimes be determined by analysis processes and then be supported next to the given multimedia file in the form of metadata. In the case of our image, this could be simple image related information like the height, width, or date of the picture, while the simple analysis processes can use for example pixel information in order to identify the main colour or edges and hard colour breaks of the picture. Nevertheless, with this level of information it is rarely possible to determine the "meaning" of

the given multimedia object, therefore manifesting the Semantic Gap problematic.

To conclude, this section gave a short insight on the Semantic Web. A lot of good ideas in combination with a thorough and rich vision led to the development of comprehensive languages and standardisations that allow the interaction with Semantic data. Eventually, the aim is to make data not just use- and understandable for humans but also computers, in order to make the information directly accessible for intelligent systems. With the initial idea being dated back almost two decades at the point of writing this thesis, it has also been seen by numbers and evaluations that the vision however has not yet been fully enfolded and the potential of the Semantic Web is not applied, especially in the industry sector. In addition, a representative problem that arises from missing or bad quality metadata in the multimedia domain has been shown.

In order to find an explanation for this circumstance, some potential issues were identified that are expected to eventually lead to the weak acceptance of the Semantic Web concept. Lifting the concept to an all around higher acceptance or even a by default application in the Web world of today features many more other starting points than mentioned here, and would exceed the limits of this thesis alone. Nevertheless, this thesis will show that it is possible to bring the Semantic Web concept closer to developers and users by solving or avoiding the issues mentioned above. It is assumed that these require an up-front effort from the potential user in order to apply the Semantic Web technologies, eventually discouraging him to go further. With different contributions that align, extend, apply, enhance, or abstract from the existing Semantic Web technologies, it is envisioned to overcome the assumed problems, achieving a broader and higher amount of users, in the end creating not just better quality metadata, but also more metadata altogether.

1.2 A TECHNICAL SETTING FOR THIS THESIS - THE MICO PROJECT

Before the contributions of this thesis can be enlisted in Section 1.3, it is important to define the technical environment in which they are embedded in. This serves both as a foundational use case as well as a best-practice showcase that evaluates the results of this thesis from a technical standpoint. A general description of the overall project is displayed in Section A.1.1 in the Appendix, while a more detailed description of the mentioned setting is given in Section 5.2.

The MICO project focuses a multimedia analysis scenario, which can be closely related to the exemplary Semantic Web use case described in Section 1.1, as the incorporated peers are similar and can be theoretically exchanged through their high-level roles. In both cases, information peers are present that work autonomously and provide

information once requested. This returned information is generally available in a heterogenous format, which needs to be combined to be commonly understood. Dependencies can exist between different peers and eventually further information is to be gained once all the information bits have been gathered and recombined.

Another accordance can be found in the perceived issues at hand that both scenarios involve. The issues detected with the overall Semantic Web concept in Section 1.1.3 can directly be encountered and mapped to the MICO use case, as new technologies with high initial barriers need to be learnt and incorporated by various development peers, and eventual results were contributed by various information peers with varying backgrounds or purposes. Because of this, solving the circumstances in the MICO environment does have direct implications on the overall Semantic Web domain. Therefore in order to show approaches for the Semantic Web, the contributions of this work are embedded in MICO, which does serve as a technical basis as well as a best practice example.

1.3 RESEARCH QUESTIONS AND CONTRIBUTIONS

The contributions of this work in the domains of the Semantic Web and Multimedia aim to solve the scientific issues enlisted throughout Chapter 1. They can be divided into three core pillars with the addition of evaluation results. In combination with raised research questions, they are the following:

1. How to express, create, and especially recombine the created results of various heterogenous metadata producers?

Contribution: *Multimedia Metadata Model*
Metadata modelling and associated models are the backbone of every Semantic database, as they enable both the persistence and querying of the data in a structured fashion. Especially in the multimedia domain, metadata is very important for the multimedia's further application.

For that reason, the **MICO Metadata Model MMM** is developed which covers both of the domains of multimedia and the Semantic Web. The MMM is a modular, extensible, and rich metadata model for multimedia data that follows both Linked Data principles as well as requirements found in related work in terms of an ontology for multimedia. On top of comprehensive possibilities of addressing both the semantic content of a multimedia analysis process as well as the target that the content is referencing, the MMM introduces several provenance and composition elements, eventually allowing to create a diverse and traceable metadata background for analysed multimedia items and their results. With both its expressiveness and extensibility, the MMM is also able to incorporate heterogenous metadata

producers that are not known to a given system or application in advance.

2. Are there ways to lower the initial barrier that is posed by Semantic Web Technologies, which manifests mainly in the fields of creation, processing, and requesting of the respective semantic data?

Contribution: *An Abstraction Layer for Semantic Technologies*
The technologies applied in the Semantic Web impose high initial barriers for developers that are newcomers to the Semantic Web. Oftentimes this can lead to a discouragement and therefore non-participation of the respective developers which counteracts the overall vision of the Semantic Web.

As a solution to this, the Object-Rdf-Mapping library **Anno4j** is proposed, which supports an abstraction layer "on top" of the actual Semantic Web database. Using Anno4j, developers are not constrained to use Semantic Web technologies but can rather use concepts and processes they are familiar with, in this case, the object-oriented programming language Java. Doing so, the development of Semantic Web applications gets both facilitated and accelerated. Additionally, the accessibility of Semantic Web technologies gets diversified enabling a much broader field of potential users. Especially the querying component features several different ways to request existing metadata of a database, ranging from very basic, but sparsely configurable querying capabilities to specialised and more comprehensive path-based queries.

The following research questions will feature the term **complexity**, which is related to both **ontologies** of the Semantic Web and the respective pendant of **domain models** in the Anno4j library. These are assessed to be more complex, the more distinctive classes with particular relationships and properties they implement, and how the interplay amongst these classes is in terms of an inheritance hierarchy [153], which can be both deep and broad. A more detailed definition will be given in the course of this work.

3. With the Anno4j library applied as use case:
   - 3.1 Can Object-RDF-Mappings deal with complex domain models?
   - 3.2 Is the application of complex domain models possible in an efficient way?
   - 3.3 What can be done with Object-RDF-Mappings in order to (better) support data consistency?
   - 3.4 Can the application of Object-RDF-Mappings be enhanced by allowing different access technologies?

Contribution: *Enhancements to the Object-RDF-Mapping Concept*
Furthermore, technical extensions to the Anno4j library lift the mapping concept to a higher functional level, both in terms of more convenient application as well as better quality of produced metadata. The **Anno4j Generation Tool** allows the parsing of existing descriptive metadata schemata in order to automatically generate the necessary Java Classes for the library. This allows users to directly work with more complex domain models as well, as in contrast the implementation by hand is prone to errors and requires a high amount of timely effort. A possible concurrent generation of so-called "proxies" needs some up-front processing time, but its produced results reduce the subsequent time constraints of working with the complex domain model to a imperceptible minimum. Next to this feature, a validation component is realised that can automatically check data that is to be created, if it suffices previously defined validity requirements. If these are not met, no data is produced and therefore no faulty data is inserted into the database, in the end increasing the databases metadata quality. In addition to this, qualitative and descriptive feedback is given back to the user in the case of validity violations. The access diversity of the library is further extended by a RESTful extension that can directly be applied to an existing set of Anno4j Classes. This allows users to conveniently interact with the Semantic database via the commonly known HTTP standard.

Aside from the contributions enlisted above, which are motivated by a concretely defined research questions, the work done for this dissertation has also motivated further interesting topics that constituted meaningful results and therefore can also be seen as scientific contributions:

Further Contribution: *Embedding of the Application Contributions*
In a final step, all the other contributions are combined functionally in the MICO Platform. With the platform being affected by both the domains of Multimedia and the Semantic Web, the MMM and Anno4j represent solutions to increase the platforms applicability. Next to this point of view, this does also serve as a technical use case for the contributions.

Beyond that, further extensions to the last officially published version of the MICO Platform are proposed that mainly base on the latest Anno4j features (which were not incorporated in said release version). These have the potential to increase the convenience in application and the correctness of the produced results of the MICO Platform further. This construct is subsumed under the name MICO+.

As another minor contribution, the lifecycles of metadata in both the domains of Multimedia and Linked Data will be highlighted and discussed. A merged lifecycle is proposed that incorporates the re-

quirements of the lifecycles in order to generate a detailed representative about Semantic Multimedia metadata. This allows more fine-grained and modular insights about the process of metadata creation in the field of the Semantic Web, enabling better and more comprehensive capabilities to produce metadata that forms backgrounds for multimedia items. This approach is also incorporated in the MICO+ Platform concept.

Further Contribution: *Evaluation of Anno4j*
Enlisting related work in the field of Object-Rdf-Mapping backs up the Anno4j library from an overall technical standpoint. Therefore, evaluations and experiments have been conducted focusing on the additional features that the Anno4j library introduces to this kind of concept in order to support qualitative backing for the implementation. By enlisting runtime evaluations, it is shown that the incorporation of Anno4j does not impede the overall runtime of an application. These evaluations do also support the answer for research question 3.2.

As a side-result of the Anno4j Generation Tool evaluation, an awareness for the structure of metadata models and ontologies is gained, which allows to determine the "structural complexity" of a given schema. The evaluation process applies different structural parameters originating of both classic RDFS or OWL schema features and the Anno4j extensions to it, which eventually dictate the structure of the given schema. With the assessment of the mentioned complexity, it can be decided if the utilisation of a generation step is costly sensible or not in terms of overall application runtimes.

1.4 STRUCTURE OF THIS THESIS

This thesis is composed of five parts. The preface gives an introduction to the overall topic and domain of the Semantic Web and motivates the remainder of the work. Every chapter that introduces a core contribution is opened with a related work section, which supports insights as well as delineations and starting points for the respective own contributions. Also, these chapters are enclosed by an own short summary and outlook.

After an initial picture of the Semantic Web and its vision, Part ii establishes the concepts in a more detailed fashion. Therefore, Chapter 2 commences by a definition of the term "Semantic Web" and compares it to other existing, partially very similar, other concepts. Then, an in-depth overview is given of the related standardisations and languages that are present and commonly used in the Semantic Web. A focus point is thereby set on the backbone of every Semantic database: metadata and its modelling.

With the knowledge of metadata modelling, Chapter 3 combines this practice with the domain of multimedia. After a survey about related approaches, the Web Annotation Data Model as a basis for the own contribution is enlisted and explained. An extension to the WADM lifts its context to a cross-multimedia use case. The result of this constitutes the first contribution of this thesis: the MICO Metadata Model MMM.

Part iii complements the theoretical background gained from prior sections with technical pendants. Therefore, Chapter 4 covers the topic of Object-Rdf-Mappings, firstly in terms of related approaches and then elaborately introduces the second contribution in the form of the own ORdfM implementation Anno4j with its various features that allow an abstracted and rich approach to Semantic Web technologies. Afterwards, Chapter 5 ties all the prior contributions together and highlights the MICO Platform as a technical basis that incorporates the MMM and Anno4j for their beneficial factors that alleviate the addressed issues of the Semantic Web. Also, the technical steps that metadata takes throughout its lifecycle are accentuated and covered. Eventually, an extended form of the MICO Platform, called MICO+, is proposed with a description about its further potential benefits.

In order to validate the approaches and claims made throughout this thesis, Part iv first shows related evaluations and experiments conducted for the general concept of Object-Rdf-Mappings. Secondly, own extensions to the concept are evaluated qualitatively and discussed for their real-world application.

Part v summarises the thesis with a résumé, an outlook, and a future work section in Chapter 7.

Part II

MODELLING METADATA - CLASSIC APPROACHES AND A MULTIMEDIA CONTEXT

2

PRINCIPLES OF METADATA MODELING AND QUERYING

The vision of a fully integrated **Semantic Web** described in Chapter 1 presents a state of the Web in which computers can assist human workflows by supporting aggregated information found in the Web. The data that is incorporated in these semantic processes is envisioned not only to be readable, but rather be understandable for these computers. This enables a richer field of application for various procedures like a semantic search. An issued search query can be interpreted in a more "humanly fashion", with the result that functionality beyond the classic possibilities can be applied. This could for example be background knowledge about some fact. It is also possible to connect data collections and/or sources to one another in order to create a combined knowledge base that increases the applicability and usefulness of the data for every peer involved.

To achieve this, a more precise instrument of describing documents is essential. This description, or more precisely the parts of it, is called **metadata**, which is commonly defined as *data about data* [106] and constitutes the "fuel" of the Semantic Web technologies. This data generally contains information like title, dates, author information, and so on. Without a baseline of correct and sophisticated metadata associated to the data in the Web, as well as an adequate degree of quality of the same, a Semantic Web is not feasible. Next to the data itself, it is also important to have that data interlinked in order to achieve a coherent state that can encourage knowledge discovery. As it will be seen in the upcoming discussion about the evolvement of the Web, the glue that holds together the traditional document-centric Web are hypertext links between HTML pages. For the Semantic Web, the Web of Data, these links are RDF links between single pieces of data or data collections. Such an RDF link symbolises the relationship between two pieces of data. This relationship can have various types, each illustrating another semantic between the given data, for example friendship between two human entities or the affiliation of a book to its corresponding author [32].

With this importance of metadata and a linked structure in mind, this chapter will first position the idea of the Semantic Web in the landscape of other prevailing concepts in order to foster its potential advantages in Section 2.1. Afterwards, Section 2.2 explains the Resource Description Framework RDF, the de-facto standard when it comes to describing and modelling metadata in the Semantic Web, as

well as SPARQL - the SPARQL Protocol and Query Language - as the de-facto standard for querying semantic data in Section 2.2.7.

2.1 THE SEMANTIC WEB AND OTHER FAMILIAR CONCEPTS

To this point, various insights about the Semantic Web, its concepts and ideas, advantages, and also factors that are not yet as they have been envisioned initially have been discussed. Next to this, there exist some other concepts around the world of the Semantic Web, namely "Web of Data", "Linked Data", and "Open Data". Although every concept by itself has several definitions in its own environment, there has been some discussion when it comes to the question of how every approach "interacts" with each other. Do they influence each other? Do they have mutual influence or enabling? Does one concept fully include the other concept or do they make use of each other? A blog post[1] from Tom Heath gives an impression of the different views (also being entailed by a discussion of its own), picking up on a comment of Tim O'Reilly. This section is intended to pick up on single definitions of all the mentioned concepts and give insights about their arrangement. At its end, a subsumption interprets the various concepts and explains how they are understood for this work.

WEB OF DATA The vision that was introduced in Chapter 1 imagines a World Wide Web that can help to solve difficult semantic tasks that consist of various different components, ranging from temporal constraints like time scheduling to availability checks and so forth. To achieve this, the structure of the Web and its entities needs to support the proposition as well as the querying for this kind of information. The "classic" Web as it is known by now is not suited for these kind of tasks, as it was designed to be a **Web of Documents**. The Web pages and documents contained in this kind of Web were solely intended for humans to read, use, and understand. Links between the documents allow for navigation to make them browsable, so the process of finding desired information sometimes leads to several hops over different Web pages combined with a cumbersome search through each Web page for the desired bits of information. The recombination of those bits is also imposed to the user himself. In this design, the data that encapsulates discoverable knowledge and information is merely contained in data silos that are "hidden" behind their Web presences, and consecutively only accessible through mostly proprietary APIs. As mentioned before, this information is processed by the respective peer and then presented only in human-readable form. This does not enable the mechanism that is described by Tim Berners-Lee's vision, as the acquisition of the desired information out of mainly human-

1 http://tomheath.com/blog/2009/03/linked-data-web-of-data-semantic-web-wtf/ (last visited 03/12/2018)

readable sources is very costly, if possible to be done at all. The Web of Documents needs to evolve into a **Web of Data** that allows computers to retrieve data in a more convenient fashion [154] [32] [81]. Figure 3 illustrates this evolution.

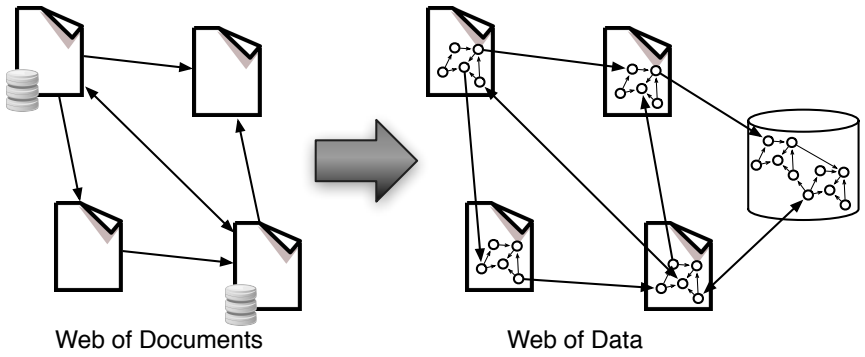


Figure 3: The Evolution of a Web of Documents to a Web of Data.

With the vision and the ideas discussed so far, the design of applications evolves towards making its data and associated information more accessible. Rather than having isolated data silos in combination with Web pages that link to each other, the data is "interwoven" in the Web resource or it is made accessible through specially designed endpoints that are paired with their respective Web page or access point. This makes the Web usable to both humans and computers at the same time. This process has been initiated with the first concepts in 2006 [26]. More sophisticated efforts have been made since then, in order to lift the overall concept of "data" in the Web to a higher position and give it more significance in comprehensive processes, resulting in a new closely related layer interwoven with the classic document-oriented Web. Heath and Bizer [81] comprise the content of the contributions of Bizer, Heath, and Berners-Lee [30] and Mendelsohn [118] by stating that many organisations as well as individuals are applying Linked Data principles for their published data, which leads to the fact that it is not "simply" put on the Web but rather supported in a Linked Data foundation. This eventually constitutes a global data space that is called the Web of Data.

As this work will later on focus on the combination of metadata with multimedia items, it is also interesting to highlight the benefits of those items that are incorporated in the Web of Data. Schandl et al. [146] state that there are two main advantages. The first one focuses on the fact that multimedia data has an important requirement when it comes to a proper metadata allocation. This metadata is used when the multimedia items are utilised, queried, and applied in various occasions like multimedia players. For this purpose, the Web of Data might already support a wide range of available metadata that could be applied conveniently in order to increase the semantic description

of the item. The second point they make is the visibility. Once inserted into the Web of Data, the multimedia item might be discovered over yet unseen (metadata) connections.

LINKED DATA As the statement above about the Web of Data shows, an essential part of making it come true is the utilisation of **Linked Data**. Here again, ambiguity exists about the concrete definition of the concept, or more precisely, which data really is allowed to be entitled as Linked Data [40]. The central argument centres around the question if the given piece of data abides the "Linked Data Design Issues" defined by Berners-Lee, which are namely:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs, so that they can discover more things.

Since their formulation in 2006, these rules have been adopted by a growing community. In general, however, they are only a recommendation that one should follow when producing Linked Data. By not following the defined issues, the benefit and applicability of someone's data in the context of the Web of Data is compromised by a great deal, if present at all. The Semantic Web community would even argue if that data can be called Linked Data altogether (Side note: in the dispute about the definitions, it is also practice by some to call data which is conform to the design issues "Linked Data" with capitalised letters, while data without that claim is denoted as "linked data" [40]).

In [146], Schandl et al. enlist the technologies that are utilised in conjunction with Linked Data. The HTTP protocol and the architecture of RESTful Web Services enable the possibilities to request, manipulate, and transfer resources over the Web, as well as supporting links to other resources. Content negotiation allows to support the data resource conveniently next to a web presence. The actual technical representation of the data is not specified by these two technologies, however there are two recommendations also supported in the Linked Data design issues: RDF for the resource representation and SPARQL for the querying protocol (both technologies will be covered in Section 2.2).

OPEN DATA The third term that will be integrated in the field of the Semantic Web family is **Open Data**. To the best of my knowledge, there is no fixed definition for this term, but in general it addresses

data that is released under an open license and therefore there are no restrictions for someone to use the data freely and in any way envisioned. The concept of Open Data is very often recombined in conjunction with Linked Data under the name of **Linked Open Data**. However, as one can see, there are differences in both terms and it is possible for open data to not be linked, as well as linked data that is not open [40]. As for Linked Data, Tim Berners-Lee also formulated a ruleset that one should follow when producing Open Data. It is implemented as a star system, suggesting the usage of incremental rules. The more rules your data can satisfy, the more stars will be acquired and the more "powerful" and conveniently usable your data gets [26]. The ruleset is formulated as follows:

1. (⋆) Make your data available on the web (whatever format) but with an open licence, to be Open Data.
2. (⋆ ⋆) Make your data available as machine-readable structured data (e.g. excel instead of an image scan of a table).
3. (⋆ ⋆ ⋆) As (2.) plus: Use non-proprietary formats (e.g. CSV instead of excel).
4. (⋆ ⋆ ⋆ ⋆) All the above plus: Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your data.
5. (⋆ ⋆ ⋆ ⋆ ⋆) All the above plus: Link your data to other people's data to provide context.

As one can see, these recommendations are somewhat related or overlapping to the set defined for Linked Data. In a best-case scenario, published data should even align to both rulesets in order to make the best possible contribution to the Linked Open Data (LOD).

CURRENT STATUS When talking about the entirety of the available datasets published as Linked Open Data and their interconnections, it is called the **Linked Open Data Cloud**. Abele, McCrae, Buitelaar, Jentzsch, and Cyganiak, the team and contributors of `http://lod-cloud.net/`, have committed several diagrams that contain all datasets that have been/are currently registered at `https://datahub.io/`, to show the status of the LOD Cloud at different points of time. Figure 4 shows the evolution with illustrations of the years 2007 (May and November), 2008, 2009, 2014, and the central coloured one, which illustrates the current version of 2017[2]. In all diagrams a node represents a dataset, while an edge represents a connection between two datasets, or more precisely, a link between data contained inside the

2 by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/ (last visited 03/12/2018)

given datasets in order to generate context between the data. Through all the illustrated inquiries of datasets, the amount of datasets started with 12 recorded datasets in early 2007, 28 in late 2007, over to 45 in 2008, 95 in 2009, 570 in 2014, and a total of 1139 datasets in 2017, which represents a growth of 100% of linked datasets in the last three years. The colours of the nodes represent the domain that the dataset is attributed to. This reveals that the majority of Linked Data in 2017 lies in the sector of Life Science (red), Linguistics (green), Government (yellow), Social Networking (grey), and Publications (white).

INTERPLAY Now that all concepts have been highlighted in short, Figure 5 illustrates how the interplay between them is interpreted in this work. The insights enlisted about Linked Data and Open Data cover technologies which allow the production of data to incorporate it in the Linked (Open) Data Cloud, as well as sets of best common practices that should be applied when doing so. When complying to these recommendations, the data is best fit to be incorporated into the LOD Cloud and hence it can be discovered and reused by other users in a convenient way. This data can then also be integrated into web presences, like homepages or publicly accessible databases, enabling the data to be not only a part of the classic Web of Documents, but rather lifting the Web to be a Web of Data by interweaving the concepts of both. Last but not least, if all this is combined and the result is a discovery, application, reuse, as well as a playing back of information in order to contribute to a combined and collective knowledge base, talk is about the **Semantic Web**. Further delineations against other concepts such as Artificial Intelligence [114] [137] [65] or Entity-Relationship models [47] can be found in [29].

The main advantages of the applied concepts have been mentioned throughout this section, to sum them up, the following enlistment will outline them in a combined fashion [30] [81]:

- Data that is integrated into the Web of Data can be generic and is not bound to be of a specific domain or field of application. By using RDF as a common standardised data format, any kind of data can be incorporated. In addition to this, the contribution to the Web of Data is not constrained to a defined set of people or experts. Anyone can publish Linked (Open) Data and therefore can make a contribution to a combined knowledge base.

- This freedom could however also bear some disadvantages, as a completely dynamic range of possibilities to model someone's data could lead to chaotic data heaps that cannot be understood by anyone else. Nevertheless, the Linked Data supports the definition of ontologies, vocabularies, and/or schemata (see section Section 2.2 for an explanation) in order to give meaning to the supported data. Because of this and the application of the next

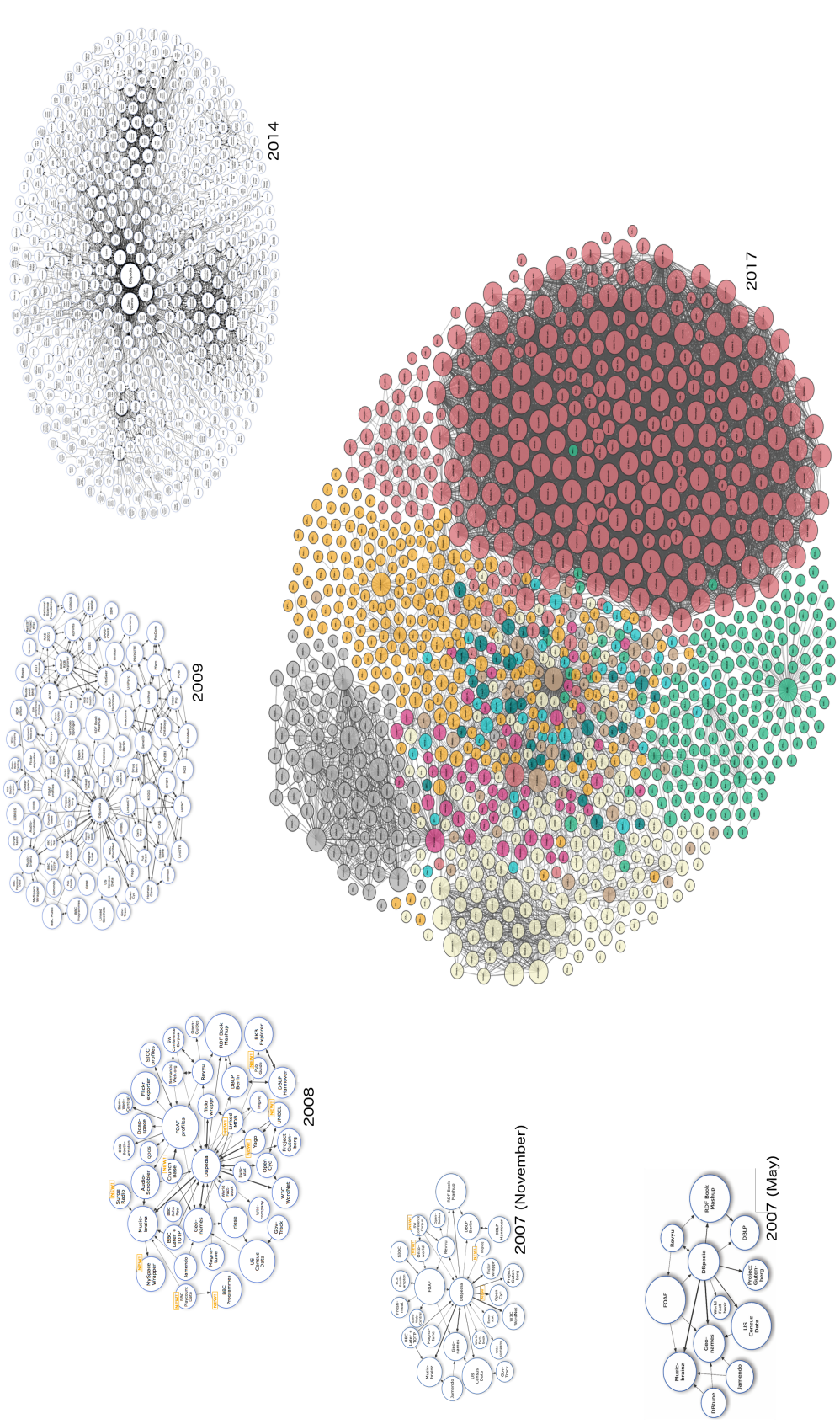


Figure 4: The Evolution of the Semantic Web.

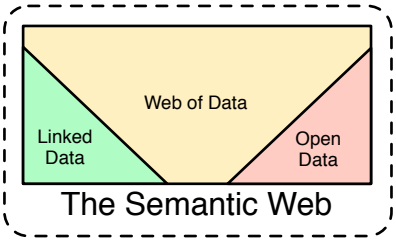


Figure 5: Interaction Between the Concepts Around the Semantic Web.

advantages, the defined data can be dereferenced, supporting further information of how to use it. As a result, the data becomes self-describing.

- As described above, next to the links between documents, the Web of Data also interlinks the data that is contained in it. Because of this, the discovery of knowledge, further information, and data that might be interesting to a user evolves from a self dependant search through the web to an automatic discovery, as links from one resource or a data silo can link to other instances. Semantics can also be added to these links, further enhancing the usability of the discovered linked data.

- There are also advantages from an application point of view. Rather than programming against a fixed endpoint with restrictive requirements, Linked Data relies on the standardised technologies HTTP, RDF, and SPARQL, in order to support a homogeneous way of accessing data. As a consequence, the whole network can be discovered using the same querying mechanisms.

As can be seen, essential cornerstones of the Semantic Web are the Resource Description Framework RDF as well as the corresponding de-facto standard querying language and protocol SPARQL. Because of this, Section 2.2 will give an extensive description of both specifications.

At this point it is important to mention that with the status of this work an awareness is present of the fact that beyond the Web of Data there are other, further advanced concepts of the WWW landscape: the **Web of Things** [183] applying technologies of the **Internet of Things** [178], and even beyond there exist ideas about the **Web of Thoughts** or **Emotional Web** [18] [127], whose vision it is to create a working environment between human and computer via the perceived emotions and feelings of the interacting person.

Just as with the other discussed concepts, the Web of Things does not include a strict definition, but rather promises to implement ubiquitous computing and context-awareness among devices, creating ambient intelligence. This is achieved by enabling devices or objects (this

can be classical devices taking part in the Web like computers, smartphones, tablets etc., but especially everyday devices that were not originally planned to do so, for example temperature scales and refrigerators) to identify, receive, communicate, or populate as well as process data not just for and by itself, but rather in a network of such devices over the Internet in order to comply a useful objective. This process is executed making use of various technologies originating from both "basic" Web technologies as well as technologies of the concepts described above. In addition, long time known and applied technologies like Sensor Networks [6] and Near Field Communication [174] find use in the Internet of Things to further enhance its capabilities. Furthermore, the concept broadens the described functionalities especially in terms of number and the kinds of devices that can take part in such a network. Modern examples are the classically known "intelligent home", supporting useful information to the resident like food stocks that are running low and other processes like turning on the heat when the resident is close to his home.

It is important for this work to delimitate itself against the concept of the Web of Things, which poses the next step of evolution after the Semantic Web. The extractors that will be utilised throughout the following work are not as "intelligent" and ubiquitously working as devices operating in the Internet of Things. They are rather devices that rely on being called, only their internal process is working autonomically. Because of this, the remainder of the work will rather focus on the representation and allocation of passed information, as well as its processing and utilisation, which is to be positioned in the fields of the Semantic Web rather than the Web of Things.

2.2 THE "RESOURCE DESCRIPTION FRAMEWORK" - THE BACKBONE OF METADATA MODELING

The Resource Description Framework RDF is the de-facto standard when it comes to metadata in the Semantic Web. Additionally, RDF can also be utilised when the modelling of information needs to be described. RDF is composed of a family of W3C specifications with the first recommendation issued in 1999, which has become a full specification document in 2004 [113].

There are several advantages posed by RDF, which align and support the advantages of the Semantic Web described in Section 2.1. Through RDF, computer-readable information can be produced that is understandable by other software or computers in a convenient fashion. This enables your own dataset not only to be linked to others, but also be linked to by others as they can understand and use it quite comfortably. Enriched and combined datasets contribute to the collective knowledge of every participant. Additionally, combined data allows for federated queries at one access point rather than ex-

pensive information gathering at various locations. In the pursue of Linked Open Data (see Section 2.1), openly accessible datasets of common use cases are designed and offered for public usage. By not just linking the data but also incorporating persons as well as provenance information, a social network can be established that further enhances data quality and confidentiality.

This section will give a step by step introduction to the **Resource Description Framework RDF**. To achieve this, Section 2.2.1 explains the basic information unit of RDF called a **statement**. Afterwards, Section 2.2.2 will relate the statement to the overall concepts that constitute the metadata implemented by RDF. Therefore, Section 2.2.2 also enlists and explicates the specification documents of the RDF framework that contain the relevant information. Section 2.2.3 then utilises these RDF basics in order to introduce ontology modelling with RDF. A disclaimer that explains the extrinsic structure of RDF graph pictures of this work is also contained. Afterwards, Section 2.2.4 explains the possible contextualisation of RDF data via so-called named graphs. Section 2.2.5 enlists various serialisation formats of RDF data, while Section 2.2.6 introduces the advanced RDF features **inferencing** and **reification** in short. The overall illustration is concluded in Section 2.2.7 by an overview of the **SPARQL** specification, the de-facto standard language/protocol to query and manipulate RDF data.

2.2.1 *Information Units in RDF and Their Structure*

Using the term "structure" to describe information units and their representation in RDF is actually quite exaggerated, as it is in fact pretty straightforward in its basic components. The "smallest" entity in RDF to describe metadata is the **statement**, a construct to convey a fact or piece of information. Therefore, the statement consists of three parts that are corresponding to the concept of a rudimentary sentence of spoken language: *subject*, *predicate*, and *object*. The intended fact is concluded by the combination of predicate and object, which is then related to the subject. The following sentences can be seen as examples (yet abstracting from proper RDF syntax, which will be introduced later):

- Bob is a human.
- Cesar is a cat.
- Cesar is the pet of Bob.
- Bob's surname is "Green".
- Bob is 30 years old.
- Cesar is 5 years old.

The subjects in these statements are *Bob* and *Cesar*, while the objects are *human*, *cat*, *Bob*, (the textual string) *Green*, and two numerical *age declarations*. The predicates that complete the statements between subject and object in this case concern the assignment of a kind of worldly instance, the assignment of a surname, age definitions, and the association of a pet towards its owner.

Statements are either represented in their subject-predicate-object form or, more human-friendly to read, as a graph composed of nodes and edges. Subjects and objects are the nodes, while predicates are illustrated as (directed, from subject to object) edges between those nodes. An example for the first two statements "Bob is a human" and "Cesar is a cat" can be seen in Figure 6.

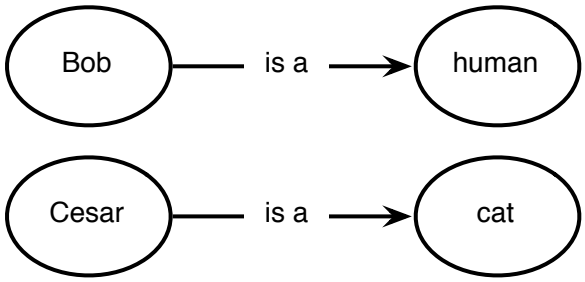


Figure 6: Two Simple Statements Presented as Graph.

These two statements are not related in any way, but when more and more statements are gathered, relations between nodes and statements are developed and a network of information emerges. A graph structure is created eventually when one displays multiple statements using this representation. An example with the statements defined above can be seen in Figure 7. With the addition of the pet correlation between *Bob* and *Cesar*, the statements issued in Figure 6 are connected. The other statements - the ages and the surname - add more meaning to the described knowledge base.

Several important key aspects of modelling information with RDF can already be seen at this point. The nodes of the graph can both serve as subject and object, which allows interconnection of the information holders in a thorough and extensive fashion. The targets of the predicates are of different nature: they can describe simple values like numbers and strings, some kind of instance description of a given individual, and as mentioned before, other nodes of the graph to create connections between two individuals.

This shows in summary, that simple information units like the statements can be used in combination in order to create meaningful accumulations of information. Statements are built with three components - subject, predicate, and object - to convey their information, and can be drawn as a graph using nodes and edges. In order to enhance this

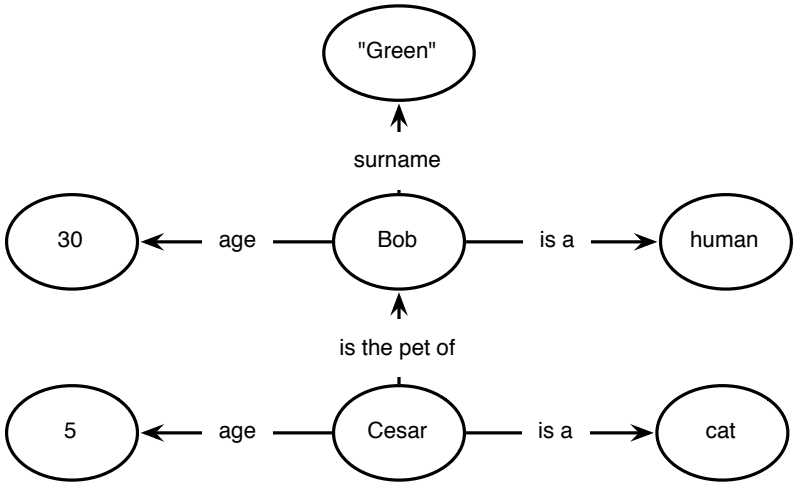


Figure 7: Combined Statements Presented as Graph.

formalism to convey information, certain rules and a schematic approach to produce the information units are necessary. The de-facto standard in the Semantic Web of today is the specification of the Resource Description Framework.

2.2.2 *RDF Basics and Concepts for Metadata Modelling*

The RDF specification supports readers with knowledge required in order to understand and produce RDF data. Additionally, for more in-depth information, further reading is enlisted at various occasions. The specification is divided into several documents with different purposes, namely:

- **RDF Primer**: Gives the reader the basic background about every feature of RDF and explains where further explanations are to be found (original draft issued in 2004 [113], updated to version 1.1 in 2014 [151]).
- **RDF Concepts and Abstract Syntax**: Defines the abstract syntax for all RDF elements, enabling to create and link RDF-based languages and specifications. Also gives more in-depth explanation of the important structures and elements of the RDF specification (original draft issued 2004 [99], updated to version 1.1 in 2014 [52]).
- **RDF Semantics**: This document describes the semantics of the RDF vocabulary, explaining how certain elements and concepts are to be interpreted. Additionally, entailment and inferencing rules are defined (original draft issued 2004 [79], updated to version 1.1 in 2014 [80]).

- **Various Standardised RDF Serialisation Formats**: These formats describe syntaxes that are used to represent RDF data in textual form and therefore formulate RDF graphs. Different syntaxes have varying syntactical output, but are logically and semantically the same when based on the same underlying data. The best-known among the serialisations are discussed in Section 2.2.5.
- **RDF Schema Specification**: The RDF Schema (RDFS) specification is an extension for the basic RDF specification and describes a vocabulary and the process of using RDF to describe own vocabularies. It also describes the already built-in vocabulary of RDF and RDFS (original draft issued 2004 [37], updated to version 1.1 in 2014 [72]).
- **OWL 2 Web Ontology Language**: This suite of specifications (with the overview document [70]) is developed by the W3C OWL Working Group[3] and is the revised and extended version of the OWL Web Ontology Language [76]. Like RDFS, OWL is used in order to design and develop ontologies for semantic data that can be shared over the Web and therefore be understood by computers. To "design" an OWL ontology, the application of different core concepts is necessary. The conceptual structure is defined in the **ontology**, while a **syntax** defines the external structure of the data in order to make it parse- and understandable by computers. On top of these two, a **semantic** defines the meaning of the designed ontology. The OWL 2 specification supports different profiles that are all sub-languages of the core language OWL 2, which allow the user to trade some of the potential expressiveness in favour of computational and/or implementational benefits.

In the following, an explanation of the RDF concepts oriented towards the specifications enlisted above will be given. The focus is laid on the requirements posed by the following work, so not all aspects of the specifications will be highlighted or explained in-depth. Use-case oriented examples will support the process. The explanation will nature from a mix of all enlisted specifications RDF, RDFS, and OWL.

As mentioned in Section 2.2.1, the information units in the RDF context are the statements, consisting of three components: subject, predicate, and object. Accumulations of multiple so called **RDF triples** are called a **graph**. This graph describes information about resources, which are represented by the nodes of the graph. A resource in this case can be anything, from abstract things to physical, worldly things, like people, documents, (historical) events, numbers, textual descriptions, and so on. To achieve this, the RDF specification uses three concepts that can occur in the graph: **IRIs**, **literals**, and **blank nodes**.

3 https://www.w3.org/2007/OWL/wiki/OWL_Working_Group (last visited 03/12/2018)

IRI IRIs (International Resource Identifier [60]) in RDF are used as identifiers for resources. While RDF itself is neutral about the actual text string of the IRI itself, most use cases or vocabularies give them an intent or meaning, with the main purpose of better readability for humans. Additionally, the uniqueness of an IRI should apply to a preferably large domain, given that one major quality of the Semantic Web is to concentrate as much information as possible at one single point. The application of different IRIs for the same worldly "thing" is an actual flaw of the Semantic Web, as the multiple created resources might pose some ambiguity in their representations, and the process of determining if two IRIs address the same "thing" is costly and cumbersome. Sometimes an answer to this is not even possible, as the two representations might also differ in terms of underlying data or information.

As an example the IRI `"http://dbpedia.org/page/Brown_bear"` is simply a text string to describe a resource in RDF, while it will globally be used in order to uniquely address a bear species by everyone in an optimal scenario. Furthermore, from the IRI itself the user can conclude that the resource is hosted by DBpedia[4] and the name of the bear species itself. IRIs extend the allowed characters in regards to the URIs (Uniform Resource Identifier [24]) by allowing all characters contained in the Universal Coded Character Set [90]. An IRI can always be converted to a legal URI, while the other direction may not produce the original/correct IRI. URLs (Uniform Resource Locator [25]) are a subset of the URIs and contain information about the location of a given resource. Consequently, as URLs also are a subset of the IRIs, location information could also be given to IRI resources. In general, it is conventional to support an IRI that can be dereferenced via common HTTP methods in order to display the content of the responding resource.

An IRI can be found at all three positions of an RDF triple, thereby denoting the subject and/or object as well as the defining relationship between them by the triple's predicate.

LITERALS Literals in RDF represent actual values in contrast to IRIs. Examples among them are textual strings ("Barack Obama", "Germany"), numbers ("7", "-0,42"), and dates ("1.1.2017"). In order to be able to be interpreted by computers, a datatype should be associated with the respective literal. The mentioned examples would then be written as an RDF value as follows:

- "Barack Obama"^^`xsd:string`
- "Germany"^^`xsd:string`
- "7"^^`xsd:integer`

4 http://wiki.dbpedia.org/ (last visited 03/12/2018)

- "-0,42"^^xsd:double
- "1.1.2017"^^xsd:dateTime

An extensive list of all supported possible datatypes can be found in the Concepts and Abstract Syntax document of the RDF specification [99] [52]. Textual strings can be related to the language they originate from by adding the associated language tag to the string. These language strings are a subclass of the xsd:string class used above and are added to the string with an "@" character, so the string for "Germany" would be formulated as "Germany"@en in english and as "Deutschland"@de for its german translation. Literals can only be the object of an RDF triple.

BLANK NODES A blank node is used when you need to describe a resource without it having a unique and global identifier. However, the blank node does receive a local identifier in order to make it addressable in the local use case. Blank nodes can be used in the positions of a subject or object in RDF.

Figure 8 shows an example with two blank nodes. As it has been seen in earlier examples, "Cesar" is a cat, and pets in general have a daily requirement on food. For the modelling of this circumstance, blank nodes can come in handy, because the food intake can be modelled in a very generic way. Rather than specifying the exact food that is required for a given pet, general food amounts with various requirements (in this case the distinction between wet and dry food in addition with an amount for wet food) can be created.

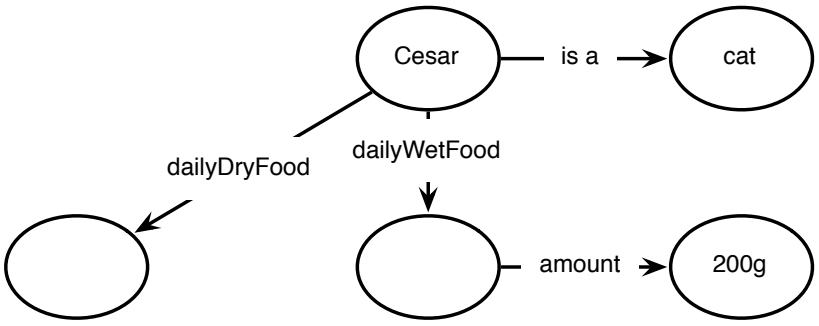


Figure 8: Exemplary Blank Node.

Finer specification of both food types (e.g. the brand) is not necessary, as long as the other semantic requirements are met. This is why a universally defined IRI for both nodes is not reasonable but a local declaration is sufficient.

With the discussed basic components that can occur in RDF triples, these can now be used in order to convert the statements described

above into actual RDF triples with IRIs. Transforming the information content contained in the example shown in Figure 7, the following triples would be produced:

```
<http://petsandowners.org/Bob> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://petsandowners.org/Human>
<http://petsandowners.org/Bob> <http://petsandowners.org/surname> "Green"^^<http:
    //www.w3.org/2001/XMLSchema#string>
<http://petsandowners.org/Bob> <http://petsandowners.org/hasAge> "30"^^<http://www
    .w3.org/2001/XMLSchema#integer>
<http://petsandowners.org/Cesar> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://petsandowners.org/Cat>
<http://petsandowners.org/Cesar> <http://petsandowners.org/hasAge> "5"^^<http://
    www.w3.org/2001/XMLSchema#integer>
<http://petsandowners.org/Cesar> <http://petsandowners.org/isPetOf> <http://
    petsandowners.org/Bob>
```

Listing 1: Triples Contained in the Graph Shown in Figure 7 Formulated as RDF.

The triples of Listing 1 have exactly the same information content as the graph shown in Figure 7. Therefore every line represents one triple consisting of the aforementioned subject-predicate-object structure. Altogether, there are eight IRIs, three literals, and no blank nodes present. There are IRIs originating from an exemplary "pets and their owners" use case, which have identifiers assigned that are preceded with the URI of a homepage "`http://petsandowners.org/`". Furthermore, there are already actual IRIs contained in this example, as "`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`" (in RDF, this directed edge reflects the "is a" relationship that defined the worldly instances of the example) and the datatypes "`http://www.w3.org/2001/XMLSchema#string`" and "`http://www.w3.org/2001/XMLSchema#integer`" are standardised RDF concepts or terms.

NAMESPACES As it can be seen, the IRIs associated in RDF contexts can get quite long and illegible. Because of this, namespaces in conjunction with prefixes are introduced. These serve as abbreviations that represent parts of an IRI, and therefore are better to read in textual representations of the RDF data. Listing 2 shows the RDF data represented in Listing 1 with the utilisation of namespaces and prefixes.

```
PREFIX pao: <http://petsandowners.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

pao:Bob rdf:type pao:Human
pao:Bob pao:surname "Green"^^xsd:string
pao:Bob pao:hasAge "30"^^xsd:integer
pao:Cesar rdf:type pao:Cat
pao:Cesar pao:hasAge "5"^^xsd:integer
pao:Cesar pao:isPetOf pao:Bob
```

Listing 2: Triples Contained in the Graph Shown in Figure 7 with the Addition of Namespaces and Prefixes.

In lines 1-4 of Listing 2 the connections between namespace and corresponding prefix are defined. Line one for example specifies that the namespace "`http://petsandowners.org/`" is abbreviated with the prefix "`pao`". This means that in the following RDF data representation, whenever a term is preceded with the fragment "`pao:`", the IRI for the term would contain the given namespace instead. The same counts for the other three namespaces.

For the following work, several different RDF vocabularies and ontologies (see below) will be utilised. For the sake of clarity, from now on, mainly the abbreviated value of the namespaces will be used. Table 3 in the Appendix shows an enlistment of those ontologies in combination with their respective namespace and prefix.

2.2.3 *RDF Vocabularies and Ontologies*

As already mentioned before, the IRIs used in RDF are generally not enforced to have a semantic associated to them. In practice however, there are collections of IRIs and resources that are used in order to model concepts with a defined semantic for a specific field of application or use case. In the literature, these collections are called an **RDF vocabulary** or **RDF ontology**. The distinction between both terms is ambiguous, often times they are used as synonyms. However the trend is to attribute more meaning towards the collection of an ontology. While vocabularies are rather a mere and simple gathering of words or terms, Berners-Lee, Hendler, Lassila, et al. [27] explain that researchers in Web-related domains and other fields like Artificial Intelligence have adapted the term **ontology**. Following their definition, an ontology is comprised of a document or file that formally describes relations among terms, which is typically a combination of a taxonomy, describing the terms that are contained in the ontology, and a set of inference rules that further specify those relations. Because of this, the semantics behind the described data is presented in a standardised way, allowing computers to "understand" the meanings of semantic data in the Web, as they can apply the rules and terms defined in the ontologies by following the supported links to those ontologies.

The inferencing and reasoning mentioned in the citations above concern the logical conclusion of additional knowledge "hidden" in the ontologies or data that is created obeying to the rules defined in the ontology. with the application of these mechanisms, additional knowledge for given RDF data can be discovered. An example of this could be the modelling of family relations between people. When three people, conveniently named *father*, *son*, and *grandson*, are contained in an information base with the relations referring to fatherhood being present as RDF triples between *father* and *son*, as well as *son* and *grandson*, it is reasonable to assume that there should also be

a triple indicating the relatedness between *father* and *grandson*, even if it is not modelled as an actual triple itself. This RDF feature will not be required heavily in the course of this work and thusly it will only be covered in short in this section. Also, the definition of the terms vocabulary and ontology will follow the definition given above for the remainder of this work.

Making use of the rules and concepts defined in ontologies, an information base can be created in a standardised manner, and therefore it can be understood way easier at another peer, as the semantics of the defined IRIs are fixed. These semantics can also be retrieved by computers. In the example of Listing 1 the IRI "`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`" is such a predefined IRI of the RDF vocabulary. When used at other locations, the standardised semantics of assigning a type relationship can be inferred.

Ontologies themselves are defined as RDF triples. Core concepts in order to create basic ontologies originate from the RDF and RDFS ontologies, and are defined as follows:

- **Class**: Classes are used in order to create categories to which IRI nodes can be associated. The relationship between the Class and the node, which is then called an **instance** of the respective Class, is the type property with the IRI "`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`" already used in the examples above. A class can be enhanced with various requirements which are then passed to its instances by adding properties.

  The IRI for Class nodes is "`http://www.w3.org/2000/01/rdf-schema#Class`".

- **Property or Relationship**: A property or relationship is a relation between two given RDF nodes, creating a semantical association between them. These nodes can be two IRIs or an IRI as subject and a literal as object.

  The IRI for Property or Relationship nodes is "`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`".

- **subClassOf**: This property, associated between two Classes, creates a subclass/superclass relationship between the two associated Class nodes. The subclass inherits all of the properties defined for the superclass. This also enforces, that every instance of the subclass is also an instance of the corresponding superclass.

  The IRI for the subClassOf-Property is "`http://www.w3.org/2000/01/rdf-schema#subClassOf`".

- **subPropertyOf**: This is the equivalent relationship to the subClassOf relation, but for RDF properties or relationships. When

this property is associated between two Property nodes, a subproperty/superproperty relation is created. This means, if a subproperty is created between two nodes, then there is also an association of the superproperty issued.

The IRI for the subPropertyOf-Property is "`http://www.w3.org/2000/01/rdf-schema#subPropertyOf`".

- **domain** and **range**: These two Properties are used to define the allowed Classes that can be the source and the sink of a given Property edge.

  The IRI for the domain-Property is "`http://www.w3.org/2000/01/rdf-schema#domain`" and for the range-Property "`http://www.w3.org/2000/01/rdf-schema#range`".

There are many different RDF vocabularies and ontologies with various use cases and fields of application available in the Web. In some cases, they are so well designed and cover such an important scope that the W3C standardises them. The general rule in the Semantic Web is to use existing vocabularies and ontologies wherever possible, which is especially true for the standardised ones. Whenever a public ontology is used, the barrier to understand produced RDF data is lowered, as all peers agree upon the chosen ontology and its semantics.

Two of the most established ontologies (next to RDF, RDFS, and OWL that are already described and used in examples so far) are the FOAF [38] [68] and SKOS [17] [120] ontologies. The FOAF ontology is designed to describe and link people. Example concepts are the "`http://xmlns.com/foaf/0.1/Person`" class and the "`http://xmlns.com/foaf/0.1/surname`" property associated with a person. The existence of the IRI for the surname already shows bad practice in the examples above, e.g. in Listing 1, as "new" IRIs have been introduced for the surname and age of a person rather than using the ones defined in the FOAF ontology. The SKOS ontology supports concepts to describe, share, and link knowledge organisation systems like thesauri, taxonomies, classification schemes, and subject heading systems.

As a simple example, Listing 3 shows the ontology that could be created for the "pets and their owners" use case. Prefixes and namespaces are used as shown in Listing 2.

```
1  PREFIX pao:  <http://petsandowners.org/>
2  PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4
5  pao:Human rdf:type rdfs:Class
6  pao:Cat rdf:type rdfs:Class
7  pao:Animal rdf:type rdfs:Class
8
9  pao:Cat rdfs:subClassOf pao:Animal
10
```

```
11 pao:isPetOf rdf:type rdf:Property
12
13 pao:isPetOf rdfs:domain pao:Animal
14 pao:isPetOf rdf:range pao:Human
```

Listing 3: Definition of the "Pets and Their Owners" Ontology.

In lines 5 to 7 of Listing 3 the RDF classes used in the specific use case are enlisted. All of them are assigned to the type "`rdfs:Class`". Next to "`pao:Human`" and "`pao:Cat`" used in the examples above, an additional class "`pao:Animal`" has been added to show the subclass relationship formulated in line 8, as a "`pao:Cat`" is defined to be a subclass of "`pao:Animal`". This implies, through semantic inferencing that all properties and relationships that are defined for the "`pao:Animal`" class are also added for the "`pao:Cat`" class. Line 10 specifies that "`pao:isPetOf`" is an "`rdf:Property`", which is then further enriched semantically in the lines 13 and 14 with the addition of a domain and range. These make sure that the "`pao:isPetOf`" edge is always directed from a "`pao:Animal`" instance to a "`pao:Human`" instance. The reasoning process behind all these defined statements of the ontology enable a "`pao:Cat`" to also be a legal source of a "`pao:isPetOf`" relationship.

DISCLAIMER FOR THE TERM "COMPLEXITY" OF SEMANTIC WEB ONTOLOGIES In the remainder of this work, ontologies of the Semantic Web will often be considered in terms of their **complexity**. As the core concept of the ontology is explained in this section, more information can be given about this term. Firstly, ontologies are assessed to be more complex simply by the amount of different "`rdfs:Class`" and "`rdfs:Property`" entities they implement. Secondly, a higher complexity is implemented by the possible kinds of connections between these two entities. This is mainly established by subclass hierarchies, which can evolve both in terms of depth and broadness, and the allocation of properties or relationships to given "`rdfs:Class`" representatives.

A third facet to the complexity, which is not associated with Semantic Web metadata modelling, is later introduced by the Anno4j library in Chapter 4. This library allows to map RDF data to Java objects and thereby uses a so called domain model, which is the technical pendant to ontologies. These domain models can be enhanced by own implemented features, which further increases its assessed complexity.

A more thorough and detailed **complexity measure** for Semantic Web ontologies and therefore also domain models will later be introduced in Section 6.2.

DISCLAIMER FOR RDF GRAPH PICTURES For the rest of this work, whenever RDF triples are visualised in the form of a graph, the rules

enlisted in List 1 in the Appendix are used to draw the graph. They then also imply which RDF feature they illustrate.

2.2.4 *RDF Datasets and Named Graphs*

All previously shown examples have been representing exactly one use case, one collection of information which is represented in RDF. This is called an **RDF graph**. In some cases however, it is necessary or desired to describe this information in a more fine-grained fashion, or to divide the statements contained in one graph into smaller semantically distinctive pieces. Therefore, an **RDF dataset** contains various RDF graphs, divided into (possibly multiple) so-called **named graphs** and up to one **default graph**. A named graph is used to describe a subset of statements of an RDF dataset. To achieve this, each of those statements is associated with an additional IRI that represents the name of the graph. The default ("unnamed") graph on the other hand contains all those statements that are not added to one of the named graphs. Figure 9 shows an example. For the "pets and their owners" use case, a possible application of subgraphs would be to divide the database into different cities, allocating the persons and animals of the database to the city they live in.

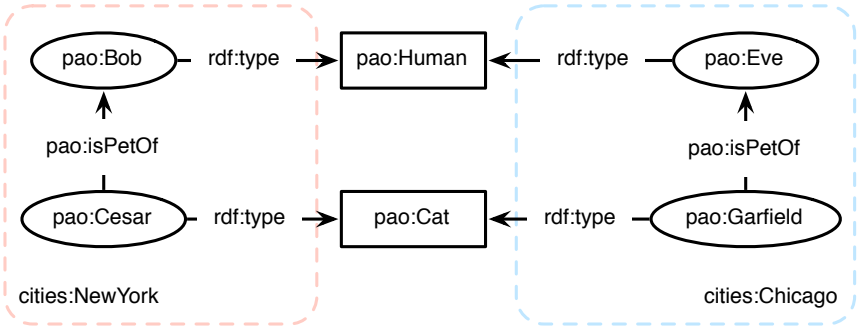


Figure 9: Example RDF Dataset with Two Named Graphs `"cities:NewYork"` and `"cities:Chicago"`. The Exemplary Namespace `"http://examplecities.com/"` is Assumed for the Prefix `"cities:"`.

In the example of Figure 9, `"pao:Bob"` and `"pao:Cesar"` are located in `"cities:NewYork"`, while `"pao:Eve"` and `"pao:Garfield"` live in `"cities:Chicago"`. Both Classes `"pao:Human"` and `"pao:Cat"` used in the example are placed in the default graph, as these are statements and concepts that can not (and should not) be related to one named graph, while the `"rdf:type"` assignment between the various instances and their associated classes is placed in the respective named graphs.

The division into the different graphs for the refinement of the database was primarily aligned with **SPARQL**, the SPARQL Protocol and RDF Query Language [71] (Section 2.2.7 will give more details about SPARQL), which is the de-facto standard querying language for RDF data. In combination, it is possible to fine-tune queries in order to make use of the defined named graphs. In the example shown in Figure 9, a simple query for "*all animals*" (ignoring named graphs) would return both `"pao:Cesar"` as well as `"pao:Garfield"`. Enhancing the "*all animals*" query to only request data from a named graph rather than the whole dataset, will only return the `"pao:Animal"(s)` that are contained in the respective named graph.

2.2.5 *Expressing and Transporting RDF Data - RDF Documents and Their Serialisations*

An **RDF Document** represents the encoding of a single RDF graph or an RDF dataset, as described in Section 2.2.2. Next to the RDF data, a concrete **syntax** or **serialisation format** is also necessary in order to enable others to parse a document. For these syntaxes, the W3C proposed several standards. The best-known are enlisted in this section in short. **N-Triples**, **N-Quads**, **Turtle**, and **TriG** are called the "turtle family of RDF languages" [113]. Additionally, there are serialisations oriented towards the JSON, XML, and HTML standards.

To show practical implementation examples of the various syntaxes, the following graph of an RDF document shown in Figure 10 will be used and presented in the different RDF syntaxes. The until now used exemplary use case of "pets and their owners" is extended with interests of humans. Those interests are already present in another database (referenced to by the IRI and namespace `"http://dbpedia.org/"`), so the respectively existing IRIs are utilised. In this example, "*Bob*" has interest in the novel "*Conviction*", which is represented via the IRI `"http://dbpedia.org/page/Conviction_(Star_Wars_novel)"`. Its "*author*" is referenced via the IRI `"dpr:Aaron_Allston"`, while a "*label*" and "*release date*" is also issued. The prefixes `"dpo:"`, `"dpr:"`, and `"dpp:"` stand for the three namespaces `"http://dbpedia.org/ontology/"`, `"http://dbpedia.org/resource/"`, and lastly `"http://dbpedia.org/page/"` respectively.

N-TRIPLES N-Triples [43] is the simplest syntax for RDF data. It serialises one triple per line which is ended with a full stop. Every IRI is enclosed in angle brackets "<>". The produced output is very bulky, but its simplicity contains all the necessary information in one line respectively, and the line by line notation allows convenient parsing for computers. Listing 4 shows the RDF data contained in Figure 10 serialised as N-Triples.

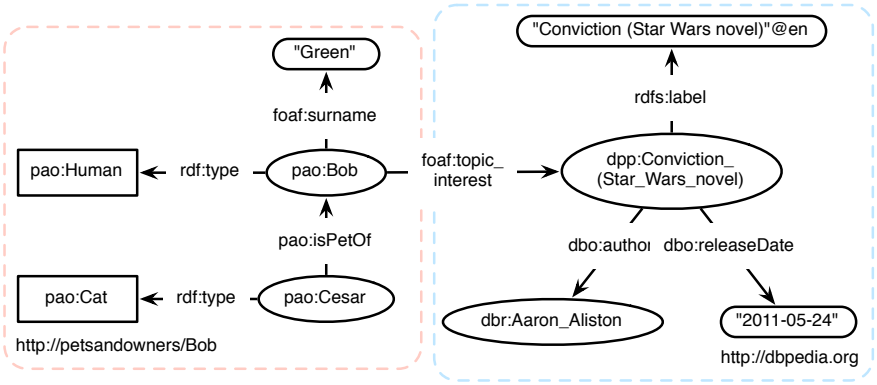


Figure 10: Exemplary RDF Graph Used for Showing the Different RDF Serialisation Formats.

```
<http://petsandowners.org/Bob> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://petsandowners.org/Human> .
<http://petsandowners.org/Bob> <http://xmlns.com/foaf/0.1/surname> "Green" .
<http://petsandowners.org/Bob> <http://xmlns.com/foaf/0.1/topic_interest> <http://
    dbpedia.org/page/Conviction_(Star_Wars_novel)> .
<http://dbpedia.org/page/Conviction_(Star_Wars_novel)> <http://www.w3.org/2000/01/
    rdf-schema#label> "Conviction (Star Wars novel)"@en .
<http://dbpedia.org/page/Conviction_(Star_Wars_novel)> <http://dbpedia.org/
    ontology/author> <http://dbpedia.org/resource/Aaron_Allston> .
<http://dbpedia.org/page/Conviction_(Star_Wars_novel)> <http://dbpedia.org/
    ontology/releaseDate> "2011-05-24"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://petsandowners.org/Cesar> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://petsandowners.org/Cat> .
<http://petsandowners.org/Cesar> <http://petsandowners.org/isPetOf> <http://
    petsandowners.org/Bob> .
```

Listing 4: The Information Content Illustrated in Figure 10 represented as RDF Triples Using the N-Triples Syntax.

N-QUADS The N-Triples syntax does not support named graphs, however its extension N-Quads [42] does. This is implemented for every triple by adding the IRI of the named graph it is contained in behind the subject-predicate-object notation, hence the name "**quad**" instead of triple. This serialisation is also enlisted line by line. Listing 5 shows the N-Quads serialisation for the RDF data contained in the graph shown in Figure 10, using the named graph with the IRI "`http://petsandowners.org/Bob`".

```
<http://petsandowners.org/Bob> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://petsandowners.org/Human> <http://petsandowners.org/Bob> .
<http://petsandowners.org/Bob> <http://xmlns.com/foaf/0.1/surname> "Green" <http:
    //petsandowners.org/Bob> .
<http://petsandowners.org/Bob> <http://xmlns.com/foaf/0.1/topic_interest> <http://
    dbpedia.org/page/Conviction_(Star_Wars_novel)> <http://petsandowners.org/Bob>
     .
<http://dbpedia.org/page/Conviction_(Star_Wars_novel)> <http://www.w3.org/2000/01/
    rdf-schema#label> "Conviction (Star Wars novel)"@en <http://dbpedia.org/> .
```

```
5 <http://dbpedia.org/page/Conviction_(Star_Wars_novel)> <http://dbpedia.org/
      ontology/author> <http://dbpedia.org/resource/Aaron_Allston> <http://dbpedia.
      org/> .
6 <http://dbpedia.org/page/Conviction_(Star_Wars_novel)> <http://dbpedia.org/
      ontology/releaseDate> "2011-05-24"^^<http://www.w3.org/2001/XMLSchema#date> <
      http://dbpedia.org/> .
7 <http://petsandowners.org/Cesar> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
       <http://petsandowners.org/Cat> <http://petsandowners.org/Bob> .
8 <http://petsandowners.org/Cesar> <http://petsandowners.org/isPetOf> <http://
      petsandowners.org/Bob> <http://petsandowners.org/Bob> .
```

Listing 5: The Information Content Illustrated in Figure 10 represented as RDF Triples Using the N-Quads Syntax.

TURTLE Another extension to N-Triples is called Turtle [131]. It introduces some syntactical enhancements to improve the readability for humans. Listing 6 shows the Turtle serialisation for the RDF data contained in the graph shown in Figure 10. The improvements that can be seen contain prefixing (with a basic prefix just using a colon) of IRIs and the aggregation of relationships that refer the same subject. A subject, rather than being entailed by a predicate and object with a full stop to end the line, can now be followed by several lines containing a predicate and object being ended by a semicolon (“`;`”), which means that all of these lines target the same respective subject. The last line still needs to be ended with a full stop. An example for the resource “`:Bob`” (referencing the full IRI “`http://petsandowners.org/Bob`”) can be seen in lines 10 to 13 in Listing 6.

```
1  @prefix : <http://petsandowners.org/> .
2  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
5  @prefix dpo: <http://dbpedia.org/ontology/> .
6  @prefix dpr: <http://dbpedia.org/resource/> .
7  @prefix dpp: <http://dbpedia.org/page/> .
8  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
9
10 :Bob
11     a :Human ;
12     foaf:surname "Green" ;
13     foaf:topic_interest <dpp:Conviction_(Star_Wars_novel)> .
14
15 <dpp:Conviction_(Star_Wars_novel)>
16     rdfs:label "Conviction (Star Wars novel)"@en ;
17     dpo:author dpr:Aaron_Aliston ;
18     dpo:releaseDate "2011-05-24"^^xsd:date .
19
20 :Cesar
21     a :Cat ;
22     :isPetOf :Bob .
```

Listing 6: The Information Content Illustrated in Figure 10 represented as RDF Triples Using the Turtle Syntax.

TRIG As N-Triples needed an extension to support named graphs, it is the same for the Turtle syntax, which does not support subgraphs by itself. TriG [44] is therefore an extension which allows to implement named graphs while using the Turtle syntax for describing the triples. A "GRAPH" keyword surrounds the triples that are to be contained in the given named graph. Listing 7 shows the TriG serialisation for the RDF data contained in the graph shown in Figure 10, the named graph declaration can be seen in line 9 and 21, which span over the triples from lines 10 to 19 and 22 to 27 respectively.

```
1  @prefix : <http://petsandowners.org/> .
2  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
5  @prefix dpo: <http://dbpedia.org/ontology/> .
6  @prefix dpr: <http://dbpedia.org/resource/> .
7  @prefix dpp: <http://dbpedia.org/page/> .
8  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
9
10 GRAPH <http://petsandowners.org/Bob>
11 {
12     :Bob
13         a :Human ;
14         foaf:surname "Green" ;
15         foaf:topic_interest <dpp:Conviction_(Star_Wars_novel)> .
16
17     :Cesar
18         a :Cat ;
19         :isPetOf :Bob .
20 }
21
22 GRAPH <http://dbpedia.org/>
23 {
24     <dpp:Conviction_(Star_Wars_novel)>
25         rdfs:label "Conviction (Star Wars novel)"@en ;
26         dpo:author dpr:Aaron_Aliston ;
27         dpo:releaseDate "2011-05-24"^^xsd:date .
28 }
```

Listing 7: The Information Content Illustrated in Figure 10 represented as RDF Triples Using the TriG Syntax.

JSON-LD JSON-LD [105] is a serialisation for RDF based on JSON. In general, every RDF resource is implemented as single JSON document with a universal identifier (the `@id` element) in order to link from one document to another. A context document can be installed (and again linked to by other JSON documents) in order to support context information like namespaces, details to properties, and so on. As shown in Listing 8, the "`@graph`" element is used in order to implement a whole RDF document. The listing shows the JSON-LD serialisation for the RDF data contained in the graph shown in Figure 10. Also, the context is contained in the same JSON document rather than being linked to via the "`@context`" element.

```
1  {
```

```
  "@context": {
    "pao": "http://petsandowners.org/",
    "foaf": "http://xmlns.com/foaf/0.1/",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "dpo": "http://dbpedia.org/ontology/",
    "dpr": "http://dbpedia.org/resource/",
    "dpp": "http://dbpedia.org/page/",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "@graph": [
    {
      "@id": "dpr:Conviction_(Star_Wars_novel)",
      "dpo:author": {
        "@id": "dpr:Aaron_Allston"
      },
      "dpo:releaseDate": {
        "@type": "xsd:date",
        "@value": "2011-05-24"
      },
      "rdfs:label": {
        "@language": "en",
        "@value": "Conviction (Star Wars novel)"
      }
    },
    {
      "@id": "pao:Bob",
      "@type": "pao:Human",
      "foaf:surname": "Green",
      "foaf:topic_interest": {
        "@id": "dpp:Conviction_(Star_Wars_novel)"
      }
    },
    {
      "@id": "pao:Cesar",
      "@type": "pao:Cat",
      "pao:isPetOf": {
        "@id": "pao:Bob"
      }
    }
  ]
}
```

Listing 8: The Information Content Illustrated in Figure 10 represented as RDF Triples Using the JSON-LD Syntax.

RDFA To incorporate semantic data more conveniently into HTML pages, RDFa [1] [33] (Resource Description Framework in Attributes) allows users to decode the information of triples into their Web page. This enables to extend the already humanly-readable homepage with semantics that can be (automatically) parsed and processed by computers. For RDFa, there exist several implementations, namely the core and lite versions, as well as a version oriented towards XHTML and one towards HTML5. Listing 9 shows an example, containing the RDF data contained in the graph shown in Figure 10. For the sake of clarity, the HTML tags have not been written out as it would be in a normal Web page. It is also important to highlight, that the

"order" of the displayed tags is randomly generated. The important attributes contained in the `<div>` tags are "`resource`", "`property`", "`typeOf`", and "`prefix`". They can also be integrated in other typical HTML tags like links and anchors.

```
<div xmlns="http://www.w3.org/1999/xhtml"
  prefix="
    foaf: http://xmlns.com/foaf/0.1/
    rdfs: http://www.w3.org/2000/01/rdf-schema#
    rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
    xsd: http://www.w3.org/2001/XMLSchema#
    ns1: http://petsandowners.org/
    ns2: http://dbpedia.org/ontology/"
  >
  <div typeof="rdfs:Resource" about="http://dbpedia.org/resource/Conviction_(
     Star_Wars_novel)">
    <div property="rdfs:label" xml:lang="en" content="Conviction (Star Wars novel)
     "></div>
    <div property="ns2:releaseDate" datatype="xsd:date" content="2011-05-24"></div
     >
    <div rel="ns2:author" resource="http://dbpedia.org/resource/Aaron_Allston"></
     div>
  </div>
  <div typeof="ns1:Cat" about="http://petsandowners.org/Cesar">
    <div rel="ns1:isPetOf">
      <div typeof="ns1:Human" about="http://petsandowners.org/Bob">
        <div rel="foaf:topic_interest" resource="http://dbpedia.org/page/
    Conviction_(Star_Wars_novel)"></div>
        <div property="foaf:surname" content="Green"></div>
      </div>
    </div>
  </div>
</div>
```

Listing 9: The Information Content Illustrated in Figure 10 represented as RDF Triples Using the RDFa Syntax.

RDF/XML The RDF/XML syntax [150] is the "oldest" serialisation for RDF data and is, as the name suggests, based on the XML standard. The whole document is encapsulated with the classic "`<xml>`" tag, followed by a "`<rdf:RDF>`" tag that contains the declarations of namespaces as attributes (the "`xmlns`" attribute). Each RDF resource is implemented as a "`<rdf:Description>`" element, its corresponding IRI is associated as "`rdf:about`" attribute. Using these, references to other RDF resources can be established. Every child element of a description represents a triple that is referring to the RDF resource as their subject. Listing 10 shows the RDF data contained in the graph of Figure 10 in the RDF/XML serialisation. Note that the defined namespaces only apply for XML elements and attributes.

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:ns0="http://dbpedia.org/ontology/"
         xmlns:ns1="http://dbpedia.org/property/"
         xmlns:ns2="http://petsandowners.org/">
```

```
8
9   <rdf:Description rdf:about="http://petsandowners.org/Bob">
10    <rdf:type rdf:resource="http://petsandowners.org/Human"/>
11    <foaf:surname>Green</foaf:surname>
12    <foaf:topic_interest rdf:resource="http://dbpedia.org/page/Conviction_(
       Star_Wars_novel)"/>
13  </rdf:Description>
14
15  <rdf:Description rdf:about="http://dbpedia.org/resource/Conviction_(
       Star_Wars_novel)">
16    <rdfs:label xml:lang="en">Conviction (Star Wars novel)</rdfs:label>
17    <ns0:author rdf:resource="http://dbpedia.org/resource/Aaron_Allston"/>
18    <ns1:releaseDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
       2011-05-24</ns1:releaseDate>
19  </rdf:Description>
20
21  <rdf:Description rdf:about="http://petsandowners.org/Cesar">
22    <rdf:type rdf:resource="http://petsandowners.org/Cat"/>
23    <ns2:isPetOf rdf:resource="http://petsandowners.org/Bob"/>
24  </rdf:Description>
25
26 </rdf:RDF>
```

Listing 10: The Information Content Illustrated in Figure 10 represented as RDF Triples Using the RDF/XML Syntax.

For the remainder of this work whenever RDF data is to be illustrated, the graph structure described in the disclaimer in Section 2.2.3 will be utilised, while RDF documents and their triples will be mainly presented in the Turtle syntax, when no named graphs are necessary, and TriG otherwise.

2.2.6 *Advanced RDF Features - Inferencing, Reasoning, and Reification*

In Section 2.2.3, the advanced RDF feature called **inferencing** was mentioned. Inferencing allows systems to find "hidden" semantics or rather implied semantics in a given RDF document, which in terms also deduces new triples. This is done by using a **reasoner**, which applies a defined set of rules to the given RDF data. These reasoners can have a varying range of applied processes and therefore be quite simple and efficient, while others can be composed of a multitude of difficult inferencing rules and hence take longer periods of time to compute the implied semantics. Following these rules, the reasoner can also imply if the supported set of triples is logically correct or if they contradict each other. Most reasoners are implemented using RDFS or OWL. This is a very important feature for RDF, as it enables ontologies to be implemented in a much shorter fashion by only implying the additional semantics (and therefore the implied triples) when the ontology is designed. Nevertheless, for the remainder of this work the required inferred knowledge is rather shallow, so this topic will not be discussed in all its details. In-depth discussion and explanation can be found in [79] [80].

Two of the standard inferencing rules consider the subclassing of defined "`rdfs:Class`" objects, as well as the type checking of the ranges and domains of "`rdfs:Property`" objects. Considering the triples of the RDF document shown in Listing 11, some triples could be inferred by an applied reasoner.

```
1  @prefix : <http://petsandowners.org/> .
2  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4
5  :Cat rdfs:subClassOf :Animal .
6
7  :Animal rdfs:subClassOf :LivingThing .
8
9  :Cesar
10     a :Cat ;
11     :isPetOf :Bob .
12
13 :isPetOf
14     rdfs:domain :Animal ;
15     rdfs:range :Human .
```

Listing 11: RDF Document Used for Simple Inferencing.

The result (containing all logically correct triples) is shown in Listing 12. Via the defined subclassing of a "`:Cat`" being an "`:Animal`", and an "`:Animal`" being a "`:LivingThing`", the logical inference can be deduced that a "`:Cat`" is also a "`:LivingThing`". Its triple is added in line 7 of Listing 12. With this new subclass hierarchy it is also safe to assume, that "`:Cesar`" is not only a "`:Cat`", but also an "`:Animal`" as well as a "`:LivingThing`" (with the triples being found in lines 13 and 14). With the information of Listing 11, "`:Bob`" has no assigned type. However again, through inferencing and the application of the information of the "`:isPetOf`" property, one can deduce that "`:Bob`" has to be a "`:Human`" in order to satisfy the correctness of the semantics of the RDF document.

```
1  @prefix : <http://petsandowners.org/> .
2  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4
5  :Cat
6      rdfs:subClassOf :Animal ;
7      rdfs:subClassOf :LivingThing .
8
9  :Animal rdfs:subClassOf :LivingThing .
10
11 :Cesar
12     a :Cat ;
13     a :Animal ;
14     a :LivingThing ;
15     :isPetOf :Bob .
16
17 :isPetOf
18     rdfs:domain :Animal ;
19     rdfs:range :Human .
20
21 :Bob a :Human .
```

Listing 12: RDF Document Containing Inferred Triples with the Base Triples of Listing 11.

All of the triples of an RDF document that have been dealt with until now, meaning the ones that are materialised initially as well as those that are added by a reasoner, are actual metadata that is used to describe information about a given universe. In RDF, next to this, there is also the possibility to make statements about the metadata and the corresponding triples itself - statements about statements. This is called **reification**. A reified triple can give further information about another triple, such as provenance information or composition details. This does however not imply if the original statement is true or false in the overall context. All actual triples are always considered to be true and "take effect" in the described universe, while reified triples are fuzzy about that circumstance and can therefore be used to model fuzzy information.

To model the reified statement, an "`rdf:Statement`" node uses three relationship - "`rdf:subject`", "`rdf:predicate`", and "`rdf:object`" - to link to three other instances in the RDF document and it therefore consists of the same core concepts like the information units of a normal RDF triple. The statement node can have an assigned IRI or it can be a blank node, depending on whether the statement should be applied to a local or global context.

In the example of Listing 13 "`:Bob`" makes an assumption about the eating behaviour of his cat "`:Cesar`". This statement does rather fit being modelled as a reified statement than a normal statement, as the preference of a pet can be assumed by a human but never really be assured by the pet. In the example, "`:Bob`" "`:assumes`" the statement "`:x`", which in terms asserts that "`:Cesar`" does like "`:DansDryFood`".

```
@prefix : <http://petsandowners.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:x
    a rdf:Statement ;
    rdf:subject :Cesar ;
    rdf:predicate :likesFood ;
    rdf:object :DansDryFood .

:Bob :assumes :x .
```

Listing 13: Simple Reification Example.

2.2.7 *Querying and Manipulating RDF Data - SPARQL, the SPARQL Protocol and RDF Query Language*

Up to now, Section 2.2 has given insights into the RDF specification and the de-facto standard way of modelling, producing, and convey-

ing information in the form of RDF data. What is missing is the procedure of requesting and querying the persisted data. The main instrument to perform this is **SPARQL**, the SPARQL Protocol and RDF Query Language [71]. As with RDF, the SPARQL specification [71] is divided in various different documents that highlight particular key aspects of the language. This section will give an overview of SPARQL and thereby explain these aspects in short. The supported specifications are to be consulted if deeper insights are necessary. In the remainder of this work, whenever this SPARQL explanation does not suffice to understand given examples, further detailed information will be given.

In its basics, every SPARQL query consists of two parts: the `"SELECT"` part specifies which variables are to be incorporated in the query and how the result is to be presented, while the `"WHERE"` part defines graph patterns that are to be fulfilled by the query. The result of a query is a single subgraph that matches the defined pattern exactly or, if multiple subgraphs match, a sequence of solutions. More complex queries may include unions, optional query parts, filters, aggregations, path expressions, and nested queries. The `"SELECT"` part can be exchanged by the `"ASK"` or `"CONSTRUCT"` keywords, which are used to return a "yes or no answer" to a query or to construct new RDF graphs respectively.

Listing 14 shows an example of a simple SPARQL query that could be used for the RDF data of Figure 10. The overall query requests for the books that have been written by the author associated with the IRI `"dpr:Aaron_Aliston"`, and returns the book's title in combination with an aggregated counter of all the people who declare an interest in this book. To achieve this, in line 6 the `"SELECT"` part of the query defines three variables that are to be used. `"?name"` will stand for the title of the book, while `"?fan"` and `"?count"` are used for the aggregation of interested people. These variables are then bound in the `"WHERE"` part of the query. All lines from 8 to 10 are associated with a local variable `"?book"` that represents the node of the book in the defined graph pattern. Line 8 associates the name of the book with the relationship `"rdfs:label"` to the query variable `"?name"`, while in line 9 the same is done with the book's author and the relationship `"dpo:author"`. The difference between line 8 and 9 is that the author is bound to an instance, rather than a variable as placeholder. This restricts the resulting subgraph in regards to the author with the IRI `"dpr:Aaron_Aliston"`. In order to count the interested people in regards of the books, the query variable `"?fan"` has to be connected to the book, which is done by using the inverted relationship `"^foaf:topic_interest"` in line 10. Line 10 could also be written as `"?fan foaf:topic_interest ?book ."` to not use the inverted edge, the results however would be the same. Line 11 groups the respective result to combine the entries for a single book.

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  dpo: <http://dbpedia.org/ontology/>
PREFIX  dpr: <http://dbpedia.org/resource/>

SELECT ?name (COUNT(?fan) AS ?count)
WHERE {
    ?book rdfs:label ?name .
    ?book dpo:author <dpr:Aaron_Aliston> .
    ?book ^foaf:topic_interest ?fan .
} GROUP BY ?name
```

Listing 14: Exemplary SPARQL Query Issued on the RDF Data Contained in the Graph Shown in Figure 10.

As underlying dataset is rather small, the result to the query of Listing 14 is rather simple: there is only one book in the data that is written by "`dpr:Aaron_Aliston`" with the title "Conviction (Star Wars novel)". It has one person interested in it: "`pao:Bob`". Table 1 shows the result of the query.

?name	?count
Conviction (Star Wars novel)	1

Table 1: Result of the Query Defined in Listing 14 on the Dataset Shown in Figure 10.

By adding some more triples to the database, the result can be made more exigent. If the triples shown in Listing 15 would be added and the same "books and their fans" query (see Listing 14) would be issued, the result would look like the one shown in Table 2. A second book with the title "Backlash (Star Wars novel)" was added, with three people having interest in it: Alice, Charlie, and Dennis. Because of this, the result for "`?count`" for this given book is 3.

```
@prefix : <http://petsandowners.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dpo: <http://dbpedia.org/ontology/> .
@prefix dpr: <http://dbpedia.org/resource/> .
@prefix dpp: <http://dbpedia.org/page/> .

:Alice
    a :Human ;
    foaf:topic_interest <dpp:Backlash_(Star_Wars_novel)> .

:Charlie
    a :Human ;
    foaf:topic_interest <dpp:Backlash_(Star_Wars_novel)> .

:Dennis
    a :Human ;
    foaf:topic_interest <dpp:Backlash_(Star_Wars_novel)> .

<dpp:Backlash_(Star_Wars_novel)>
    rdfs:label "Backlash (Star Wars novel)" ;
```

```
22    dpo:author dpr:Aaron_Aliston .
```

Listing 15: Triples to Add to the RDF Dataset of Figure 10 to Create an Extended Dataset for the Query Shown in Listing 14.

?name	?count
Conviction (Star Wars novel)	1
Backlash (Star Wars novel)	3

Table 2: Result of the Query Defined in Listing 14 on the Dataset Shown in Figure 10 with the Addition of the Triples Shown in Listing 15.

Next to this "basic" application of SPARQL, the overview of its specification [71] enlists a series of documents that describe the functionality and the possible scope of SPARQL. Besides the Query Language, which has been introduced in short in this section, there is also an own language that concerns the update, creation, and modification of graphs of an RDF datastore. Other features cover various result formats that can be applied to the results of a query, the federation of queries towards different data sources, entailment regimes that allow queries to be enhanced with inference over RDF and OWL information (see Section 2.2.6), an own protocol for SPARQL as well as a SPARQL service description, and a graph store HTTP protocol.

ROUNDUP - PRINCIPLES OF METADATA MODELING Altogether, this section positioned the concept of the Semantic Web into a landscape of other well-known concepts like the Web of Data and Linked Open Data. All of them are not feasible without metadata, and for describing, producing, and modelling metadata the de-facto standard is the Resource Description Framework RDF. With the W3C specifications for RDF, RDFS, OWL, and SPARQL in mind, introductions and their connections have been given for all of these standards. Special focus has been laid on the process of metadata modelling, as it represents the backbone for ontology design. Overall, RDF and RDFS allow the user to have a general freedom when it comes to metadata modelling, which is a good thing, but when it comes to a defined use case or application scenario, agreed upon sets of terms and concepts are needed. Ontologies therefore are an accumulation of such terms which cover a specific domain and which are shared and accepted among users. Additionally, relationships between such terms define semantics that can be used in order to convey more information about the desired metadata.

One such envisioned field of application is the annotation of multimedia data, which allows the allocation of metadata to multimedia, making it more useful for computers and other systems. The concept of the Web of Data emphasises this point, as through this metadata,

links between multimedia items can be established and data can be provided and accessed openly in a more convenient way. The Web Annotation Data Model WADM is one ontology that covers this use case. The WADM will be explained in Section 3.2. The MICO Metadata Model MMM is an extension to the WADM and broadens the context of the WADM to lift the metadata process to a scope of multimedia items in order to implement whole workflows of metadata allocation. The MMM is presented in Section 3.3 and constitutes a core contribution of this work.

3

COMBINING METADATA MODELING WITH MULTIMEDIA

Chapter 2 introduced the world of metadata in the Semantic Web and gave insights on how metadata is formulated. Alongside the advantages and also some disadvantages that exist in the Semantic Web, several use cases as well as areas of applications come to mind. One of the most important fields of application for metadata modelling lies in the field of multimedia. Media items are tagged with various informational features that range from simpler low-level ones like its title, author, or date of creation, to more sophisticated high-level features like detection results or a segmentation of the given media file into subparts. As with metadata for documents or Web pages, it is of equal importance to model the metadata for multimedia in a thorough fashion, as a rich metadata background enables the media item to be used in a wider context. This also contributes to an improvement in the quality of applications that make use of these metadata [155] [88]. Next to this, it is important that the metadata is produced in a common format rather than proprietary implementations that cannot be used at other instances. This circumstance is even aggravated by the fact that in the Web of today, multimedia content is produced easily, while the possibilities to accurately annotate or to add metadata to the content are rarely present or the result is persisted in a proprietary format, making it invaluable for other peers. On the other side, the same issue holds for processes of finding respective content in the end [78].

The process of applying metadata to multimedia items is also hindered by the following factors [146]:

PLENTY OF DIFFERENT DATATYPES In contrast to documents and Web pages, there exists a larger variety of datatypes for multimedia items. Each of those datatypes can have different unique features that must be treated differently. This is also reflected at the point of querying. In contrast to textual media that can be searched through by mere simple string queries, for multimedia like video and audio this process is more complex, as it relies on difficult comparison operators on the media's features or on the processing of possibly proprietary vocabulary items that describe the metadata [7].

FRAGMENTATION OF MEDIA ITEMS Oftentimes, the point of interest or the most interesting thing of a multimedia item is only present in a fracture of the whole item, for example a fragment

of a picture or a short period during an audio stream. These subparts of the media item need to be addressed in a meaningful and understandable fashion. Additionally, metadata features need to be addressable to exactly that fragment.

STACKING OF MEDIA ITEMS Several multimedia datatypes can be seen as a "stacked" media item, meaning that one item is composed of multiple other media items, or that there are objects "inside" the video that require a metadata description for themselves. For example, with the respective extraction processes, a video can be separated into a number of key shots (images) in combination with its audio track. Hence a thorough metadata acquisition can (and should) not only describe the video on its own, but it can also be necessary to reference components of the video (which, in addition, have other datatypes than the video itself - see point one of this enlistment). Next to the existence of this problem, the process of metadata allocation gets more complex, as the description as well as the corresponding querying of information needs to adapt to the intertwined structure.

QUALITY OF MULTIMEDIA METADATA In terms of multimedia it is also important to have a good metadata quality, which is heavily reflected on the reutilisation of the multimedia items. For example, in a use case where videos are to be queried and streamed from a database, it is important for the metadata to be exact and easily parseable, as the playback of an "incorrect" video (in relation to the defined search criteria) can be costly in various factors like bandwidth and time. To counteract this, it is an essential requirement to have a sound way to allocate the multimedia metadata in an extensive fashion at the first place. Then, querying mechanism need to be as sophisticated as possible in order to match the available metadata.

Hausenblas et al. [78] also list some use cases for metadata acquisition for multimedia. Their examples are the sharing of video clips on a social media network like twitter, the annotation of faces in personal photos, tracking your favourite artists on a platform that tracks the appearance of given artists in shows and reviews of the BBC programme, and a more complex use case in which employees of a company can search for and watch broadcasted videos of meetings held in that same company. Those videos are also fragmented and tagged according to what person(s) are relevant in the specific fragment. Therefore the videos can also be searched through by person names, next to the classic approaches like title, date, and so on.

All of these use cases show the importance of metadata allocation to multimedia, as none of them would be feasible without metadata. Chapter 2 has given insights into the field of metadata modelling and

as with the advantages and examples mentioned in that chapter, implementing an ontology that is able to fit the defined requirements is the way to go also for multimedia metadata. To give a deeper insight into the field of metadata modelling for multimedia and its history, this chapter will first enlist related work in this domain in Section 3.1. Afterwards, the Web Annotation Data Model WADM as one of the most influential standardisations in the fields of multimedia metadata for this thesis will be explained in Section 3.2. Section 3.3 then introduces the MICO Metadata Model MMM - one of the core contributions of this work - which is an extension to the WADM. The WADM supports the baseline for the multimedia metadata modelling, while the MMM add components in order to apply the concept of a **Web Annotation** to multimedia workflow-based processes. Throughout this chapter, requirements for a multimedia metadata ontology and the desired use case will be defined, which are added to the ones enlisted above. After the multimedia metadata models have been introduced, Section 3.3.6 will conclude this chapter by evaluating the proposed MMM against these requirements. The entirety of this chapter proposes a solution to research question 1.

3.1 RELATED WORK - MULTIMEDIA METADATA MODELING

This section will highlight related work in the fields of metadata modelling in terms of models and datatypes, the essential combination with multimedia items as well as the process of metadata allocation to some extent. The beginning will be done by the non-Semantic Web cornerstones that have shaped the history of multimedia metadata, as the metadata allocation most notably in the fields of multimedia objects like video, audio, and images has been an important task for a long time. As mentioned before, the better the metadata background of a given multimedia item, the better and in a broader way the item can be used, queried, or further processed afterwards. It is also important at this point to highlight, that the respective multimedia items and the allocated metadata is not only to be both created and used by experts, but also by amateurs [7]. This also imposes an important requirement for the metadata modelling process, as this must be reflected on the result, which must be usable without further detail knowledge. Afterwards, the focus will shift towards more current approaches which are positioned more towards the Semantic Web, and which are in particular thematically closer to the contributions of this work.

CLASSIC XML STANDARDS, MPEG-7 Various literature explore the management of multimedia assets or metadata, partially in the context of the Semantic Web, and therefore can be considered as related to our topic. One of the most significantly involved groups with im-

pact on this field of research is the **Moving Picture Experts Group MPEG**[1]. This group has been founded in 1988 and since then has released various specifications and projects [107] that deal with multimedia metadata modelling and closely related topics. The group describes its area of work with the "Development of international standards for compression, decompression, processing, and coded representation of moving pictures, audio, and their combination, in order to satisfy a wide variety of applications."

Amongst the outcomes of the MPEG group, the most significant because most relevant for this work is the MPEG-7 standard [115] [156] [46] which was issued in 2002 and focuses heavily on the representation of features that describe a given multimedia item in the form of metadata (in this case, the focus of the MPEG group are videos, images, and audio). In contrast, the preceding specifications MPEG 1 through 4 rather targeted the representation of the multimedia file in combination with various compressed alternatives. Additionally, they implemented better ways of transmitting, storing, as well as retrieving the multimedia information for those representation forms. MPEG-7 then takes the next step and increases not just the information description but also supports interpretation of those, which eventually also enabled computers to better deal with these kinds of information. This also benefits the goal of the MPEG-7: making the multimedia items more useful for eventual human consumption. This is also emphasised by the authors' statement, that the biggest value of this information is only achieved, when it can be conveniently found, queried, and accessed as well as applied and managed.

All of this is established by supporting a rich set of innovative tools that allow a complete description of the media's content. Also, the framework includes ways for a good storage solution, high-performance content identification as well as accurate and personalised filtering, searching, and retrieval. Additionally, the whole implementation is not aimed for one or several specific applications or use cases, but rather concentrates on allowing to generate a rich and generic metadata background that can be used universally, fuelling the interoperability between applications. This is also emphasised by the fact that there is not a single "correct solution" for a representation of features of a given multimedia item, but quite the contrary: there exist multiple valid representations, evolving different valuable possibilities at the point of time when the metadata is used.

These representations are composed by a variety of describable features that apply broad and diverse abstraction levels, ranging from low- to high level features that are combinable in every way. While many low-level features can be extracted (semi-) automatically, the high level features generally require a major amount of human interaction, interpretation, and/or supervision [115]. This MPEG-7 meta-

1 http://mpeg.chiariglione.org/ (last visited 03/12/2018)

data background description is rounded off with a set of basic features that cover classic multimedia information, like video codecs, file types, playback length, etc. At this point it is important to mention that the standardisation only focuses on the description of the metadata itself - both the processes of how it is produced as well as how it is consumed is not included in the standard intentionally. This is done in order to further promote diversification of and between applications.

The cornerstones of the MPEG-7 standard that constitute its multimedia metadata functionality are the following:

FEATURE A characteristic piece of information of a multimedia item.

DESCRIPTOR A descriptor is the representation of a feature, defining both its syntax and semantics.

DESCRIPTION SCHEME Contains descriptors as well as other description schemes and defines relationships amongst those.

DESCRIPTION DEFINITION LANGUAGE - DDL A language that enables the creation of new descriptors and description schemes.

SYSTEM TOOLS There is also a toolset supplied with the MPEG-7 standard, focusing on convenience of utilising its functionality. These cover multiplexing of descriptions, delivery mechanisms, synchronisation of descriptions with content, and coded representations (textual and binary) for storage and transmission.

The standard is divided into several parts, encapsulating different fields of functionality, in order to allow a user only use and familiarise oneself with the respective parts that are fitting to a respective use case. The parts are namely MPEG-7- *Systems*, *Description Definition Language*, *Visual*, *Audio*, *Multimedia Description Scheme*, *Reference Software*, *Conformance*, and *Extraction and Use of Descriptions*. For detailed insights it is referred to the respective specification documents[2].

TOWARDS SEMANTIC WEB MULTIMEDIA ONTOLOGIES In [115], Martinez, Koenen, and Pereira summarise the exemplary use cases of the former requirement document and applications document [55] for the MPEG-7 standard and mention the fitting examples of multimedia editing, digital multimedia libraries, home entertainment devices, searching multimedia content, broadcast media selection, managing (large) content archives, semiautomatic multimedia presentation and editing, and opening up archives to the public.

In [89] Hunter emphasises the importance of multimedia in the Web of today and that it plays an essential part in our lives as it has

2 http://mpeg.chiariglione.org/standards (last visited 03/12/2018)

achieved a pervasive role. With this importance in mind, she also formulates the claim that at that point it is also necessary that not only humans can interact with the supported multimedia items, but computers need to be able, too. This is done by establishing computational interpretation of the metadata that is stored alongside the multimedia items. Next to this, many different communities like geospatial, museum, medical, or educational have picked up on the metadata acquisitions and they combine it with metadata and semantics of their own. In the course of this argument, Hunter also sees the MPEG-7 standardisation as one of the fundamental modules of multimedia metadata modelling and therefore establishes it as her starting point towards computer-understandability of metadata. She claims however, that the process is hampered, as there needs to be a common understanding about the semantics themselves as well as the relationships between them. The XML standard, which is the basis for MPEG-7, has some shortcomings at this task, as it lacks the expressiveness to formulate the respective semantics. This is where the core contribution of [89] begins as she implements an ontology that applies the expressiveness of the MPEG-7 tools based on RDF Schema. The choice of RDFS is motivated by the fact that it better facilitates the interoperability between systems, as RDF is more understandable for computers, although the possibilities of expressing different constraints of the MPEG-7 with XML were already good. However, RDF poses a much better way of expressing semantics and semantic relationships through class and property hierarchies. This enables other ontologies to be combined with the possibilities of the MPEG-7 RDF ontology to cover additional use cases and extend the expressivity even further. Their process of implementing their solution is expressed in DAML+OIL and called **MetaNet**. It was established by utilising a top-down process with reverse-engineering of the XML Schema information of the MPEG-7 standard. Modern techniques supported this process, e.g. the recognition of patterns for generating some of the RDFS information automatically.

Still, about a decade later, the expressiveness and functionality of the MPEG-7 standard is valued highly, as researchers align their work trying to adopt its advantages. In [109], Li et al. mention and highlight the importance of the MPEG-7 standard in the field of multimedia metadata, but they also describe its weak points. They claim that the standard does do a lot in terms of closing the semantic gap by supporting good possibilities to store, manage, and retrieve large numbers of multimedia data on the Web, but there still is a shortcoming when it comes to a fuller semantic description of those, resulting in a fuzzy semantic. They compare this problem to a mismatch scenario, in which a worldly concept can be named with multiple wordings, while on the other hand a word can have multiple semantic meanings. Next to this, Li et al. criticise that with XML reasoning cannot be

done effectively. For their solution, they imply that an RDF ontology poses the best solution, as it can overcome the aforementioned shortcomings. Starting from there, the possible base schemas are enlisted, and OWL DL as the most expressive and rich one is chosen as their basis for an ontology that functionally covers their chosen MPEG-7 parts Multimedia Description Scheme, Audio-Visual, and Classification Schema. It is also stated that temporal as well as spatial relations between video, audio, and image data is extremely important, and that OWL DL does not cover these relational requirements. Because of this, the proposed solution extends the OWL DL ontology to also incorporate ways to express those.

One of the most promising solutions is the **Core Ontology for Multimedia Annotation COMM** [7] [167] [8]. The authors comply with the arguments that have been issued in this section by mentioning that some low-level features cannot be expressed in MPEG-7, and the implementation as XML does not comply to current Semantic Web standards and does not support referencable results. They also address the problem of fragment identification, which is similar to the aforementioned spatial and temporal issue, as well as the problem of semantic fuzziness of supported annotations, wherefore they also show an example of addressing the same semantic concept in three different ways in MPEG-7. To this list of problems it is explained that in the wake of the Semantic Web it is especially important to support Web interoperability for given multimedia ontologies, which would enable the produced descriptions to be linked to other information sources of the Semantic Web to further increase its information content. Next to this, the generated metadata for a given multimedia file must be embeddable into a compound document in order to create complete and sound structures.

What is also very important to learn from Arndt et al. in [7] is that they enlist core requirements that they deem essential for every Web-compliant multimedia ontology. These requirements are the following:

MPEG-7 COMPLIANCE The importance of the MPEG-7 standardisation has been highlighted throughout this whole section. This accumulated experience and the advantages of the standard must be conveniently expressible in a designed multimedia ontology. On top of that, as the existing community applying the MPEG-7 is extensive, there is a lot of existing metadata out there which should be importable into a new ontology.

SEMANTIC INTEROPERABILITY Annotations and metadata is only reusable when it can be shared and applied conveniently at different locations. Therefore, the semantics of a new ontology need to be described sufficiently, so, next to the metadata it-

self, the intended meaning is distributable between different systems.

SYNTACTIC INTEROPERABILITY Annotations and metadata is only shareable amongst different systems when they are implemented in a syntax that is commonly used and applied. In the Semantic Web, these are mainly the serialisation formats that were introduced in Section 2.2.5.

SEPARATION OF CONCERNS A clear separation of domain knowledge from administrative or structural knowledge fuels the understandability and reusability of according annotations. Both of these domains as well as their connections need to be implemented in a new multimedia ontology. An example here would be the recognition of an animal (domain) vs. the allocation of the fragment that the animal was detected in (structure).

MODULARITY Modularity in the design of an ontology can decrease execution overhead and lets new users apply the ontology in an easier fashion, as they might not need its whole functionality.

EXTENSIBILITY An ontology should be extensible in a way that new features or implementations can be added conveniently without having to alter the whole core structure of the metadata model. This feature is especially important in a domain that evolves quickly like the domain of multimedia, as for example new multimedia features and extraction processes are continously added.

With these requirements in mind Arndt et al. design their multimedia ontology that promises to deal with all the shortcomings mentioned above, while also satisfying the defined requirements for multimedia ontologies. COMM covers all the descriptors and accumulated knowledge that is present in the MPEG-7 standard and for convenience reasons, also aligns with its naming conventions, while also supporting an extensible implementation that allows the addition of new multimedia features. The ontology is implemented in OWL and is based on the **Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)** while also incorporating various design patterns like *Descriptions & Situations* and *Ontology of Information Objects* [7].

Next to a sound and rich ontology for multimedia, the authors also cover another very important, and often forgotten, point that goes beyond the use case of metadata modelling by itself: both the way of technically creating associated annotations as well as utilising existing ones. These points often pose difficulties, as a plain ontology does not support them necessarily. To counteract this, in [167] Vacura et al. present a Java API named **COMM API** that was developed to

support the infrastructure for the COMM ontology. This API will be referred to in Section 5.1.

STRUCTURAL INFORMATION FOR MULTIMEDIA METADATA As seen until now, the related work mostly covered the eventual metadata that is to be allocated to a multimedia item, the representation and interpretation of the features that are extracted. However, there is another vital information that is necessary for a thorough multimedia metadata inquiry: the proper representation of the multimedia item itself in combination with the "contact points" of the metadata. This was already mentioned by Arndt et al. with their requirement of *separation of concerns*. Saathoff and Scherp extend this requirement further, as they see it as a major factor for interoperability between different multimedia applications and systems [138] [139] [62] [147]. They extend the mentioned requirement by adding following criteria alongside with the introduction of concepts for *Information Objects IO* (the actual piece of information that should be conveyed to the viewer of a multimedia item) and *Information Realisations IR* (the form of representation of an IO). The separation between these two concepts is emphasised, as it is important to give both of them proper attribution in the process of metadata allocation. A thorough metadata model needs to be able to annotate both the objects and realisations in a proper, shareable, and understandable fashion, as this is important for the interoperability between systems. These annotations also need to cover the full range from low-level to high-level features. Last but not least, a meaningful decomposition of the multimedia item as well as the object and realisation levels needs to be supported by the model. This further increases the granularity of the metadata background and is essential for precise annotations, which will later on also affect querying capabilities. All of these requirements and innovations are implemented into the **Multimedia Metadata Ontology M3O**, which is an ontology based on RDF that is not restricted to the "classic" multimedia items such as video, audio, image, etc., but can rather also characterise more complex ones. Additionally by applying Semantic Web technologies, the ontology is open for every kind of information from other ontologies, allowing it to implement rich metadata annotations and semantic information content.

LESSONS LEARNED In the modern Web-oriented life of today and platforms like YouTube and Facebook - to mention only a few - multimedia plays a central role and has a vast range of use cases and applications. To make them as useful and convenient for humans, the metadata that accompanies the multimedia and therefore the multimedia metadata modelling becomes an important step towards the usage of multimedia. The modelling has been an important topic in the fields of computer science for a long time, but still has not stopped evolving

and enhancing further. Its main goal however has not changed since its beginnings: making multimedia as good as possible for eventual human consumption. This process also incorporates a necessary step to increase the interoperability and applicability of the metadata for computers and software in between, as computers in general play a big role in the metadata allocation, and therefore the metadata must also be computer-understandable before it can become valuable to humans in the last instance. Better multimedia metadata equals better usability for computers equals better applicability for humans.

To enable this, the related work shown in this section has a lot of insights in terms of the multimedia metadata. Many requirements and rules have been explained in order to generate "good" metadata or even a whole ontology in order to make the most out of it for the scenario explained above.

Therefore, a dynamic, modular, as well as generic implementation is essential. This is motivated by the broad range of inputs in addition to use cases that need to be modelled in the scope of the model. The first dynamic factor that comes in is the multimedia datatype that is to be backed up by metadata. Next to the commonly known items like video, audio, images, text, etc., it has become more and more popular to have combined multimedia as well as more complex types. Secondly, every of those multimedia types comes with a huge set of diverse multimedia features that need to be modelled in a thorough and non-fuzzy fashion. Thirdly, there is not a single valid metadata representation of a given multimedia item, but multiple ones, that might even originate from different other ontologies. A rich ontology must also be able to incorporate interoperability not only between computers, but also other ontologies.

Last but not least, the research field around multimedia metadata modelling has also evolved in terms of underlying as well as applying technologies. While the still existing milestone and goto standardisation of the MPEG-7 was implemented in XML, more current approaches have moved to RDF and Semantic Web technologies to apply their advantages in the domain. In terms of applying technologies, next to supporting a rich ontology, developers often also support APIs and other applications to make their ontology and metadata more accessible.

3.2 THE WEB ANNOTATION DATA MODEL

The **Web Annotation Data Model WADM** supports an extensible, interoperable framework as well as an accompanying RDF ontology for the design and implementation of so called **Web Annotations**. This term is defined as a piece of further description (e.g. marginalia or highlight) for a digital resources like a comment or tag on a single Web page or image, or a blog post about a news article. Annotations

are used to convey information about a resource or associations between resources. By being an RDF ontology, the WADM already covers a lot of the requirements of the use cases defined in Chapter 1. Further requirements for multimedia ontologies have been defined in Section 3.1, which will be evaluated in Section 3.3.6 in order to show that the WADM is well suited as the baseline for the workflow-oriented multimedia metadata ontology.

The WADM has been designed by the W3C Web Annotation Working Group[3] and has reached its final state on 23 of February 2017 [143], as the group is now closed[4]. The starting point for the implementation of the WADM has been the preceding model called the Open Annotation Data Model OADM [142], which is a contribution of the W3C Open Annotation Community Group[5].

The WADM is enclosed in a suite of specifications that are all implemented in order to support the afore mentioned model and framework. While the model specification focuses on the purpose and the functionality of the model, the Web Annotation Vocabulary document [144] gives detailed information about the classes, relationships, properties, and named entities that are utilised in the WADM. Last but not least, the Web Annotation Protocol WAP [141] describes a client-server based protocol that can be implemented to utilise the Web Annotations in a Web-centric fashion following REST best-practises.

The description of the WADM will not cover every aspect of the model and ontology in-depth, but rather tries to give an overall explanation in order to understand the concept of the Web Annotations as a basis for the following extension called the **MICO Metadata Model MMM** in section Section 3.3. Therefore, this section will stick to a more high-level explanation rather than specifying the concepts formally, which is done by the WADM specification. The MMM section will then pick up explanations given here and, if necessary, fill the overall Web Annotation concept up with more detailed information.

The WADM specification is presented with a series of examples that are implemented using a JSON representation that is mapped to the respective RDF items. In this work, the examples will continue to be illustrated as a combination of graph pictures accompanied with the respective RDF items explained in text, as it has be done in Chapter 2.

3.2.1 *Web Annotation Structure*

The WADM enables a modular way of implementing the two core components of a generalised description of something: the **thing** that

3 https://www.w3.org/annotation/ (last visited 03/12/2018)

4 This is the version of the WADM that all of the following work is based on. If there will be upcoming changes to this state, no changes will be incorporated.

5 https://www.w3.org/community/openannotation/ (last visited 03/12/2018)

is to be described and the **statement** that is supposed to be about or applied to the thing. Therefore, a Web Annotation consists of three core concepts that are modelled extensively through various resources in the WADM: the **body** component of an annotation stands for the statement, while the **target** component delimits on which resource (or subpart of a given resource) the statement is about. The WADM states that "the body is somehow about the target". Both of them are connected by an **annotation** node which can further be extended by provenance information.

Figure 11 shows an example, which depicts the showcase of assigning or recognising a person's identity on a given picture (and even more precisely, only in a defined fragment of the picture), in this case, *Barack Obama*. Therefore, the "`annotation`" node connects two other resources for the body and target, yet abstracting from a proper definition for both of them, which will be filled in later on in this section. The RDF type for annotation nodes is the class "`oa:Annotation`" (with the namespace "`oa:`" standing for "`http://www.w3.org/ns/oa#`"), whilst the resources for body and target are connected with the annotation node via the relationships "`oa:hasBody`" and "`oa:hasTarget`" respectively.

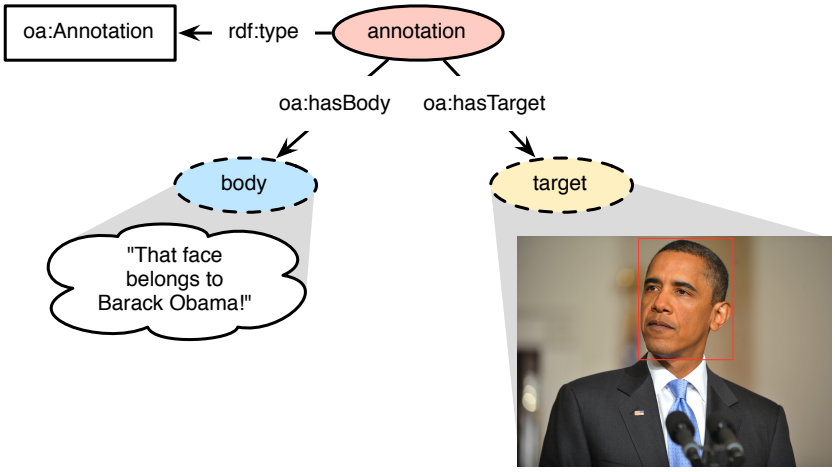


Figure 11: Exemplary Web Annotation Illustrating the Information That the Human Face Displayed on a Given Picture is Barack Obama.

In general, an annotation node is enhanced with a single body and a single target. This bears the semantics, that the statement contained in the body is exactly concerning the single resource that is described by the target node. There is however the possibility of having no body and multiple bodies and/or targets. There must always be at least one target associated with an annotation. The semantics of not having a body can for example be a simple highlighting. When there are multiple instances of both sides, then every body is equally assigned to every target. An example for this could be an image showing multi-

ple animals with the body assigning a species to those animals while multiple targets refer to the fragments of the picture where the animals are present.

At this point it should be mentioned that there are possibilities to include other semantics for the multiplicities. The only officially accepted and included variation is the "`oa:Choice`" element for the body component of an annotation. A choice contains several possible items from which only one, the best fitting, choice is to be picked. An example for a choice body is the same textual comment in different languages. The other multiplicity possibilities for both body and target (that have not been included in the WADM specification as they lacked proper implementations) are an unordered list ("`oa:Composite`"), an ordered list ("`oa:List`"), and an independent collection ("`oa:Independents`"). All of these multiplicity constructs are convenient to implement on the RDF side, but the semantic interpretation poses various difficulties.

3.2.2 *RDF Classes for Body and Target Components*

The resources associated with both the body and target component of a Web Annotation are in general **External Web Resources (EWR)**. In the WADM, an EWR refers to a resource that may also exist outside of the local host system, so when its IRI is referenced, the underlying representation of the resource is presented. A target must always be an EWR, while there is the possibility of a body being directly embedded into the annotation, which generally represents simple textual annotations. Although it is not obligatory, it is highly recommended by the standard to assign RDF classes to both the target and body component of an annotation in order to associate the datatype that is addressed by the annotation. This issue is already reflected in the introduction of Chapter 3, as this datatype association both affects the quality of produced metadata as well as the possible querying of relevant multimedia, as both are enhanced when associated datatypes can be used.

There are the standard and classically known datatypes supported with respective RDF classes in the WADM specification, namely image ("`dctypes:StillImage`"), video ("`dctypes:MovingImage`"), sound ("`dctypes:Sound`"), text ("`dctypes:Text`"), and dataset ("`dctypes:Dataset`"). All of them, as it is best practice, are adapted from the Dublin Core specification called **DCMI Metadata Terms** [54] with one of its namespace abbreviations "`dctypes:`". As there are various sub-types among these genres, a more detailed description can be given by supporting the format as well as the language of the given web resource via the properties "`dc:format`" and "`dc:language`" respectively ("`dc:`" is also a namespace of the DCMI Metadata Terms specification). With the format for example, a given image can further

be specified to be a "jpeg" or "png", while the language can help by illustrating that a given audio file contains spoken Spanish language.

Figure 12 extends the example Web Annotation shown in Figure 11 to have more metadata information. The datatypes can be assigned to both body and target component in order to illustrate the fact that the annotation is written text and targets a given image. This information can directly be used at querying point of these annotations, so only desired information will be transferred. Important to note here is that it has been abstracted from the assignment of a fragment on the picture (which is illustrated in Figure 11) and that the respective resources for the **annotation**, **body**, and **target** have simple IRIs for reasons of clarity. Especially the body and the target should have descriptive IRIs which can be dereferenced in order to retrieve the text passage or the image respectively.

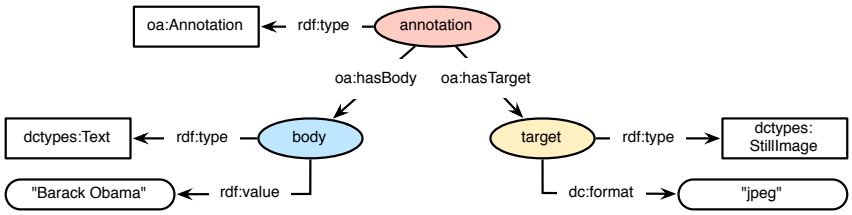


Figure 12: Exemplary Web Annotation from Figure 11 with more Metadata Information, Especially the Addition of the Datatypes Text and Image.

The use case of supporting simple text comments to other multimedia, as shown in Figure 11 and Figure 12, is one of the most common and most used application of annotations. For those textual comments, the WADM supports simpler possibilities to model the same circumstance, namely the **Embedded Textual Body** and **String Body**, in order to match the frequency of text annotations. This convenience is then also reflected in the processing of those annotations as they are supported in standardised fashion. Next to this fact, both the embedded version and the String body version can be created together with the respective Annotation and do not require a Web Resource of its own.

The embedded version is associated with the RDF type "`oa:TextualBody`", which is to be interpreted with the same features as a "classic" text annotation like in the examples described above. The embedded body can also have the properties "`rdf:value`", "`dc:format`", and "`dc:language`". A simpler text annotation can be implemented by the String body version, which is a String literal that is connected to the annotation node via the property "`oa:bodyValue`". This String may only be of the type "`xsd:String`" and there may not be any language tag assigned. The associated Annotation is to be interpreted as if it would be an Embedded Textual Body Annotation with the for-

mat "text/plain" by the client. Figure 13 shows an example of both an embedded text annotation as well as an String body implementation. The annotations both show the example already presented in Figure 11 and Figure 12. Note that both annotations should be interpreted the same way when parsed by a client.

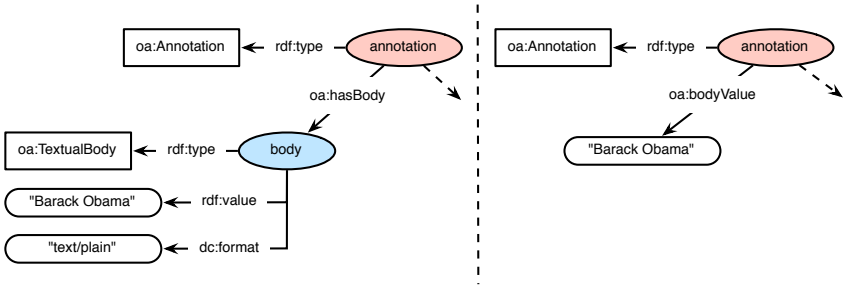


Figure 13: Exemplary Embedded Textual Body Annotation (Left) and a String Body Annotation (Right). Both Represent the Same Semantic Content. Target Components Have Been Left out for Reasons of Clarity.

3.2.3 *Fragmentation of Resource IRIs and Selectors*

In some annotation instances it is necessary to not address an entire EWR, but rather subparts or fragments of it. The WADM handles this issue with two possible ways of specifying those subparts: the **segmentation** of the resource and the WADM **Selector** classes. Both possibilities can be used at the body and/or target component of a Web Annotation.

The segmentation in the WADM is applied as defined by Jacobs and Walsh in [94]. Following their definition, the segment is defined by an IRI in combination with a fragment component, which describes both the extracted content as well as a way of how the segment is to be extracted and applied. There are various possible segmentations that can be applied to different multimedia types [163]. There are however some restrictions that arise with the application of segmentation features. It is important, that the media type is also supported for the given resource, as segmentation fragments might overlap syntactically between different media types. Next to this, the WADM specification mentions that the combination of segmentation with other features that describe the segment in more detail is not possible. Also, supporting a segment fragment does not imply automatically that the respective client will apply the segment, as they can simply ignore it.

One of the best known segmentation standards is the **W3C Media Fragments URI 1.0** specification [165], defining a best-practice to construct and interpret spatial and temporal fragments on video, im-

age, and audio media files. In order to address only the face region of the fragment of the image shown in Figure 11, a spatial fragment conform to the Media Fragments specification could be applied. Figure 14 shows how the fragment would be applied to the resource IRI of the target node, extending the normal IRI "`http://example.org/pictures/barack.jpeg`" with the fragment identifier "`#xywh=pixel:1340,25,1040,1320`", which corresponds to a selected rectangle region starting with its top left corner at the pixel with coordinates "`(1340,25)`", having a width of 1040 pixels and a height of 1320 pixels.

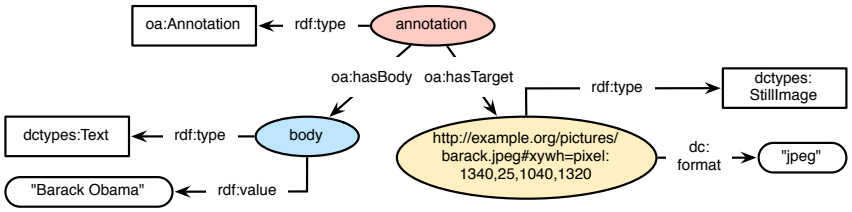


Figure 14: Exemplary Web Annotation Showing the Content Illustrated in Figure 11 with the Extension of a Segmentation Fragment for the Target Image.

Up to this point, the most essential parts of describing a Web Annotation have been mentioned. With these pieces of information, the baseline is set and a broad range of applications can be covered. However, there exist limitations to this baseline of Web Annotations. In [143] it is stated, that even simpler things like selecting a rectangular fragment or selecting a span of text in a HTML page is not possible, as the segmentation is limited to the possibilities that are supported by the segmentation standards. Additionally, there are some use cases that require "non-fragmental" information that also needs to be applied to a given Web Annotation.

To respond to this circumstance, especially in the case of applying segmentation to the resources that are targeted by both the body and target component of a Web Annotation, the WADM implements an instance that can be used to give further details and information about how a Web Annotation is to be interpreted and how the resources of the given annotation are targeted. This is done by using a **Specific Resource** as an intermediate resource between the annotation node and the body or target node respectively. Further details can then be added to those Specific Resources, and then be interpreted at the point of querying. The Specific Resource node is connected to the annotation node via the relationships "`oa:hasBody`" and "`oa:hasTarget`" respectively, while a link to the media resource that is further described is established with the relationship "`oa:hasSource`" from the Specific Resource to the media resource.

Specific Resources can be implemented to be an External Web Resource themselves, however classically they are implemented as incorporated RDF node to the Annotation, as the Specific Resource is only meaningful in combination with the given media resource, so it would never be queried alone, and in order to keep the complexity lower than necessary. Figure 15 shows an example with the use case of a face assignment from above, which utilises two Specific Resource nodes at the body and target component of the Web Annotation respectively. To highlight that Specific Resources are not enforced to be External Web Resources with an own dereferencable IRI, the media resources of the body and target are indicated to have an IRI at the domain starting with "`http://example.org/`", while the two Specific Resource nodes are entitled with a local IRI like the annotation node.

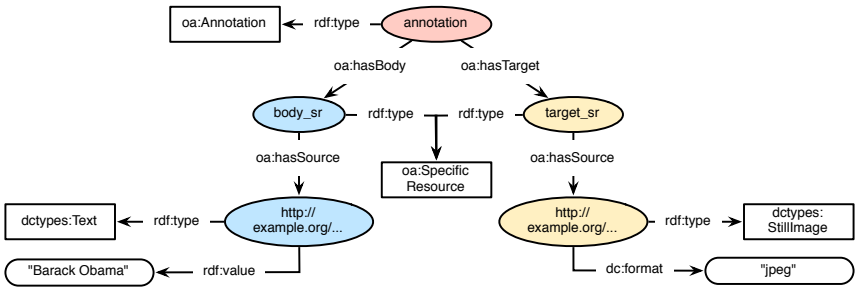


Figure 15: Exemplary Web Annotation with the Application of Specific Resources Both at the Body and Target Component.

With the implementation of a Specific Resource, a more sophisticated version for segmentation purposes is possible, which can overcome the shortcomings of the direct fragments applied to IRIs of the media resources mentioned above. This is done by using **Selector** nodes of the WADM specification, added to Specific Resource nodes as the more detailed version of the respective media resource. The Selector is dependant on the media resource it is attached to, and it describes how the segment of the media resource that is to be described is to be determined. Selectors are connected to their corresponding Specific Resource via the relationship "`oa:hasSelector`". The WADM supports several different selectors (in which mainly the FragmentSelector will be covered in more detail, as it is the most important one for the following work) by default, which are the following:

FRAGMENT SELECTOR The fragmentation process described above that is applied to an IRI of a given media resource is among the best known procedures of describing segments of the resource. This is also possible by the utilisation of a **Fragment Selector**. The actual fragment String is supported via the property "`rdf:value`". This Selector should be enhanced by supporting the specification that the given fragments conforms to by adding the property "`dcterms:conformsTo`" to the Selector.

CSS SELECTOR This selector supports the common ways of selecting elements of an HTML Document Object Model through the utilisation of CSS Selectors. The CSS expression is added via the "`rdf:value`" property to the Selector.

XPATH SELECTOR Next to the CSS selection, applying XPath queries to documents that utilise the Document Object Model like XML and HTML, also pose a comprehensive and flexible way of selecting subparts of those documents. The path is added via the "`rdf:value`" property to the Selector.

TEXT QUOTE SELECTOR This Selector selects text excerpts by supporting a direct text quote via the property "`oa:exact`". The selection can be enhanced by also specifying a textual prefix and suffix that should appear directly before and after the supported text quote respectively. Those are added via the properties "`oa:prefix`" and "`oa:suffix`" to the Selector.

TEXT POSITION SELECTOR In contrary to the Quote Selector, the TextPositionSelector specifies its targeted segment of text by supporting the start and end position of the characters of the given text that are to be selected. This is done by adding the properties "`oa:start`" and "`oa:end`" to the Selector, which indicate the start and end position respectively. Position 0 would be in front of the first character of the text, position 1 in front of the second one etc.

DATA POSITION SELECTOR This Selector is similar to the TextPositionSelector as it uses the same properties, but rather then selecting text, this selector does work on bytes in bitstream.

SVG SELECTOR A Scalable Vector Graphic can be utilised in order to express various forms and shapes to select a fragment of a given media file. The SVG can either be embedded directly into the annotation via the "`rdf:value`" property, or implemented as an own External Web Resource.

RANGE SELECTOR Although the possibilities for selecting a specific segment or fragment with the given Selectors is rich, there are still some use cases that need finer granularity for the selection. Because of this, a RangeSelector can be applied that selects the range between a starting Selector and an ending Selector (but not including it). These are added to the RangeSelector via the relationships "`oa:hasStartSelector`" and "`oa:hasEndSelector`" respectively. As an example[6], the WADM mentions the selection of two adjacent cells in a table of a Web page that are selected through two XPath queries.

6 https://www.w3.org/TR/annotation-model/#range-selector (last visited 03/12/2018)

Figure 16 shows an exemplary Web Annotation with an implemented FragmentSelector. The use case is the same face recognition as shown above in Figure 11, so the depicted annotation does have the same semantic content. The body content of the annotation is not to apply to the whole picture but rather the rectangular selection of the defined fragment of the target component with its Selector.

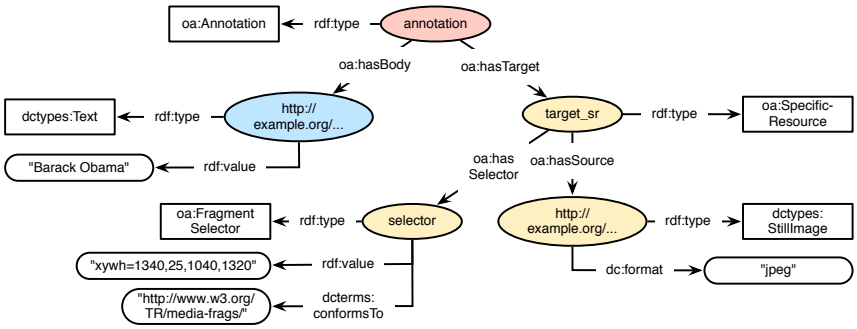


Figure 16: Exemplary Web Annotation with the Application of a FragmentSelector that Utilises the Same Fragment of the Media Resource as the Web Annotation in Figure 14.

Next to the Selectors themselves, the WADM supports ways to combine and further specify the selection process in order to fine-tune it even more to a given use case. *Multiple Selectors*, selecting the same fragment in different ways, can be assigned to a single resource in order to provide diverse alternatives to query that specific fragment. Furthermore, there is the *refinement* of selectors. By adding a refining Selector to a given selector, the resulting fragment is done by a selection of a selection. This is especially useful, when "stacked media types" are the resource that a selection is to be done on. Imagine that the exemplary picture of Figure 11 is contained inside a Web page. Addressing the same fragment could then lead from selecting a (sub-) part of that given Web page, which is then refined by selecting the picture to eventually selecting the fragment inside the picture with three stacked and refined selectors. A refining Selector is added to its original Selector via the relationship "`oa:refinedBy`".

3.2.4 *Additional Information for Web Annotation Nodes*

In the following, additions to the "`oa:Annotation`" node are enlisted that generally answer questions about the origin, context, and purpose of the given Web Annotation. This information constitutes the main provenance information of the WADM. The features are the following:

LIFECYCLE INFORMATION This point is closely related to the **Agents** enlisted in the following bullet point, as the lifecycle informa-

tion implements the assignment of who created and/or modified the Web Annotation and at what point of time that action took place. In the WADM specification, there are two important timestamps specified: the *creation* of the annotation resource and the *generation* of its serialisation. Both of them can have an Agent assigned via the relationships "`dcterms:creator`" and "`as:generator`", as well as the timestamps via the properties "`dcterms:created`" and "`dcterms:issued`" respectively. The last point of time at which a given Web Annotation has been modified is persisted as a property "`dcterms:modified`" at the annotation node.

AGENTS As mentioned before, the Agent associated with a Web Annotation is the instance that is responsible for its creation or serialisation. Since the creation of Web Annotations is a process that can not only be done by humans but also computers, a better description - beyond supporting only an IRI - of the Agent is necessary. The RDF classes to perform this enlisted in the WADM are "`foaf:Person`", "`foaf:Organization`", and "`as:Application`" with the possibilities of supporting a name ("`foaf:name`"), nickname ("`foaf:nick`"), email ("`foaf:mbox`" and "`foaf:mbox_sha1sum`"), as well as a homepage ("`foaf:homepage`"). The agent node is connected to the annotation node via the relationships "`dcterms:creator`" and "`as:generator`".

INTENDED AUDIENCE Next to knowing who created a Web Annotation, another piece of useful information is the intended addressee that it is created for. This in general are direct groupings of users through features like employment groups (a very well fitting example from the WADM specification are teachers and students) or groupings that are defined by supporting various characteristical attributes like an age range or gender. The allocation of these information does not induce that the Web Annotation cannot be used at certain location, but systems can rather filter the annotations before they get in contact with the "wrong" audience group. To represent the intended audience in RDF, the Audience classes and properties of *schema.org*[7] should be applied. Its instance node is connected to the annotation node via the relationship "`schema:audience`".

MOTIVATION AND PURPOSE In the course of applying provenance information for Web Annotations it is not only important when and who created the annotation, but also with what motivation or out of what purpose she or he did it. This is done by assigning a motivation node with the relationship "`oa:motivatedBy`" to the annotation node. There are several purposes supported

7 http://schema.org/Audience (last visited 03/12/2018)

by the WADM specification, like "oa:bookmarking", "oa:commenting", or "oa:describing", which are so-called **named individuals**. This means, that rather than having a property set to the same specific value for many RDF nodes, they have a special unique IRI and only exist once in a given RDF graph to which the nodes can point to. This is a strong graph feature, as for querying purposes the named individuals allow the quick search for a given node, then all associated nodes can be found by following the named individual's edges, rather than querying for a text String of all the properties mentioned before. Next to this, the named individuals can be typed and therefore use all those advantages that have been mentioned in Section 2.2. All of the motivations in the WADM are of the "oa:Motivation" class.

OTHERS Last but not least there is more additional provenance information that can be added to a Web Annotation. However, these are not as important for the following work, so they will only be mentioned here and not discussed in detail. The **accessibility of content** allows Web Annotations to be marked for groups of users that have diverse ranges of ability, for example impaired hearing, in order to enable better access for those groups. The section about **rights information** explains how licenses and rights statements can be applied to an annotation in order to clarify the legal usage of annotations that eventually contain content that needs the clarification of free use. The properties associated with **other identities** allow the tracking of the origin for the parts of a Web Annotation.

3.2.5 *Complete Exemplary Web Annotation*

In order to sum up the Web Annotation Data Model and its capabilities as well as to show a thorough example of a complete Web Annotation, Figure 17 shows such an example, incorporating all the features and parts that have been covered in this section.

Additionally to the features described around and applied in Figure 12 and Figure 16, namely the content of the annotation - the statement that Barack Obama is shown on the picture, in the form of a textual body component - as well as the selection of the source image in combination with a fragment, the example also includes provenance information. These contain that the Web Annotation has been created on the "20th of June, 2017" at "12 o'clock", by a person (its resource is illustrated with the RDF node "bob") with the nickname "Bob". The reason for creating the annotation is set to the process of identifying (illustrated in Figure 17 with the named individual "http://www.w3.org/ns/oa#identifying"), which is attributed

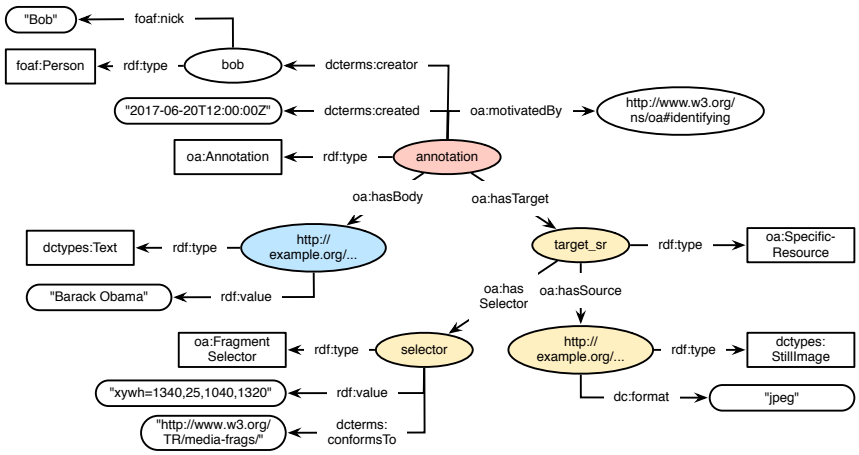


Figure 17: Complete Web Annotation with the Use Case of a Face Detection Process: Barack Obama Has Been Detected on the Given Input Image.

in the WADM to the identification of something that is depicted or described by the supported target.

3.3 THE MICO METADATA MODEL - CONNECTING ANNOTATIONS

In section Section 1.2 and Section A.1 in the Appendix, the MICO project has been described with its vision and its goals. It has also been discussed up to now, that one milestone of the project was to implement a unified data model for all its produced results. To achieve this, the **MICO Metadata Model (MMM)** [149] [23] [20] was implemented in order to satisfy this requirement. It allows to deal with a row of different extractors, all producing a manifold of different result and output formats, both in terms of producing and storing those results. Additionally, comprehensive querying capabilities are supported by the structure of metadata modelling implemented in the MMM. Provenance features are implemented to fully support the workflow-based approach of MICO processes.

The MMM was developed as an extension to the WADM, as the WADM already provides a broad range of useful tools in order to implement information about the multimedia items. The Web Annotations are generally used in order to further describe resources or associations between them, in many cases being comments or tags about web pages or textual documents [143]. The combination of these annotations forms a metadata background about the given resources. The WADM defines a structured model and ways for users to use the annotations.

Because of this, the WADM, and more precisely its Web Annotations with the possibilities of modelling the information content in

combination with selections and some provenance information, was implemented as the basis of the MMM. With the requirements listed in Chapter 3 in mind, the MMM lifts the context of the Web Annotations to a context of multimedia information allocation and workflow-based processes. A single Web Annotation, attached to the respective multimedia item, then represents the result of a single extraction step. Multiple extractors will add their information bit by bit to the item, eventually forming a rich metadata background for the item. The interplay of the media items and their enriched multimedia background promise to broaden the semantic applicability of the items. Hidden yet undiscovered semantics might lead the multimedia item to be used in a so far unseen use case.

On top of the Web Annotations and their structure of having a component for body, target, and the annotation itself, the MMM implements two additional core features in order to apply the idea of the Web Annotation to a multimedia use case. For reasons of convenience, the RDF classes, properties, and relationships have been divided into different *modules* they apply to. The modules can be seen in Figure 18: three adapted modules originate from the WADM, namely the **Content**, **Selection**, and the **Context** modules (standing for the body, target, and annotation components of the WADM respectively), with the newly added MMM modules **Composition** and **Provenance**. The colours applied in Figure 18 are representative for the modules, and the following figures and graphics will stick to this choice of colours as well, in order to depict the affiliation of the respective RDF classes and instance nodes to a module.

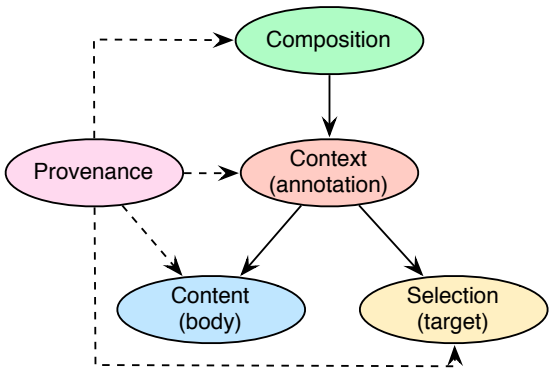


Figure 18: Module Structure of the MICO Metadata Model, Extending the Web Annotation Structure with Body, Target, and Annotation Component.

The modules adapted from the WADM fulfil the exact same role as they do in the WADM itself. The Composition module incorporates the elements that compose the multimedia items. The Provenance module extends some of the provenance information already present in the WADM, and extends overall provenance with an important

feature in a multimedia analysis workflow: the information about the extractors and their interplay.

In order to give a description of the MMM, the following section will build up the model features step by step, explaining them with examples wherever possible. In the explanations the focus will lie on the additions to the model, rather than the adaptions made from the WADM. Adapted classes, relationship, or properties are implemented as inherited RDF elements (for example "`mmm:hasBody`" as a sub-relationship of "`oa:hasBody`") and will be used and applied, but not discussed in further detail, as its semantics are not changed in regard to their role in the WADM, which was explained in Section 3.2. The model description explains module by module, but will at some occasions already refer to things contained in later modules, if a higher degree of comprehension can be achieved that way.

3.3.1 *Composition Module*

With the use case of a multimedia analysis workflow that focuses on the extraction of various features out of a given multimedia file to form its semantic background, the file itself needs more attention in the process of metadata modelling. In the MMM, this is supported by the composition module.

From now on, whenever an analysed multimedia object is referred to, the term **Item** will be utilised. In the process, every extraction result that is added to that respective Item is referred to as a **Part** or **Part Annotation**, which symbolises the participation of the analysis result in workflow chains and the overall metadata background that is created for the respective Item.

This is also applied in the metadata model. A multimedia file is implemented by the class "`mmm:Item`" with respective instance nodes. These Items form the basis for multiple possible analysis workflows and therefore create a tree-**like** structure of metadata (given that there will be backward and cross oriented edges), as an "`mmm:Item`" will have various "`mmm:Part`" instances attached via the "`mmm:hasPart`" relationship. "`mmm:Part`" is a sub-class of "`oa:Annotation`" and hence implements the structure of a Web Annotation, which will be covered in Section 3.3.2. A super-class "`mmm:Resource`" has been implemented that disjointly subsumes the "`mmm:Item`" and "`mmm:Part`" classes. This is done for ontology model reasons that will be seen later, as well as technical reasons shown in Chapter 4. Figure 19 shows an example for an Item with three different part children. Further content has been indicated by placeholders for body and target component, some type relationship have been left out for the Parts for reasons of clarity.

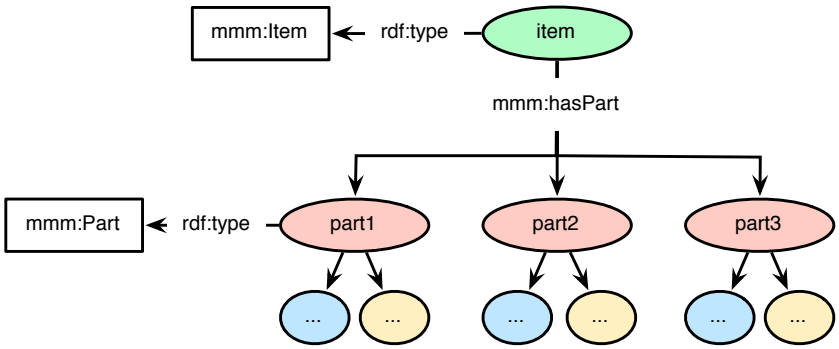


Figure 19: Exemplary "mmm:Item" Node with Three "mmm:Part" Children.

3.3.2 *Context Module*

In the WADM (see Section 3.2), the RDF class "oa:Annotation" is the central class, and its associated instance nodes connect the essential parts of a Web Annotation: the body, containing the actual information of the annotation, and the target component that may contain a selection in order to specify the multimedia file (or its subpart) that the annotation is based on. The MMM does adapt the structure and the semantic of the Web Annotation by introducing the "mmm:Part" class, which is a sub-class to "oa:Annotation". Next to some multiplicity changes for some relationships and provenance features (see Section 3.3.5), the semantic of a part is also extended in contrast to a Web Annotation. In general, the part still is kind of a description, classification, or further identification of a media item, but in a MICO workflow, a part also symbolises an intermediary or final result in the extractor workflow chains. Parts will link to one another in order to express sequential processes and allow for the traceability of those, and also link to the extractor instance that was responsible for its creation (see Section 3.3.5). Figure 20 shows an exemplary Part Annotation.

There are also two adapted relationships shown in Figure 20: "mmm:hasBody" and "mmm:hasTarget", which are sub-relationships to the relationships "oa:hasBody" and "oa:hasTarget" respectively. Both fulfil the same semantics as in the WADM, but the multiplicity of the "mmm:hasBody" relationship had to be adjusted for the multimedia analysis use case, as the single body represents the information of a single analysis step. Because of this, a part annotation always needs to have exactly one body node associated. As in the WADM, there always must be at least one target associated in order to refer to the multimedia item or multimedia subpart that the analysis was based on, but there may be more. Multiple targets indicate, that the result or statement of the Part Annotation is to be attributed to various multimedia files or their subparts.

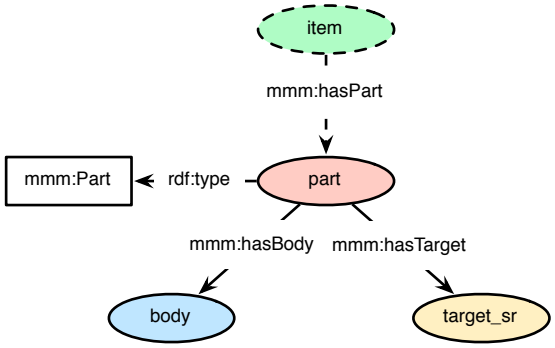


Figure 20: Exemplary "mmm:Part" Node with Attached Body and Target nodes.

3.3.3 *The Content Module / The Body of the Part Annotation*

As in the WADM for a Web Annotation, the body component of a Part Annotation in the MMM composes the central information holder of the whole construct, given that the results of the extractors will be implemented in the body component. This information is important when it comes to the utilisation of the metadata background that is created in a MICO analysis workflow.

For a starting point, a super-class "mmm:Body" has been implemented, from which all further body classes will be inherited from. The extraction results comprise a higher "semantic" level than basic datatypes applied to body components that have been seen so far. Because of this, every extractor will implement its own body class, which will be important in the workflow processes, as there will be extractions that require another preceding extraction to have taken place before, as well as querying capabilities. This allows for more fine-grained requesting of already analysed items.

During the course of the MICO project, a lot of different workflows composed of various extractors have emerged. To meet this important part of the evolution of the MMM, an own specification [20] (as a subpart of the MMM) with a respective namespace "http://-www.mico-project.eu/ns/mmmterms/2.0/schema#" and abbreviation "mmmterms" has been implemented.

Figure 21 and Figure 22 show two prominent representatives of the extractors that have been utilised over the course of the MICO project. Figure 21 shows the use case of an animal detection extractor, which is fairly similar to the face detection, but rather than detecting humanly face shapes, it tries to detect animals on a given picture. Because of this, the constructed part annotation also is similar to the one of a given face detection result: the detected animal is enclosed in the body component, while the fragment of the picture that shows the actual animal is implemented in the target component. In con-

trast to the examples that have been seen in the Section 3.2, in all MICO related Part Annotations, the extractors can issue their confidence about given results via the property "`mmm:hasConfidence`" and an associated percentage value between 0.0 and 1.0. The type of the extractor is associated with the RDF type of the body node, in this case "`mmmterms:AnimalDetectionBody`". Altogether, the statement of the whole Part Annotation is that an "`Elephant`" was found with a confidence of "`90%`" on the fragment "`xywh=300,150,50,70`" of the respective input picture.

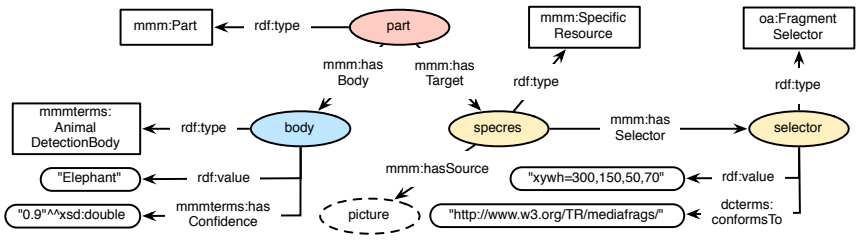


Figure 21: Exemplary Part Annotation, Illustrating the Result of an Animal Detection Analysis.

Temporal Video Segmentation TVS [129] is the process of dividing a video into temporal segments, which support the partition of the video into logical units called **shots**. In general, those shots lead over each other in **transitions**, which indicate the **shot boundary**, meaning the ending of one shot before the start of another. Videos consist of a sequence of consecutive **frames** (pictures with a very short playing time, eventually forming the video), and single frames will be used to indicate a shot boundary, called the **shot boundary frame**.

All of these information (except the transition, which is a side product of all the aforementioned details) are incorporated in the MMM. Figure 22 shows a Part Annotation that indicates a complete shot in a given video resource by supporting a "`mmmterms:TVSShotBody`" (with a confidence value like all MICO extractors) in combination with a temporal fragment in the target component. In the example of Figure 22, the shot would start at second "`1.000`" and end at second "`3.155`".

3.3.4 *The Selection Module / The Target of the Part Annotations*

The addressing of the multimedia data with its subparts that serve as the input for described analysis processes forms a fundamental part for multimedia metadata modelling. In Section 3.2.1, the supported possibilities of the WADM are documented. As those are already very diverse and cover all requirements for multimedia addressing that have been encountered in the MICO project, the MMM adopts the

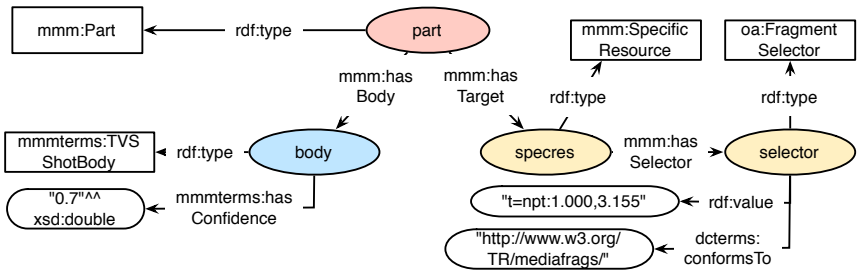


Figure 22: Exemplary Part Annotation, Illustrating the Result of a Temporal Video Segmentation Analysis.

WADM's implementation and adapts some of the multiplicities for the multimedia analysis use case.

Both the "`oa:SpecificResource`" class and the relationships "`oa:hasSource`" and "`oa:hasSelector`" have respective counterparts in the MMM. For every Part Annotation there must always be a "`mmm:SpecificResource`" associated via the relationship "`mmm:hasTarget`", but there may be more. On the contrary, a classic Web Annotation is not needed to have a Specific Resource, as the target can also be addressed directly. An already known "`oa:Selector`" can be applied to the Specific Resource instance via the relationship "`mmm:hasSelector`". There can be none or a single selector applied to a Specific Resource.

A more special role is attributed to the source of a Part Annotation, which is referenced by the specific resource node via the relationship "`oa:hasSource`" in the WADM, and "`mmm:hasSource`" respectively in the MMM. Classically, this relationship links the Specific Resource to the resource that it is a more specific representation of. For a Part Annotation, there must always be a single "`mmm:hasSource`" relationship present which links to a "`mmm:Resource`" that stands for the multimedia asset that the respective analysis process of the Part Annotation is done on. This is explained in more detail in Section 3.3.5. Examples have been shown already in Figure 21 and Figure 22.

3.3.5 *Provenance Module*

A very important feature in a scenario like MICO with multiple autonomous extractors analysing multimedia items to form metadata backgrounds and communicate with a central platform while doing so, provenance is an important cornerstone. It allows the preservation of a state of traceability through the workflows, indicate information about given multimedia files as well as a richer description about extractors and their settings.

The first thing that is needed for an extraction workflow is the multimedia file itself with its details, for which it is important to record its physical location as well as its format. The location is accessed when

an extractor wants to analyse an file, the format is used when it is determined, if a given extractor can analyse the underlying multimedia file. In the MMM, this information is associated with an instance of the "`mmm:Asset`" RDF class. These nodes are attached to an Item or a Part via the relationship "`mmm:hasAsset`". Enclosed with an Item, the respective Asset represents an original multimedia file that has been injected into the platform to be processed, while an Asset of a Part Annotation resembles an intermediary multimedia file that has been created as the result of an extraction step. This could for example be a frame and picture or an audio soundtrack that has been extracted from a video. The location is applied with the property "`mmm:hasLocation`", while the format is supported via the property "`dc:format`". Figure 23 shows an example of a video with the format "`video/mp4`", which will have an intermediary image with the format "`image/jpeg`" extracted. Note here that the supported formats should align with commonly known MIME types[8], but it is allowed to add own format types in order to associate "semantic types" as a result of an extraction workflow.

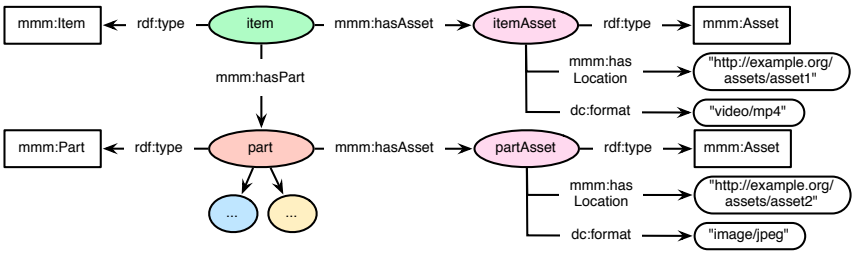


Figure 23: Exemplary Item and Associated Part Annotation with Respective Assets.

Next to the basic details about the multimedia items, timestamps are a central provenance information that need to be kept track of. For those, the relationships present in the WADM are to some extent adapted at Item and Part Annotation instances. As in the WADM, the timestamps supported should be a "`xsd:dateTime`" with the UTC timezone expressed as "Z" [112]. The adapted semantics in the MMM[9] are as follows:

- "`oa:serializedAt`": The point of time at which a given Item has been materialised at the corresponding platform instance, or when a Part Annotation has been created by an extractor.

8 https://www.sitepoint.com/mime-types-complete-list/ (last visited 03/12/2018)

9 With one of their issued versions, the WADM specification has changed the IRIs for the timestamps to "`as:generator`", "`dcterms:created`", and "`dcterms:issued`". These changes have not been incorporated into the MMM, however these properties and relationships still incorporate the same semantics.

- "oa:annotatedAt": The point of time when a Part Annotation is connected with its corresponding Item instance.
- "oa:serializedBy": The link between a given Part Annotation and its responsible extractor instance (which will be described below). This relationship can also be associated with Item instances in case there exist different ways of materialising them at the given platform instance.

The most central point about the provenance information that is represented by the MMM is the traceability of the multimedia analysis workflow processes, which in detail covers the topic of the "input" of the intermediary and final result steps. Understanding the input of every step allows to retrace the whole workflow that has been executed for metadata allocation in the data model. However it is important to cover the semantic background for both of these relations.

The term "input" for the analysis steps has to be seen twofold in our context. Firstly, there is the actual metadata input that an extractor needs in order to do his own processing. In the MMM and this work, this will be referenced to as the **input** of a Part Annotation. The input relationship is implemented with the RDF relationship "mmm:hasInput" and is directed from a Part node to a node of the type "mmm:Resource", so an Item or another Part. Secondly, there is the multimedia Asset that a given analysis process is done on. This is called the **source** of the Part Annotation, and this relationship has been explained in Section 3.3.4. Figure 24 shows an extensive example for this provenance feature, whose key points are the following.

- The workflow behind the example in Figure 24 contains the initial ingestion of a multimedia Asset with the RDF pendant "asset1" (with the IRI "http://example.org/assets/asset1") and three consecutive extractor steps, which will start their extraction once the multimedia Asset is ingested at the given platform instance and the preceding analysis step has ended eventually. The first and third extractor (responsible for creating the part annotations "part1" and "part3" respectively) are "metadata extractors", meaning that they produce metadata information only, rather than an intermediary or final result that is combined with an extracted multimedia file. By contrast, extractor two is a process that does output a file which is indicated by the relationship "mmm:hasAsset", referring to its "mmm:Asset" named "asset2". Those are called "asset extractors". The location of the created Asset is "http://example.org/assets/asset2". The input and source of every Part Annotation is implemented by the two relationships "mmm:hasInput" and "mmm:hasSource".
- "part1" as the first Part Annotation of the workflow chain refers to the Item node "item" as its source, so its "mmm:hasSource"

edge (found at the target node "target1") links to the Item "item", which in terms stands for the first multimedia Asset "asset1". The extractor is some kind process that utilises a multimedia file directly as input, indicated by the relationship "mmm:hasInput", which is also directed towards "item".

- The second extractor has another function, as it takes the metadata extracted by the first extractor and produces a file. The output is referred to via the "mmm:hasAsset" relationship. As the actual input for the second Part Annotation is resembled by the node "part1", the "mmm:hasInput" edge is directed from "part2" to "part1". Its source however is directed to the Item "item", because the extraction process illustrated by "part2" is based on the initially ingested multimedia file in combination with the metadata composed by "part1".

- Extractor number three then can take the output of extractor two, as it is some kind of process that needs the multimedia Asset "asset2" as input. Because of this, the source of the third Part Annotation "part3" links to "part2" (and not the initial Item). As that Part Annotation with its Asset also contains the metadata input needed (if needed at all) for the creation of "part3", the "mmm:hasInput" edge is also directed towards the Part Annotation "part2".

Last but not least the provenance module of the MMM features an extensive description for the extractors that take place in the analysis process. Figure 60 in the Appendix shows an comprehensive example for a full extractor model. As it is quite extensive, but on the same hand rather self-explanatory, it will only be referred to in short.

The most important part of the extractor provenance model are the classes "mmm:Extractor" and its associated "mmm:Mode". In the MICO use case, extractors could be started in different modes, allowing the same general extractor to have different settings. In the example of Figure 60, the Audio Demux could be set to different sampling frequencies, which leads to varying output. As described above, the Part Annotation is connected to the instance of the extractor via the relationship "oa:serializedBy", while the respective mode of the extractor is associated via the relationship "mmm:serializedWith" (which is an addition to the MMM that does not have a counterpart in the WADM). This is necessary because simply linking the Part Annotation to the extractor would not suffice, as the information about the applied mode is also important.

The extractor itself is described with a **name**, **id**, and **version**, while modes have a **description** and also an **id**. Next to this, modes are associated with more detailed information (by the implementation of RDF instances) for **parameters** and **input**/**output** instructions. Parameters are instantiated with a String **name** and a **value**. Input and output

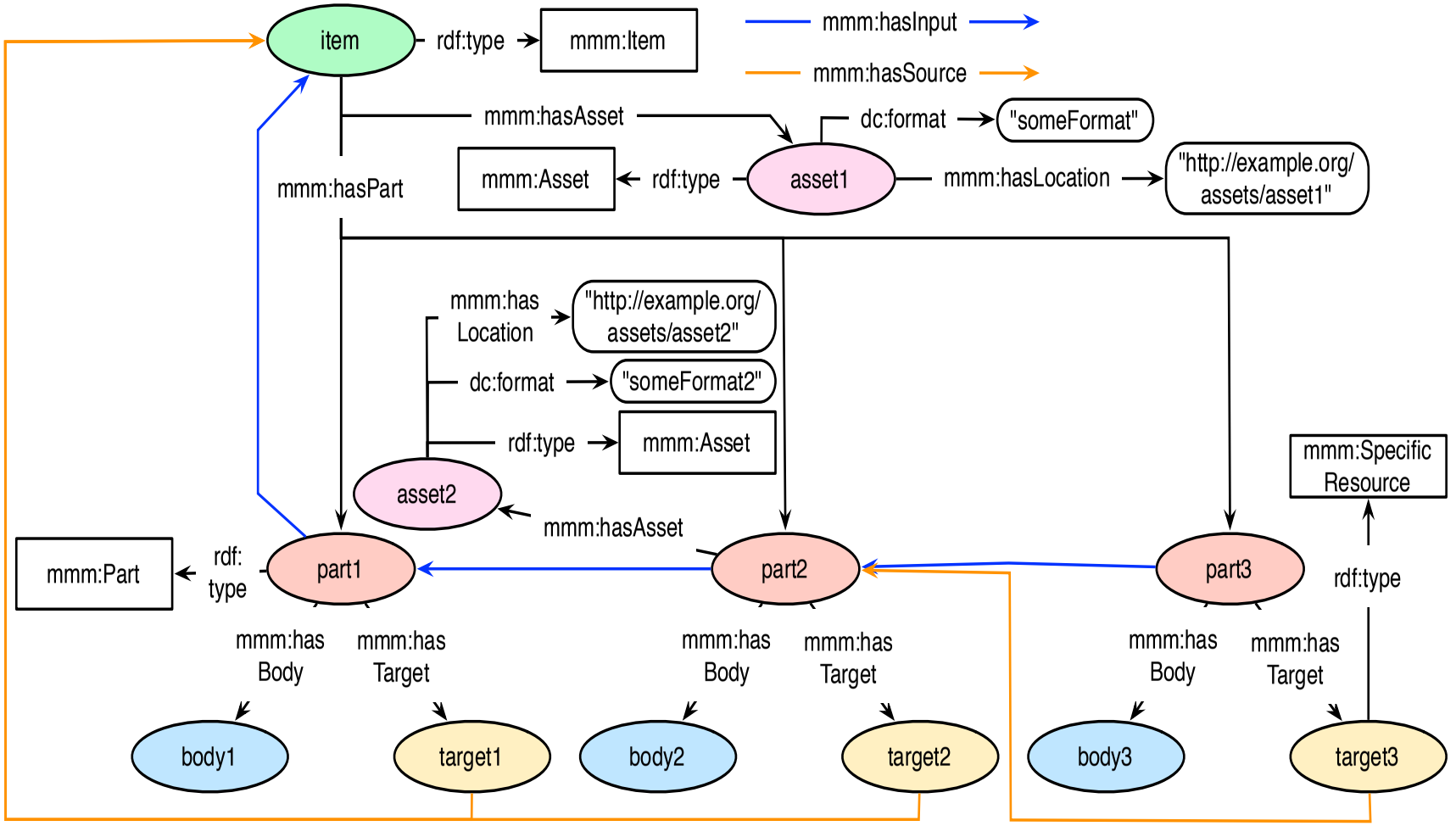


Figure 24: Exemplary Item Structure that Shows the Input Provenance (Blue Edges for "`mmm:hasInput`", Orange for "`mmm:hasSource`") of the MMM.

of the modes share the information of having a **semantic type**, **mime type**, and **syntactic type**. The former mainly describes the given input or output in a semantic high level fashion and is therefore mainly applied for humans. The mime type links to a corresponding named individual implementing the commonly known mime types for multimedia files. Lastly, the syntactic type links to a more general, MICO use case specific data type, which is used for workflow provenance.

3.3.6 *Multimedia Ontology Requirements Check*

The related work shown in Section 3.1 described a list of requirements that an ontology for describing metadata of multimedia items should comply with. To prove that the MMM suits for the application of a multimedia metadata ontology, this section evaluates the MMM against these defined requirements. Some of the following points will also be supported practically by the Anno4j library which will be seen in Chapter 4.

(√) MPEG-7 COMPLIANCE Neither the WADM or MMM implement the features of the MPEG-7 standard by default, however the model can be dynamically adapted at its body component to support every MPEG-7 feature. Some of the ontologies and approaches in the related work in Section 3.1 have already implemented RDF pendants, which could also be applied in the MMM.

√ SEMANTIC INTEROPERABILITY One of the core requirements of the WADM itself is to implement the Web Annotations in a fashion to make them shareable across different applications and peers. There is also a specification next to the WADM, the Web Annotation Protocol, whose main purpose is to implement a client server architecture to exchange Web Annotations. So by default, the MMM also supports semantic interoperability.

√ SYNTACTIC INTEROPERABILITY As both the WADM and MMM are implemented in RDF, aligned with Semantic Web technologies, the model supports syntactic interoperability between systems.

√ SEPARATION OF CONCERNS The structure of the Part/Web Annotations with their partitioning between body (domain knowledge) and target (structural knowledge) components that are connected by an Annotation or Part node respectively, supports full separation of concerns. The connecting Annotation/Part node can further contain administrative knowledge.

(√) MODULARITY In the WADM and MMM, modularity by definition is not supported as for example in the MPEG-7 standardisation. But rather than supporting various subparts, modularity

is supported when it comes to the body component that can be implemented and adapted to the specific use case. In a way, the user then only needs her or his subset of body implementations in combination with the always needed core components of a Part/Web Annotation.

√ EXTENSIBILITY Both the WADM and MMM are designed in an extensible fashion. The main features that should be extended are the body component as well as the selectors, however (as it was done for the MMM as well) every RDF class, relationship, and property can be extended and adapted to various use cases.

Next to the requirements that have been defined by the authors of the issued related work in Section 3.1, there have also been requirements that have been compiled for the data model during the course of the MICO project. These factors originate from multimedia metadata in general, as well as workflow-based features that have to be incorporated as a result of the processes that have been implemented in the project. These have been enlisted in the introduction of Chapter 3. The following list explains the compliance of the MMM in regards of those requirements:

√ PLENTY OF DIFFERENT DATATYPES The MMM does not impose any restrictions on the analysed datatypes, querying capabilities, or comparison operators. All of these features can be freely adapted to a specific use case. Pre-knowledge of produced metadata can be reduced, as the Anno4j library supports ways of applying and producing the metadata in a more convenient way (see Chapter 4).

√ FRAGMENTATION OF MEDIA ITEMS The fragmentation of multimedia files and their content is fully satisfied by the implementation of Selectors on the side of the WADM and the associated standardisations that can be applied. Because of this, the metadata and therefore the statement or result of a given extractor can be addressed in a very fine-grained fashion to the exact fragment or subpart of a given multimedia file.

√ STACKING OF MEDIA ITEMS A "stacked media item" consisting of several basic multimedia data, can be implemented in the MMM in different ways. The breakdown of the enclosing stacked item can be done by splitting it through the application of Part Annotations for every subpart. Doing this, the respective subparts can be addressed directly by following analyses. Next to this, it is possible to interpret the stacked item as it is and address it by supporting an own datatype to it. This does not allow the same level of granularity, but can have advantages of its own. The addressing of incorporated media files must be done via proper choosing of Selectors.

√ QUALITY OF MULTIMEDIA METADATA In the universe of MICO and the MMM, the metadata background that is generated for multimedia files is highly adjustable to the eventual use case. Multiple workflows can add their metadata and resulting features to one and the same multimedia item type, creating rich and diverse metadata for the analysed multimedia files. In combination with the comprehensive querying capabilities enabled through the MMM, this fuels the applicability of those multimedia files and therefore increases their quality as well.

In retrospective, this chapter has given insights into the field of multimedia metadata modelling, which is an important topic for decades now. It is still evolving and there are many applications or use cases that make use of it today. Section 3.1 has highlighted some of the influential cornerstones of the field's history, which has also lead to a row of requirements that need to be met by a modern multimedia ontology. Considering these requirements, the Web Annotation Working Group's prevailing Web Annotation Data Model stood out to be a fitting candidate for a base ontology due to its satisfaction towards mentioned ontology requirements as well as its origin in the Semantic Web technologies. The WADM is described in Section 3.2. Furthermore, as this work envisions a workflow-based application of multimedia metadata, an extension to the WADM, namely the MICO Metadata Model MMM, is introduced in Section 3.3, which extends the functionality of the WADM to be also useful to such applications. Last but not least, the MMM was positively evaluated against the requirements for multimedia metadata ontologies defined in Section 3.1.

Next to the knowledge about multimedia metadata ontologies, the related work has also shown that an important factor next to the ontology itself is to make the metadata technically accessible and usable in some way. In the field of computer science, this is often done over APIs or whole full-stack applications. Therefore, the following part of this work will show how the MMM can be embedded in a workflow or eventually a cycle of work steps, that eventually comprises the processes of producing, applying, and consuming multimedia metadata in order to create a rich metadata background for multimedia items. This workflow will first be explained and then embedded into a whole software platform in Chapter 5. Next to this, an implemented library called Anno4j will be shown in Chapter 4, which enhances the usability of Semantic Web technologies altogether and within that also applies the MMM in a convenient fashion for the user.

Part III

THE APPLICATION OF MULTIMEDIA METADATA IN A WORKFLOW-DRIVEN APPROACH AND IDIOMATIC SEMANTIC WEB TECHNOLOGIES

4

INCREASING THE USABILITY AND APPLICABILITY OF SEMANTIC WEB TECHNOLOGIES FOR MULTIMEDIA METADATA MODELING AND WORKFLOWS - ANNO4J

Up to this point, this work has covered and explained concepts and ideas of the Semantic Web world by having a workflow-centric process in mind, which is motivated by the vision of the MICO Project described in Section 1.2 and Section A.1 in the Appendix. Ontologies and metadata models have been the most important point of the last sections. They compose both a syntactical as well as a semantical structure for metadata, and thus induce how already existing metadata is to be interpreted on one side, but also how it has to be produced structurally on the other side. It has been learned up to know that especially at these factors and processes, barriers exist for the overall Semantic Web technologies [39], and especially non-Semantic Web experts. To counteract these, this chapter will introduce technical solutions for those barriers.

Therefore, after the basics of Semantic Web technologies have been shown in Chapter 2, the concept of metadata modelling has been highlighted in particular in a combination with multimedia in Chapter 3. Also, a multimedia metadata ontology, namely the MICO Metadata Model MMM, has been developed and described in Section 3.3 which servers as the modelling backbone of this thesis.

Nevertheless, it has also been brought up throughout the sections and chapters enlisted above, that an ontology on its own can serve a lot of purposes, but there is a much higher potential when it is combined with respective technologies and applications. This statement has to be seen twofold. Most of the implementations that apply the ontologies and models hinted especially in Section 3.1 are generally oriented in the direction of full-stack applications. This means that there is mainly a piece of software in combination with various interfaces that support the seemingly convenient utilisation of the underlying ontology, allowing the user to produce and consume the metadata without getting involved with the procedure of basic Semantic Web technologies, for example in the form of SPARQL queries. This obscuring is in most cases necessary and to some extent helpful, as the generic user might have only limited knowledge of the Semantic Web technologies necessary. By being forced to use unfamiliar technologies, in most cases the user would not make use the system as a consequence. This however somehow overshadows the features of the ontology and narrows its possibilities and applicability, not only

for the expert (who may have implemented the full-stack application), but also the new potential user itself. Additionally, this kind of application does not allow a user to further familiarise oneself with the Semantic Web technologies.

The second possibility to combine an ontology application with respective technology cannot be called an application of its own, but is rather targeted at smaller parts of functionality (and can therefore of course also be incorporated in a full-stack application). An example for this kind of technology could be a piece of software that does mainly focus on the querying and persistence of RDF data. Although these instances comprise smaller tasks, they are respectively better suitable for familiarising a non-Semantic Web expert with the technologies present on the one hand, and on the other hand they provide more space for users who are already familiar with Semantic Web technologies to create comprehensive and rich modules for Semantic applications.

Eventually, the building blocks of this work will represent modular implementations that mainly focus on the increase of applicability and usability of Semantic Web technologies. This does help non-Semantic Web experts to apply said technologies much more conveniently. However, these incremental implementations will also be introduced in a full-stack application serving the MICO visionary use case, in order to demonstrate, evaluate, and validate their applicability. The resulting **MICO Platform** will be explained and compared against related implementations in Chapter 5. As described in the MICO vision, the target of the platform incorporates the implementation of a central piece of software to which autonomous multimedia extractors can connect to and add their analysis results to ingested multimedia files. This at last forms the file's metadata background, which then can be used for further processing and exploitation of information. An obstacle existed at this point, which can be closely compared to problems of the Semantic Web described in Section 2.1: the barrier for the extractor developers to write and query their results in the form of RDF metadata, which existed next to the complex task of devising and implementing the extractors.

As a solution, the Object Relational Mapping ORM-like library called **Anno4j** to map Java objects to and from RDF data has been implemented. This chapter will continue to explain the basis and origin of ORMs in combination with related work to this topic in Section 4.1. Afterwards, a detailed description of Anno4j and its components will be given in Section 4.2, preceded with a summarisation of scenarios that can be solved more conveniently through the application of the library. These also comprise some of the problems present in the Semantic Web, indicating that Anno4j is able to resolve some of them or at least lower the barrier. Section 4.2 will also give a more detailed sectioning of the remaining part of this chapter. The entirety of this

chapter proposes a solution to research question 2, while its subsections may also refer to further research questions.

Although Anno4j represents a self-contained abstraction layer that is placed on top of the database layer, different approaches and solutions of varying technical perspectives will be applied. Therefore, various informative illustrations will be used that facilitate the description of the approaches. Each of those will be embedded in the respective context and will be explained accordingly.

4.1 RELATED WORK - OBJECT/RELATIONAL AND OBJECT/RDF MAPPINGS

The concept of **Object/Relational Mappings** is a well-known concept, and therefore to this day has accumulated a lot of development and research. Despite its naming, which is sometimes entitled as *object mapping* rather then ORM (which is generally used by the Hibernate[1] team), all designers of the concept agree upon its use and exertion: a piece of code or framework that can be utilised in an application to facilitate the interaction with data, mainly covering the fields of persistence, which is a fundamental part in the design as well as implementation of an application. The overall description of ORMs is aligned with the explanations and insights of Bauer and King [16] and O'Neil [123], as their descriptions are detailed and in depth, and Hibernate is one of the best known ORMs of today.

The employment of an ORM into the development an application can lead to facilitation that can be seen at various occasions. The main purpose is to help with **persistent data** (in contrast to *transient data*), information that is fully persisted to the database and which therefore can outlive for example the runtime of an application or the current workflow of a client. This process is also done automatically and transparently, as it is taken over by the ORM and interwoven into the application code, supporting robust and efficient management of the data. It also enables concurrent access to the data, thus multiple users can work on the same state of the data without interfering one another. Besides this, the ORM takes over or lightens several "low-level tasks" for the application developer. Among those are the persistence and querying of data, execution of queries for those associated tasks, dealing with query parameters, or searching through results. Although these tasks are "easier" implementation- and effort-wise than for example the code for the logic of the application, they should not be treated lightly, as writing all these interactions by hand can pose a heavy timely burden that has to be done for even simpler applications. As the model for the application gets larger or more complex, this circumstance gets aggravated even more. The utilisation of an ORM for these tasks does also require some more lines of

1 http://hibernate.org/orm/ (last visited 03/12/2018)

code to be incorporated, however as ORMs generally are designed in a non-intrusive manner, this effort is kept to a minimum and can leave the developer to concentrate on existing requirements of the actual application.

Altogether, ORMs promise to have the following advantages when applied to an application:

PRODUCTIVITY The code that has to be written for the interaction between client application and the underlying database is reduced significantly. Rather than writing queries for every single mapping requirement that has to be met (this sometimes also includes simple interactions like getters and setters), the basic functionality of issuing a query via the programming language objects is the same for every class and has only to be written once. Some of the ORMs are also able to generate parts of the interaction code right away.

DEVELOPMENT EFFORT In this criteria, a hand-written persistence layer implementation is compared with the application of an existing ORM for the same purpose. Performance-wise, the hand-written code can surely be as good as the ORM code, as oftentimes the ORM implementation has to conduct some time-consuming functionality. Nevertheless, one has also to take the development time into account, which rises quickly for hand-written code with the size of the to-be-implemented application, while most ORMs are incorporated easily. Next to this, the optimisation space is very limited for hand-written implementations or again takes considerable timely efforts, while existing ORMs have had specialists that invested time and expertise into those optimisations. In summary, using existing ORM implementations for persistence purposes takes less development time, even though the application developer has to familiarise himself with the ORM, and it is better optimised for database usage.

NON-INTRUSIVENESS The objects that take part in ORM mappings are rather simple and can mostly directly (or in combination with a slight adaption) be used in the client application as data objects or other instances as well. The ORM code that has to be implemented in respective applications is also focused on being usable in convenient and non-intrusive fashion and can in many cases be picked up quickly.

REUSABILITY In contrast to self-written queries, especially for the interaction with the database as well as the implemented mapping objects, code associated with an ORM is highly reusable. This allows code pieces to be used alongside other software and

applications nearly right out of the box. Code does not necessarily have to be implemented redundantly and from scratch.

ERROR-PRONENESS AND MAINTAINABILITY Code associated with ORMs creates room for testing capabilities. Starting with the basic ORM functionality for the communication that can be tested, it is also possible to write tests designed for specific class objects next to more thorough querying use cases, rather than cumbersome testing of difficult to maintain SQL queries. This immensely decreases the error rate of the whole system while increasing the maintainability, as errors, corresponding adaptions or updates can be implemented into the ORM system much easier and quicker. This does also influence the reusability, as the tests are written for more general parts of the system rather than specific ones and must not be recreated for every feature or mapping item that is added to the system.

APPLICATION DESIGN Software patterns can be included into an application design through the utilisation of an ORM. This makes the code much cleaner and more efficient, further developing gets facilitated by clear processes and implementations.

(VENDOR INDEPENDENCE) Most ORM implementations are independent from a specific underlying database, but rather focus their interaction on a querying protocol or language like SQL. This enables the ORMs to be used with any database that understands the respective language. This independence enables easier cross-platform implementations. Additionally, deployment processes are also facilitated, as test implementations of an application can run on lightweight databases while the productive system can work on a large scale database. With the aforementioned independence of database however, the migration process can be done seamlessly.

Despite their many advantages, ORM implementations bear several known issues that arise from the applied mapping which creates a mismatch in terms of the data representations, as two different worlds or paradigms collide. In [16], Bauer and King name and describe the following problems:

GRANULARITY Java objects have a great range of possible granularity, from simple basic datatype classes to coarse classes to more specialised and detailed information holder classes. This granularity however cannot be reflected in a relational database, as the possibilities there are more restricted. This confined scope for data modelling is then oftentimes reflected on the domain model of the application, leading to shortcomings like missing functionality that has to be implemented in another, oftentimes more costly, way.

SUBTYPES Two of the strongest features of object-oriented programming languages are *inheritance* and *polymorphism*, which both can be applied to even more functionalities like polymorphic querying. These concepts are neither reflected in SQL nor in a relational database.

IDENTITY The problem of identity is encountered once you deal with the potential *equality* of data items. This is necessary in different functionality of the database like querying. While Java supports two interpretations of identity (object identity mostly checked by memory location, and secondly the equality test via implemented `".equals()"` methods), neither of both does match the identity check that is done in the relational world: the test on primary keys.

ASSOCIATIONS **Associations** represent the linking between different information objects. The mismatch occurs with this concept as the associations in the Java world are implemented via *object references* which can be included directly in a class. On top of that, these references can have any multiplicity modelled: "one-to-one", "one-to-many", and "many-to-many" (which is a little bit more difficult but manageable). In contrast, the relational database implements these associations via *projection* and *table joins*. For a many-to-many association, separate *link tables* have to be implemented. While the object references in the Java world are simple to implement, use, and evaluate, in the relational world several joins and costly operations have to be executed in order to apply associations between database entries.

OBJECT GRAPH NAVIGATION Closely related to the problem of association, the **navigation** through the object graph that is spanned by objects and their interconnected associations poses another problem. In Java this can be done by calling the methods that are supplied for the associations (in most cases, this is a simple getter method). For relational databases, there already is no such thing as graph navigation, as it is simply not possible to implement. Costly join operations between multiple tables hamper the attempt to do so. Furthermore, when writing an SQL query, a user has to know what he wants to query in advance when the query is written, which does not truly represent the concept of navigation as it is in the Java world.

As with the overall concept of ORMs, the awareness for these problems has existed for a long period of time, and therefore current ORMs can (to a certain extent) deal with them. Handling these demands a more complex implementation which oftentimes involves more enduring queries and run-times for processes, however supporting the functionality of the object-oriented languages mentioned

above is worth the time investment. Hibernate itself contains some of the best solutions for these problems, and the authors stress again at that point that these problems are not to be taken lightly. From their experience, they issue that the main purpose of about 30% of the code written for a Java application is about handling the tasks and problems enlisted above, next to the cumbersome trouble of dealing with SQL and the database connections via JDBC [136] for example in the relational world [16]. Both of these issues necessitate a lot of timely effort and prohibit the developer to focus on the actual application itself.

From a technical point of view, ORMs in general support a mapping between objects of an object-oriented programming language (Java in the case of Hibernate) and relational data, data entries that are recorded in the data tables of a relational database. This mapping is established by supporting metadata to the domain model that is applied by the respective application. Implementation-wise, the ORM code is integrated into the persistence layer of the overall application, interacting between the business layer and the database itself. The ORM is able to generate automated queries derived from the application and the calls from higher layers.

Altogether, the ORM deals with the topics of storage, organisation, and retrieval of structured data, concurrency and data integrity, as well as data sharing. To do so, every ORM should support at least the following components [16]:

CRUD API For every mapping object defined in an ORM mapping, the basic CRUD functionality (standing for **C**reate, **R**etrieve, **U**pdate, and **D**elete) must be supported. This is to be incorporated in an API that should be usable as natively as possible.

QUERY LANGUAGE OR API Next to the basic querying functionality of the CRUD API, an ORM should also support richer querying capabilities via a querying language or an extended API. This enables more fine-grained querying, so result sets can be thinned out at querying time and therefore only objects that are relevant to the respective query are returned by the database. This querying functionality needs to be fine-tuned to be able to address the model objects as well as their defined properties in a convenient fashion. It also should align with the automated fashion of an ORM and be able to generate queries automatically to not pose any further effort on the ORM user.

MAPPING METADATA The mapping between the objects and their representation in the relational database is somehow supported by metadata. An ORM needs to define this metadata model in combination with a set of rules how to apply it. Once again, non-intrusiveness is rated highly, as the applier of the ORM must not

invest a to big amount of time to incorporate the ORM into his application.

OPTIMISATION FUNCTIONALITY Up to now, it has been seen that several issues arise in the implementation of data persistence. There are however many more functionalities that are not essential for a working ORM, however they can contribute to a better interaction with the overall architecture. Bauer and King [16] mention *transactional behaviour* and *lazy association fetching* as examples. These can mostly be incorporated into the functionality of the ORM.

The above description has given insight into the world of ORMs that work with relational databases. As already mentioned, Hibernate is one of the best-known and applied representatives, others are for example Doctrine[2] [61], Apache Cayenne[3], Apache OpenJPA[4] (formerly known as Kodo), or ActiveJPA[5].

A similar clash of technologies exists in the Semantic Web world, which has also taken up the concept of ORMs to enable more convenient access to their databases via commonly known (object-oriented) programming languages. The database is represented by a triplestore, which is accessed by SPARQL queries, rather than SQL queries. Just as with the ORMs, the underlying concept provides the same advantages as described above. To prevent naming confusions and prohibit concept clashes, the naming **ORdfM** for **Object RDF Mapping** is introduced, which implements the same concept as it is known from ORMs, with the difference of being applied to the Semantic Web databases rather than relational pendants.

The aforementioned clash is seemingly less severe than in the relational world for a comparison between RDF data and an object-oriented programming language like Java. Although the *granularity* problem exists as well in this mapping, *subtyping* can be modelled straight away, as an RDF instance node can have several RDF classes/-types assigned. If not done by the database immediately (as only the most specific type is generally added), a reasoner could deal with this (reasoning has been described in Section 2.2.6). The *identity* problem is also less severe, as an RDF database does not support primary keys, but RDF instance nodes are rather distinguished by having unique identifiers, which comes closer to the Java world than its relational pendant. In terms of *association* and *graph navigation*, the data representation in the Semantic Web poses no problems for the clash, as the object graph defined by the objects of the programming language can directly be mapped to the RDF graph. Accessing an association from

2 http://www.doctrine-project.org/ (last visited 03/12/2018)
3 https://cayenne.apache.org/ (last visited 03/12/2018)
4 http://openjpa.apache.org/ (last visited 03/12/2018)
5 https://github.com/ActiveJpa/activejpa (last visited 03/12/2018)

Java object to object is directly representable via an RDF edge for a relationship or property between RDF instance(s) or a literal.

There are several prevailing representatives of ORdfM implementations out there, which are seen as related work towards this thesis. **RDF2GO**[6] [148] (or also earlier known as RDFReactor [171]) defines an abstraction layer over a triple/quad store. Therefore, easy and quick implementation is envisioned, while the choice of data storage can be done later. Adapters are supported for Jena[7], Sesame (now known as RDF4J[8]), and OWLIM[9]. **Tinkerpop**[10] is an open-source graph-computing framework that lets their users model their domain as a property graph (a graph built from common nodes and edges, which *both* can be enhanced with key/value pairs). These graphs then can be analysed using their own graph traversal language Gremlin[11]. Tinkerpop systems can be consolidated easily, allowing for convenient collaboration, while every system can have its own appropriate graph technology underlying. The **Callimachus project**[12] [15] is also an open-source project that supports a framework to construct Linked Data applications. To achieve this, the user is able to generate human-readable web pages that are combined with RDFa (see Section 2.2.5) templates that are supplied by a RDF database, resulting in a web page that is populated with RDF data.

The following three implementations follow the same approach of using Java Reflections[13] in order to establish the mapping between a Java POJO and the resulting RDF data, enabling the process of incorporating the ORdfM in a non-intrusive and idiomatic fashion. **Empire**[14] is mainly centred on supporting a standard Java Persistence API (JPA) style interface for RDF databases in order to create the ORdfM layer. Empire is an open-source framework which was motivated out of the need to create RDF-backed web apps. **RDFBeans**[15] is built upon the Eclipse RDF4J API[16] (former Sesame) and is a mapping framework that mainly focuses on convenient CRUD functionalities of the mapping, as well as a dynamic proxy mechanism for transparent access of RDF data. By extending the RDF4J API, RDFBeans is applicable for many different third-party database solutions.

The most promising approach is given by the **Alibaba** (former Elmo codebase)[17] library. It is aimed towards the development of

6 http://semanticweb.org/wiki/RDF2Go (last visited 03/12/2018)
7 https://jena.apache.org/ (last visited 03/12/2018)
8 http://rdf4j.org/ (last visited 03/12/2018)
9 http://semanticweb.org/wiki/OWLIM.html (last visited 03/12/2018)
10 http://tinkerpop.apache.org/ (last visited 03/12/2018)
11 http://tinkerpop.apache.org/gremlin.html (last visited 03/12/2018)
12 http://callimachusproject.org/ (last visited 03/12/2018)
13 https://docs.oracle.com/javase/8/docs/technotes/guides/reflection/index.html (last visited 03/12/2018)
14 https://github.com/mhgrove/Empire (last visited 03/12/2018)
15 https://github.com/cyberborean/rdfbeans (last visited 03/12/2018)
16 http://rdf4j.org/ (last visited 03/12/2018)
17 https://bitbucket.org/openrdf/alibaba (last visited 03/12/2018)

complex RDF storage applications by supporting a set of modules for RDF database abstractions. It introduces several features that are not present in Sesame/RDF4J, enabling users to further increase their application's applicability. By supporting object-oriented features like specialisation, which is often overlooked in other ORdfM implementations, Alibaba targets the mapping concept and its advantages that have been described above.

An evaluation of the applicability and usefulness of ORdfMs has been given by Quasthoff, Sack, and Meinel [134]. They conducted an experiment in which participants (coming from the fields of computer science) were asked to solve different tasks on the one hand with the utilisation of a ORdfM and on the other hand with the Jena library, which only allows direct access to RDF triples. The results show that the participants prefer the access to RDF data supported by an ORdfM as they felt it was easier to do. Next to this, the solving of the tasks in average took less time with the application of a mapping. It was also oftentimes mentioned, that maintenance was estimated to be much lower with the application of an ORdfM.

In summary, the concept of ORMs/ORdfMs has many advantages and can help the development and usefulness of applications, which is supported by the evaluation shown above. This holds true especially in the fields of persistence and convenience of its associated processes. With the vision of the MICO Project described in Section 1.2 and Section A.1 in the Appendix, the corresponding application implemented in the project poses a perfect fit for the incorporation of an ORdfM. With its potential to implement complex mapping features but at the same time low effort to adapt its functionality, Alibaba constitutes an excellent basis for the implementation work of Anno4j of this thesis, which will be described in the rest of this chapter. Benefits and advantageous factors will be explained throughout the Anno4j sections, the overall impact of Anno4j on the MICO Platform implementation will be described in Chapter 5.

4.2 ANNO4J - AN OBJECT-RDF-MAPPING LIBRARY

Anno4j [22] [21] is an ORdfM library implemented in Java, which supports a idiomatic mapping between RDF triples or data and Java objects. The library is implemented as an open-source project, can be found at GitHub[18], and it can freely be used under an Apache License Version 2.0[19]. Next to the GitHub page, Anno4j is also distributed as a Maven[20] dependency and can be accessed for example via the

18 https://github.com/anno4j/anno4j (last visited 03/12/2018)
19 http://www.apache.org/licenses/LICENSE-2.0 (last visited 03/12/2018)
20 https://maven.apache.org/ (last visited 03/12/2018)

Maven Central Repository[21], which enables convenient incorporation and usage of the library in Java projects.

In order to give and overview over the technical features of the Anno4j library and understand their interplay, a short description about the following sections is given here:

- The basic functionalities of an ORdfM implementation like the Anno4j library are given by its persistence and querying functionalities, as it is the focus point of both writing and requesting RDF data from a Semantic Web database. Therefore, Section 4.3 will give an in-depth description about the persistence procedure of the library, also highlighting technical foundations. Afterwards, Section 4.4 enlists the querying capabilities that allow users to apply different levels of granularity, offering easier but imprecise queries up to a more detailed and therefore more precise querying.

- With the core functionality covered in the prior sections, it is important to cover associations that the ORdfM concept has with its underlying database in the processes of persisting and querying data. Various database requirements and concepts exist, with the most commonly known ACID criteria as one of the present representatives, that should be fulfilled when working with a database in order to support both convenient and secured working. Therefore Section 4.5 explains how the Anno4j library supports such database requirements and in what way. Focus is thereby posed on transactional behaviour supported by the library and innovative validation features that especially can help with the data correctness and data consistency. Here again, different possibilities to do so are supported in Anno4j.

- The sections mentioned beforehand cover all necessary information in order to work with Anno4j and a Semantic Web database in regards to ACID compliance. Hence, the following sections can focus more on the overall abstraction layer concept that underlies the communication with the database when working with the library. The communication is based on the so-called domain model, Java classes that represent the mapping between the object-oriented classes and the RDF data that they are translated to, and vice versa. These domain models can potentially become large and thereby more complex, and they hence can be error-prone and time-consuming to create by hand. The Anno4j library introduces the Anno4j Generation Tool in Section 4.6, which can be used to turn RDFS or OWL schemata, which are often existing next to a Semantic Web database, into such domain models in an automated way. This reduces the initial effort to work with a given ontology and Anno4j to a minimum.

21 https://search.maven.org/ (last visited 03/12/2018)

Next to this enhancement, the generation tool is also capable of generating Web- and REST compliant resources that allow the automated composition of a RESTful application that enables to work with the created domain model also via commonly known HTTP requests and therefore support Web interoperability. This is documented in Section 4.6.3.

- That followed, Section 4.7 introduces several convenience features that increase the usability of the overall library, namely contextualisation of persistence and querying, serialisation possibilities for both input and output, and querying plugins that allow the introduction of own SPARQL expressions in combination with an interpretation logic in order to facilitate and fine-tune the querying capabilities of the library. Section 4.8 summarises all results and contributions of the whole chapter and enlists envisioned additions to Anno4j which can further increase the applicability of the overall library.

In order to set up a given application to work with Anno4j, the connection to the database needs to be established. The Anno4j implementation can connect to every RDF triplestore that is conform to the SPARQL 1.1 specification [71]. This is done by supporting the *query* and *update* endpoint URLs of the triplestore (which are generally exposed and open in order to work with the respective application) for the Anno4j Java object. Oftentimes, both of these endpoints are combined into a *SPARQL* interface URL, which can also be added to the Anno4j library instead. Listing 16 shows an example how this looks like.

```
1  String queryUri = "http://petsandowners.org/database/query";
2  String updateUri = "http://petsandowners.org/database/update";
3  String sparqlUri = "http://petsandowners.org/database/sparql";
4
5  // Triplestore with query and update endpoint
6  SPARQLRepository queryUpdateRepository =
7          new SPARQLRepository(queryUri, updateUri);
8  Anno4j anno4j = new Anno4j(queryUpdateRepository);
9
10 // Triplestore with sparql endpoint
11 SPARQLRepository sparqlRepository = new SPARQLRepository(sparqlRepository);
12 Anno4j anno4j = new Anno4j(sparqlRepository);
13
14 // Without a supported endpoint, a local instance with in-memory database is
       created
15 Anno4j anno4j = new Anno4j();
```

Listing 16: Instantiation of an Anno4j Object.

With the sole addition of a `SPARQLRepository` object (which represents the RDF triplestore and enables the connection to it) to the Anno4j instance, objects can be created at and queried from that database instance. How this is done will be covered in the following sections. There is also the possibility of not adding a remote

repository to an Anno4j object, which then creates a local in-memory database that will hold respective data. This is shown in line 15 of Listing 16. Its data content however is lost as soon as the respective Anno4j instance is invalidated. This enables lightweight and quick implementation possibilities mainly for testing purposes, as it is not managing persistent data as described in Section 4.1. Another configuration possibility allows to persist the mapping information like class, property, and type definitions, which are currently implemented in the respective Anno4j application. These information can sometimes be useful, but also clutter the database as they constitute many triples.

Next to the general ORM/ORdfM advantages that have been enlisted in Section 4.1, the application of the Anno4j library in a Java application also poses the following generic advantages:

VERIFICATION OF INPUT Through the approach of creating and querying RDF information via Java objects, a natural way of verification is given. Java objects can only be created as dictated by their class file, so the eventually created RDF triples are always conform to the underlying schema from a structural point of view. Further verification can be added to the Anno4j classes via Java Annotations implemented specifically for this library (see Section 4.5), which control the semantical and logical soundness of the RDF data.

REDUCE PRE-KNOWLEDGE OF THE USER Basic Java POJOs and the basic Anno4j Classes are mainly composed of straight forward data interaction methods, in general a getter and setter for every supported attribute. Hence they are simple to understand and use, making the entry to the underlying schema much easier than learning it from an RDFS or OWL schema. Combined with the feature of verification, users need much less time to start creating (sound and correct) RDF data. Additionally, the Anno4j library and its verification implementation is able to give feedback to the user in in the event of faulty input data. This further increases the learning process of the user while keeping the data status consistent as well.

STANDARDISED SERIALISATIONS Anno4j can both read and write RDF data from and to various standardised serialisations (see Section 2.2.5). This broadens the applicability of the library to also support "written triples" in both directions of persisting and querying.

SCHEMA UPDATES An update to an existing schema can be very difficult, in addition to the fact that it might also have an impact on existing data and defined queries. In contrast, updating Anno4j classes to introduce a change in the underlying schema is much

simpler, queries can be adapted automatically as they also base on the Java objects. A concurrent update to an existing database can be established via Java processes.

DATABASE ACCESS TECHNOLOGIES RDF triplestores support some access functionalities via REST calls, but rely mostly on SPARQL queries to write and retrieve their respective data. This circumstance narrows the range of possible users of using Open Data, which does counteract its general purpose. With the addition of Anno4j the access possibilities get broadened, as not only SPARQL queries are possible, but also querying via Anno4j and LDPath (see Section 4.4) and a schema-oriented REST API that is conform to the defined mapping (see Section 4.6). Figure 25 visualises the access possibilities. Three layers on top of the database (the basic access layer of the database itself, with two added by Anno4j) automatically transform and pipe through an issued request over various technologies (from REST calls to Java POJOs (to LDPath) to eventual SPARQL queries). Important to note is, that "lower" layers are not blocked by preceding ones. This allows users to choose their desired way of communicating with the database, enabling a broad spectrum of interactivity.

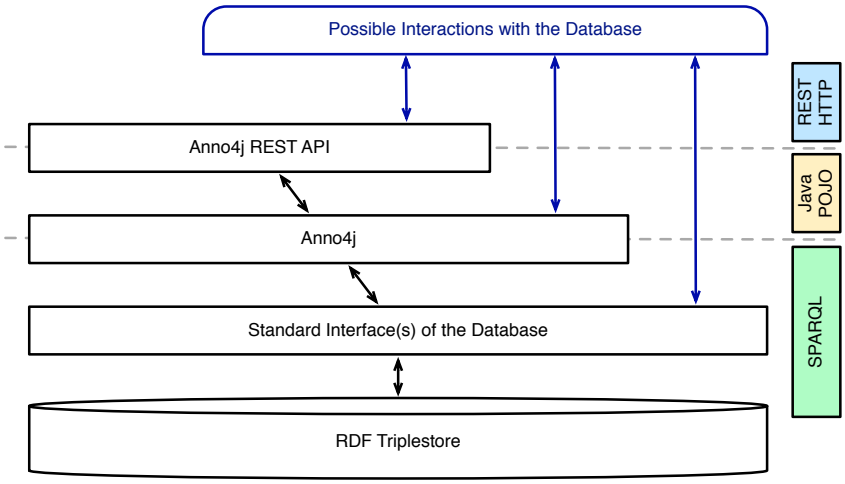


Figure 25: Access Possibilities for a Triplestore Augmented with the Anno4j Library.

Next to its generic advantages, the utilisation of the library also benefits the overall application by means of its technical features. These will be documented throughout the following sections, examples will cover Java code in order to show the basic appliance of the features. These examples will be concise and to the point to show the basic functionality to support given theoretical explanations, as extensive

code examples would exceed the frame of this thesis. However, these examples can be found on the Anno4j GitHub page[22], which illustrate the full potential of the library.

To give a description of the Anno4j library and its features, the following sections are structured as follows: Section 4.3 explains the core feature of persisting RDF data via Java. It therefore describes how the respective Java classes for Anno4j are created and how they can be enhanced with mapping metadata of Anno4j. Additionally, the basic functionality of this construct can be enhanced with different Java features by an Anno4j extension, increasing the ORdfMs possibilities for data creation and its functionality. Following this, Section 4.4 will document the other "side" of the mapping with the querying of existing data. To achieve this, the path-based querying language **LDPath**[23] [145] will be introduced, which is implemented via Anno4j to issue comprehensive queries to the connected database or local memory store.

After the "basic" features of the library have been elaborated, Section 4.5 covers database characteristics in the form of consistency and validity of the associated database status, which is also commonly known as the ACID criteria of databases. After the explanation, a description of how these requirements are also provided in the Anno4j library is given. Therefore, Section 4.5 explains the implemented transactional behaviour of the library and how it is applied, enabling consistency when working with Anno4j. These transactions can then be enhanced to also include validation checks that will test the to be added data for model conformity. Only when that is given, that data will eventually be persisted to the connected database. This is explained in Section 4.5.2. The validation checks rely on respective validation information to be present in the database that is associated in a given process. The validation information can be incorporated in the data status itself or it can be conveniently supported next to the Anno4j mapping, which is shown in Section 4.5.3.

All of the aforementioned features indicated that most implementation is done by hand. This is not only time consuming but also error-prone. Anno4j supports the so-called **Anno4j Generation Tool**, which is able to support many building blocks of the Anno4j library by inputting an RDFS or OWL schema file only. The overall application of the tool and what it is able to generate is shown in Section 4.6 and Section 4.6.1. Next to this, Anno4j is also able to alleviate its defined metadata mapping to work as a RESTful API, allowing users to issue their interactions with the RDF database via common HTTP methods that are aligned to the associated mapping.

The last technical section of this chapter mentions smaller additions to the Anno4j library in Section 4.7. It starts by explaining how the

22 https://github.com/anno4j/anno4j (last visited 03/12/2018)
23 http://marmotta.apache.org/ldpath/ (last visited 03/12/2018)

basic RDF feature of subgraphs or named graphs is included in the functionality of Anno4j. Also, the overall mapping is extended with a parsing functionality that allows users to write Anno4j objects to RDF serialisations an vice versa. Own querying plugins allow for a richer, more comprehensive, and personally fine-tuned querying functionality of the library.

Overall, one of the core claims of the library is to facilitate the persistence interactions with RDF data for users and their respective applications. There are various starting points for this facilitation, which can be summarised in usage scenarios or use cases. In the following, some of those envisioned scenarios are enlisted that focus on single activities or processes that are to be implemented and supported by the Anno4j library, depicting also the advantage that Anno4j enables. In many cases, these scenarios can be combined or lined after one another, in order to produce an even richer use case or application scenario. Some scenarios can depend on a preferred access technology for the database, but the following sections will show that there are redundant and interchangeable processes possible and the other access technologies are not prohibited, as seen in Figure 25. In general, the scenarios share the same approach that a *user* wants to access an *RDF database*. The envisioned use cases are the following:

NON-SEMANTIC WEB EXPERT Derived from the insights that have been already mentioned in this section and preceding sections, this is the most straightforward scenario, featuring a user that does not have any or only limited knowledge of Semantic Web technologies but adequate skills in Java. Therefore it is envisioned to support this kind of user to interact with the respective databases in an idiomatic fashion. Anno4j allows these users to interact with the database via other technologies than SPARQL queries and retrieve Java POJOs. This can be done with the basic Anno4j querying functionalities, which only needs Java knowledge (see Section 4.4), or if generated the via HTTP request to the Anno4j REST API (see Section 4.6.3) which is also presumable more familiar to the general user.

UNKNOWN DATABASE CONTENT/STRUCTURE Oftentimes when a database is expose for open access, its semantical description as well as the data structure is explained on a very high level basis. This can hamper potential users or enforce them to invest a large amount of time in order to interact with the database properly. In this scenario, Anno4j can help to facilitate this process. The database can offer its schema (either as RDFS or OWL schema) which should be exposed to the users. The user can then generate the data model with respective Anno4j classes to not just interact with the database, but also get a much quicker insight into the data structure.

CONTRIBUTION TO THE DATABASE CONTENT Next to the first two scenarios enlisted, another natural use case exists in which users want to contribute to the content of an already existing database. Next to the technologies that facilitate the communication with the database mentioned above, the persistence of data does also need to be supported. Through the generation of Anno4j classes via the schema supported by the database, users are automatically "forced" into the database's data structure, enabling convenient access to the data both for persisting and querying. Next to this, if supported by the schema, the validation properties introduced through Schema Annotations can further increase the validity of the input data produced by a user.

MERGE HETEROGENOUS DATABASES As one of the more casual scenarios, the merging of heterogenous databases is possible via the implementations done in Anno4j. Parsing the schemata of multiple databases allows to synchronise the data structures more easily, as Java POJOs are supported. This process is way more convenient than synchronising the RDF schemata in combination with the existing data. For the data, new Anno4j Classes can be created, filled, and persisted through class merging, leading to a new consistent and combined data status.

WORKFLOW BASED USE CASE This type of scenario is the most similar to the MICO vision described in this work, which has been introduced in Section 1.2. This scenario can be interpreted in two different ways. The "singular" scenario depicts a user who wants to collaborate in a workflow designed around an access point, that incorporates the database of the scenario. For the collaboration, a mixture of the functionalities of the scenarios above is necessary, as the user needs to understand the data structure of the workflow in order to query and use the data, as well as make a contribution to the data status itself. Additionally, the user does not only want to create data that is conform to the platform's, but may also extend the data structure, as he introduces new forms of metadata to the workflow. The understanding, application, as well as the contribution to the database has been covered above. In order to extend the data structure of the database, it is envisioned that the user can extend the currently underlying schema with his or her own features. The new extended data structure can then be synchronised back to the to the central access point. This scenario can also be extended to a "plural" use case, like the MICO platform (see Section 5.2), in which multiple parties do this cycle of actions and especially create a commonly agreed upon data structure. This way, every party can conveniently interpret the results of their predecessor(s) in order to apply the contained information in order to

possibly generate further information. This eventually can create a rich and diverse metadata background.

4.3 CREATION OF METADATA WITH ANNO4J

The most fundamental step for an ORdfM mapping and therefore also the Anno4j library lies in the persistence of (RDF) data. This means conceptually, that possibly interconnected Java objects are created and consecutively turned into RDF triples that are then persisted at a respective triplestore automatically by the library. This section gives insights on how this process looks like in the Anno4j library, what has to be done by a user, and what internals of the library take part in this workflow. Anno4j is built on the existing Alibaba library that has been described in Section 4.1, which supports the persistence functionality.

Before the explanation for metadata creation with Anno4j can be given, some definitions are needed. **Anno4j Classes**, which will be explained by text and exemplary code later, resemble the mapping object for the ORdfM of the object-oriented programming language Java. (Technically, these are implemented as Java interfaces, but the naming is more convenient this way, as will be seen later.) Just like in normal Java fashion, objects can be created in regards to a given Anno4j class, which will be called **Anno4j instance objects** or just objects of class x. So-called **Partial Classes** can implement the interface of an Anno4j Class to extend the functionality of the respectively created objects.

The final functionality of a given object, its **behaviour**, determines "what the object can do": what methods are supported, to what RDF information it is mapped, which RDF class(es) it has, and so on. This is defined by the class hierarchy that the Anno4j class is in and the added Partial Classes that are added to the respective classes. Every associated Anno4j and Partial Class of a strand contributes its own behaviour to the overall behaviour of a created object. For those objects, the Anno4j library implements a proxy pattern[24] [67] to increase the applicability of the objects, mainly in terms of scalability. This is necessary, as the scalability for such mapping implementations can be critical [176], as proxy creations in larger class hierarchies can be costly and it does put additional timely effort on implemented lazy evaluations for example. Therefore the Anno4j library implemented automated functionalities that can bypass these circumstances. This will be covered shortly in this section and in more detail in Section 4.6. An evaluation of the time savings will be shown in Chapter 6.

How actual metadata can be created via the Anno4j library, and what internal processes take part, is shown in Figure 26.

24 http://www.oodesign.com/proxy-pattern.html (last visited 03/12/2018)

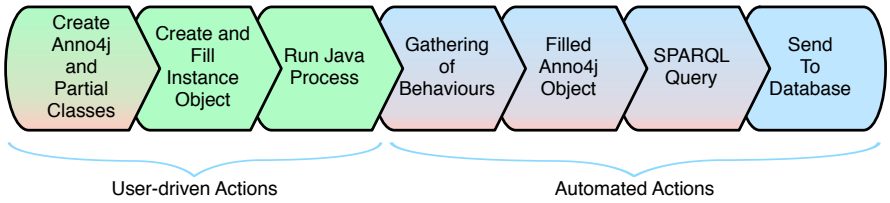


Figure 26: Sequence of Steps Executed for Creating Metadata with Anno4j, Including Internal Steps of the Library.

Figure 26 shows the steps that are undertaken when a developer wants to make use of the Anno4j library in one of his Java projects. This overall process is composed of different kinds of activity which are reflected by the colours applied in the figure. The green entities symbolise actions that in general have to be undertaken by the developer, hence these are the steps that are not supported by automated functionalities by the library. In contrast, the blue entities are the ones that run fully autonomously, executed by the library with the input provided by the green preceding steps. Both of these action classes have steps that are supported by the functionalities of the Alibaba library. The steps that are illustrated with a red border at the bottom are specifically enhanced via the Anno4j library. These enhancements and the basic functionality of every workflow step are explained in detail in the following enlistment.

CREATE ANNO4J AND PARTIAL CLASSES The essential beginning of every metadata production via Anno4j lies within the creation of **Anno4j Classes**, which can also be enhanced by the **Partial Classes**. A basic Anno4j Class resembles the "building plan" for RDF instances with an RDF class assignment as well as a list of its possible relationships and properties. Hence, this encapsulates one direction of the mapping between Java objects and RDF data.

Listing 17 shows an example of a simple Anno4j Class that resembles the information of the RDF class named "`pao:Cat`" that has been introduced throughout Section 2.2 with the addition of having "cat friends" via the relationship "`pao:hasCatFriend`".

```
1  // RDF Class definition
2  @Iri(PAO.CAT)
3  public interface Cat extends Animal {
4
5      // Getter and setter for the foaf:age property
6      @Iri(FOAF.AGE)
7      int getAge();
8
9      @Iri(FOAF.AGE)
10     void setAge(int age);
11
12     // Getter and setter for the pao:getOwner relationship
13     @Iri(PAO.HAS_OWNER)
14     Human getOwner();
15
```

```
16      @Iri(PAO.HAS_OWNER)
17      void setOwner(Human owner);
18
19      // Getter and setter for the pao:hasCatFriend relationships
20      @Iri(PAO.HAS_CAT_FRIEND)
21      Set<Cat> getCatFriends();
22
23      @Iri(PAO.HAS_CAT_FRIEND)
24      void setCatFriends(Set<Cat> friends);
25
26      // Additional behaviour that is implemented by a respective Partial Class
27      void addCatFriend(Cat friend);
28  }
```

Listing 17: Anno4j Class for the "pao:Cat" RDF Class.

Line 3 of Listing 17 implements the classic interface header of a common Java interface. The name for the interface can be freely chosen (according to common Java best practises) and is mainly used for interaction with the Java objects of the mapping. Nevertheless, this line shows how class inheritance can be implemented via the extension of the "Animal" interface (which is another Anno4j Class implementing the "pao:Animal" RDF class). This heritage information will also be represented in resulting created RDF data. The RDF class association is defined in line 2, showing a Java Annotation "@IRI" that assigns the String argument of the Java Annotation to the "rdf:type" relationship of created instances, which in terms defines their RDF class. In this example, a simple namespace class "PAO" is implemented for convenience reasons, which contains all IRIs - represented in Java as Strings - that are used in the given namespace. This allows shorter terms to be used in the Java implementations, just like it was done with the namespacing in the RDF serialisations in Section 2.2.5.

Then, in the lines 7 and 10, 14 and 17, as well as 21 and 24 there are blocks of code that define pairs of getters and setters, each defining different relationships and one property for the Cat Anno4j Class. Depending on the datatype that is associated with the respective pair, one can define a property when basic datatypes are assigned, or a relationship for complex datatypes. Examples for this can be seen in the first and second pair in Listing 17 for the assignment of the "foaf:age" property and the "pao:hasOwner" relationship, which have a basic integer or a complex Human Anno4j class as parameters respectively. As with the RDF type association for the overall Anno4j class, the RDF association of relationships and properties is also supported via the Java Annotation "@IRI". These must be defined above both the setter and getter of a respective pair and contain the IRI as a String parameter. Examples can be seen in lines 6/9, 13/16, and 20/23 of Listing 17 respectively.

In contrast to the first two additions to the Cat Anno4j Class that were defined to have single values, it is also possible to add multiple values to a given property or relationship. This is done by adding

a Java Set to the respective getter/setter pair, which can also be either of a complex or primitive Java type. An example can be seen in Listing 17 with the "pao:hasCatFriend" relationship, which can have multiple instances of a Cat added.

Up to line 24, Listing 17 showed the basic shape of an Anno4j Class. However, in line 27 there is also an extension to this basic implementation apparent, which allows to add single instances to the "pao:hasCatFriend" relationship, rather than the whole set via the basic setter. This can be supported by the Anno4j library to increase the basic behaviour of produced Java objects and allow for more comprehensive implementation. Because of this, it is also possible to make use of other Java functionality and logic to further increase the usability of the mapping. The respective implementation is done in a **Partial Class** that is supported in addition to the given Anno4j Class by implementing the relative interface. An example how this looks is shown in Listing 18.

```
1  // Java Annotation that induces an Anno4j Partial Class
2  @Partial
3  public abstract class CatSupport extends AnimalSupport implements Cat {
4
5      // Implemented for the method defined in the Cat class
6      @Override
7      public void addCatFriend(Cat friend) {
8          Set<Cat> catSet = new HashSet<>();
9
10         Set<Cat> currents = this.getCatFriends();
11
12         if(!currents.isEmpty()) {
13             catSet.addAll(currents);
14         }
15
16         catSet.add(friend);
17         this.setCatFriends(catSet);
18     }
19 }
```

Listing 18: Anno4j *Partial* Class for the "pao:Cat" RDF Class.

Code-wise, the most important thing to add to an Anno4j Partial Class is the Java Annotation "@Partial", which is needed in the step of behaviour gathering, as the Anno4j Java process needs to "find" these annotated classes at the point of time when Anno4j objects are created. Next to this, the class declaration needs to be "abstract" and the respective interface, in this case the "Cat" class, needs to be implemented. The Partial Classes also build up a hierarchical structure, so this class needs to extend the "AnimalSupport" class, which is the Partial Class for the semantically higher positioned "Animal" Anno4j Class. In the body of the class, methods defined in the respective interface can be implemented. In the example of Listing 18, starting from line 7, the method is implemented whose signature is defined in Listing 17 which allows the addition of single "pao:hasCatFriend" relationships rather than in full sets only.

The functionality applied in Listing 17 and Listing 18 is rather simple in order to show a minimal functional example of the Anno4j structure for basic mapping objects. However, especially the Partial Classes can include richer and more comprehensive Java features if needed. Respective basic setter and getter methods defined in the interface can be called from the Partial Class in order to also produce associated triples. Other Java processes can also be merged technically, for example to trigger a calculation that eventually sets the value of a given RDF property.

The creation of Anno4j Classes itself can be done by hand, but it has been mentioned at several occasions up to now, that Anno4j also supports the possibility of generating these classes via Java code generation and the Parsing Tool of Anno4j. With the input of an RDFS or OWL schema, corresponding Anno4j Classes as well as their Partial Classes are created accordingly. This topic will be covered in Section 4.6.

CREATE AND FILL JAVA OBJECT The next step of metadata creation via Anno4j encapsulates the process of writing Java code that uses the Anno4j Classes and Partial Classes defined in the step above. With the only difference of not creating Java objects via the commonly known "new" operator, respective objects have to be instantiated via an Anno4j instance, which can be connected to a triplestore or run with a local in-memory database as shown in Listing 16. The objects created are being used like common Java objects, which underlines the convenience of use for non-Semantic Web experts. A simple example, using amongst others the Cat Anno4j Class defined in Listing 17 can be seen in Listing 19.

```
1  Anno4j anno4j = new Anno4j();
2
3  Human bob = anno4j.createObject(Human.class);
4  Cat daisy = anno4j.createObject(Cat.class);
5  Cat eve = anno4j.createObject(Cat.class);
6
7  Cat cesar = anno4j.createObject(Cat.class);
8  cesar.setAge(5);
9  cesar.setOwner(bob);
10 cesar.setCatFriends(new HashSet<Cat>(){{add(daisy); add(eve);}});
11 // With the behaviour of the Partial Class, above line could be exchanged with:
12 // cesar.addCatFriend(daisy);
13 // cesar.addCatFriend(eve);
```

Listing 19: Simple Application of the Cat Anno4j Class Shown in Listing 17.

In the example of Listing 19, after an instance of Anno4j is instantiated in line 1, one Human entity and two Cat entities (which will serve as friends of our known Cat "cesar") are created, which will serve as dummy objects and therefore will not be filled with further information. Line 7 creates the Cat object "cesar" that will have additional information in the form of its age (line 8), its owner (line

9), and his Cat friends (line 10). All of this is done with commonly known Java implementation. The graph of the resulting RDF triples can be seen in Figure 27.

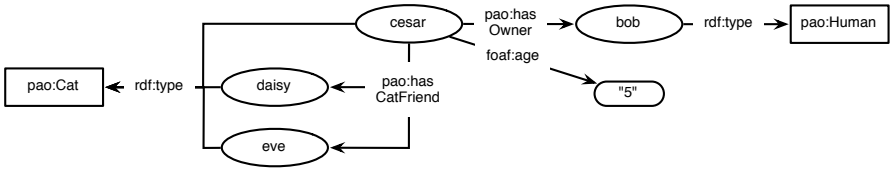


Figure 27: Resulting RDF Graph of the Executed Anno4j Code Shown in Listing 19.

Another example in the domain of multimedia and the MMM, which has been discussed in Section 3.3, is shown in Listing 30 in the Appendix. Its resulting RDF graph and triples are shown in Figure 61 in the Appendix. An MMM Item is created and enhanced with two Part Annotations that represent a multimedia low-level feature in the form of a ColorLayout analysis (a descriptor featured in the MPEG-7 standardisation that shows color distribution [46] [96]), and a high-level feature in the form of an animal detection.

RUN JAVA PROCESS Just like the definition and the instantiation of Anno4j objects is done in classic Java fashion, the overall process behind it is also set in the Java world. This means, that any Java process can incorporate Anno4j instances and objects and therefore create metadata at the corresponding database. Once the Java process is executed, respective metadata information is created and eventually persisted. This is the last step that a developer is involved in. The steps after this one are processed automatically.

GATHERING OF BEHAVIOURS When the Anno4j process is triggered, the respective Anno4j instance dynamically generates the entire metadata model that is induced by the defined Anno4j Classes and Partial Classes in order to be able to generate proxies of the mapping later. This is a required step as the model definitions are not known to the Anno4j instance before the point of time of the build, and the behaviours of every Anno4j object need to be gathered in order to compile an eventual mapping between the Anno4j (Java) object with its implemented functionality and the resulting RDF data.

The creation of an Anno4j Class instance triggers the compilation of the overall behaviour for that representative. The set of behaviours that is gathered for every object results from the implemented hierarchy of both the Anno4j Classes and Partial Classes. In addition to a standard behaviour of general mapping entities of Anno4j, each of those classes imposes one strand of behaviour to a given object's overall behaviour. All of them will eventually be combined and merged down to generate the final behaviour.

For an exemplary object of the Cat Anno4j Class shown in Listing 17, the behaviours gathered would be the standard behaviour, two for the Anno4j Classes "`Animal`" and "`Cat`", and two for the respective Partial Classes "`AnimalSupport`" and "`CatSupport`" (shown in Listing 18) enabling the final object to have behaviours (methods, type associations, ...) from both concepts.

This composition process requires Java reflections[25] to scan through the defined Anno4j Classes and Partial Classes. The merging of behaviours to create a proxy object, especially in large class hierarchies, is time consuming which can result in long waiting times when working with Anno4j. This circumstance is commonly known in the fields of ORMs / ORdfMs [176]. As a solution, a so-called **Proxy Generation Tool** has been implemented which allows to generate the eventual proxy objects in advance. These can then be supported as .jar-file to the respective Java process in order to avoid the necessary own creation of proxy objects of the library. Using this, when working with a non-changing model, process times can be reduced immensely. This saving of time has been evaluated in an experiment, applying the project use case of the ViSIT project[26], which uses the Conceptual Reference Model CIDOC CRM version 6.2[27]. This experiment in combination with an evaluation and project description is shown in Chapter 6.

FILLED ANNO4J OBJECT As shown above, the result of the behaviour gathering process is the generation of the internal mapping that connects the Java objects to the eventual RDF data that is to be created. When an Anno4j instance object is created, a proxy of the respective mapping is created, whose methods can then be accessed to add metadata information in the form of properties and relationships. The overall result is then a "filled" Anno4j instance object (internally a Java object with set attributes) that can be persisted to a given database in the next step.

SPARQL QUERY The next superordinate step consists of the conversion of the filled Anno4j objects to RDF data. This is done by the Anno4j and Alibaba library by transforming the filled objects into respective SPARQL INSERT DATA[28] queries that are sent to the database automatically.

SEND TO DATABASE The generated SPARQL queries described above are sent to the database that has been connected to the Anno4j

25 https://docs.oracle.com/javase/8/docs/technotes/guides/reflection/index.html (last visited 03/12/2018)
26 http://www.phil.uni-passau.de/dh/projekte/visit/ (last visited 03/12/2018)
27 http://www.cidoc-crm.org/Version/version-6.2 (last visited 03/12/2018)
28 https://www.w3.org/TR/2013/REC-sparql11-update-20130321/#insertData (last visited 03/12/2018)

instance or, if Anno4j was created locally, an in-memory database. The connection to the database does not require any further actions by the developer, simply the creation of the respective Anno4j instance as shown in Listing 16 is enough.

4.4 QUERYING OF METADATA WITH ANNO4J

Now that the first "direction" of the RDF metadata mapping possible with Anno4j has been seen in Section 4.3, this section will introduce the other half: the **querying** of metadata via Anno4j that is stored in a database. Conceptually, the design of the querying functionality followed the same intention as it was done for the persistence. The main goal in mind is to reduce the barrier that Semantic Web technologies pose to developers that are not familiar with them, which could otherwise eventually hamper and deny additional contributors to a common knowledge base. Once again, the Anno4j library tries to alleviate the process of querying for the developer in a way that he has to mainly work with concepts they are familiar with, in this case Java objects, rather than having to deal with the Semantic Web technologies. To achieve this, Anno4j supports two different ways of querying: basic and extended. In contrast to metadata creation however, at this point the Semantic Web technologies cannot be fully avoided for extended querying purposes, as querying at some point requires a metric of some sort in order to formalise a respective query and its querying criteria in order to retrieve desired metadata. However, the Anno4j library supports a more convenient possibility for Semantic Web querying instead of the commonly applied SPARQL: the path-based querying language **LDPath**[29] [145] [5].

Basic Anno4j querying allows straightforward querying for the requirement of querying *"all representatives"* of a given Anno4j Class. The basic version does not require Semantic Web skills, but does lack technical advantages, as querying criteria does have to be evaluated "by hand", as a list of Java objects will be returned that have to be searched through with Java functionality. This does also demand more time, as all representatives of a queried Anno4j Class will be returned without any restrictions.

LDPath is similar to XPath [49] and the SPARQL Property Paths [152] and is particularly well-suited for querying RDF data from the LOD. Compared to classic SPARQL that defines its queries by supporting graph patterns that are then matched against the graph stored in the respective database to retrieve results, LDPath allows to construct queries by defining a path that is evaluated from a given starting point entity in the graph. If the defined path with associated criteria can be found, the respective entity will be contained in the result set. These path-based queries are supposedly easier for humans to

29 http://marmotta.apache.org/ldpath/ (last visited 03/12/2018)

issue than SPARQL queries, especially combined with the graph-like view that has been applied over the course of this work. Important to note here is, that although the query features by the Anno4j library are present, the respective database access is not limited to it, as explained earlier and shown in Figure 25. Classic SPARQL queries can still be issued at standard database interfaces, allowing users to freely choose the access technology and therefore operate with various use cases.

The path queries defined in LDPath combine the edges present in an RDF graph with a broad variety of tools and various functionalities to form comprehensive queries. Amongst them are property selections, reverse property selections, wildcard selections, self selector, path traversal, unions, intersections, recursive selections, tests (for language, type, path value, and path existence), and even own functions can be implemented and used.

Implementation-wise, the LDPath library is built up in a structured and modular fashion, which allows developers to include only modules of it that are relevant to them. Amongst those modules are some core modules in combination with backend modules, in order to make LDPath work with different databases conveniently, as well as extension modules that broaden the applicability of the querying language. With Anno4j, the required LDPath dependencies are enabled automatically, and the path criteria can be issued via the **QueryService** Java object of the library. This combination extends the already rich querying language with an important feature for our use cases via the Anno4j native implementation: testing for literal values at the end of paths, which can also be enhanced with different comparison operators.

The QueryService follows the Anno4j philosophy and offers access to Semantic Web technology, in this case the requesting of RDF data contained in a triplestore by using Java. Therefore, the QueryService is a Java object that can be created by an Anno4j instance. The QueryService object can be augmented with different configurations, containing prefix definitions (which then can be applied in the defined querying criteria), querying criteria formulated as LDPath expression, as well as limit and offset preferences for the issued query. This is done by the implementation of a Fluent Interface[30], which allows the chaining of respective configuration methods, as every call to the method returns a further specified instance of the QueryService object. Once all these are added to the QueryService, the `".execute()"` method can be executed, which issues the query that is created with all the currently associated information of the QueryService object towards the triplestore that is associated with the respective Anno4j instance that created the QueryService in the first place. This method

30 http://java-design-patterns.com/patterns/fluentinterface/ (last visited 03/12/2018)

also takes an Anno4j Class as parameter, which defines the starting point as well as the result type for the query to issue. So for example the call of the method ".execute(Cat.class)" issues its LDPath querying criteria starting from those RDF objects that have the respective RDF/Anno4j Class associated.

Next to the features listed above, the QueryService evaluates its queries in lazy fashion in order to improve the performance of the library. This means that data is only queried at the point that it is directly requested or necessary for the process. This especially holds when information objects become larger and/or there are many associations in place. Instead of a "full object", an object contained in an Anno4j query response contains only direct representations of defined properties, every relationship and therefore association to another object is firstly represented by the IRI of the related object. This is a lightweight presentation of the object which will be accessed and then "exchanged" with the real object once the respective information is requested. This is done automatically by the library and allows on average for quicker process and reaction times when working with a database and Anno4j. Additionally, query optimisations of the Apache Jena[31] framework are implemented and applied to every Anno4j query automatically. These incorporate optimised join orders as well as the optimisation of filter arrangement that is utilised in the respective query.

Listing 20 shows the basic utilisation of the QueryService in Anno4j. The only thing that has to be supported is the Anno4j Class that should be queried for. The result list contains all RDF instances that have the associated RDF type (in this case "pao:Cat").

```
1 Anno4j anno4j = new Anno4j();
2
3 Cat cat1 = anno4j.createObject(Cat.class);
4 Cat cat2 = anno4j.createObject(Cat.class);
5
6 // Query for all cats
7 QueryService qs = anno4j.createQueryService();
8 List<Cat> allCats = qs.execute(Cat.class);
```

Listing 20: Basic Querying Example of the Anno4j Library, Querying for All Persisted Cats (Representatives of the Cat Anno4j Class).

To show an example of both the QueryService and LDPath for extended Anno4j querying, the exemplary use case of an animal detection of the MMM is applied, which was covered in Section 3.3.3. The underlying metadata that poses the basis for the querying is shown in Figure 28, which shows the Part Annotation of an extractor result indicating that an elephant has been found on the input image.

An example code snippet of how to query the Part Annotation illustrated in Figure 28 can be seen in Listing 21. This has to be seen

31 https://jena.apache.org/ (last visited 03/12/2018)

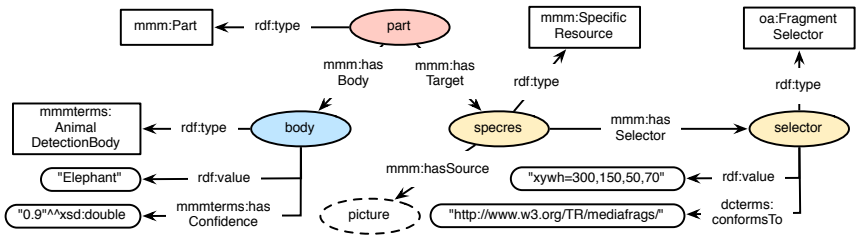


Figure 28: Showcase MMM Animal Detection Result Used as Anno4j Querying Example.

only as one of many different ways of retrieving the Part Annotation, as many different queries can contain this respective Annotation in their result set. The listing does also show the fluent interface that has been implemented for the QueryService, meaning that the configuration parts can be added after one another instead of adding every single one with an own code line (this can be seen in lines 2 to 6 in Figure 28).

```
1 QueryService qs = anno4j.createQueryService();
2 qs.addPrefix (MMMTERMS.PREFIX, MMMTERMS.NS).addPrefix(MMM.PREFIX, MMM.NS)
3     .addCriteria ("mmm:hasBody[is-a mmmterms:AnimalDetectionBody]")
4     .addCriteria ("mmm:hasBody/rdf:value", "Elephant")
5     .addCriteria ("mmm:hasTarget/mmm:hasSelector/rdf:value",
6                   "50,70", Comparison.ENDS_WITH);
7 List<PartMMM> result = qs.execute(PartMMM.class);
```

Listing 21: QueryService of the Anno4j Library Using LDPath Criteria. The Query is Issued to Retrieve the Part Annotation Shown in Figure 28.

In line 1 of Figure 28, the QueryService object is created, which is done via a method of given Anno4j object. Because of this, the queries issued by the service are automatically routed to the database connected to the Anno4j instance. The next 5 lines add various configuration parameters to the QueryService object in fluent interface fashion. Therefore, line 2 adds two new prefixes with corresponding namespaces, as the QueryService does not know them in advance, which lets the LDPath criteria be defined with commonly known prefixed terms. Lines 3 through 5 define various LDPath criteria for the query that will be explained below. Lastly, line 7 executes the QueryService by calling the execution method with the Anno4j Class parameter "`PartMMM.class`", as Part Annotations are to be queried. As indicated, the return type is a Java Set containing objects of the respective PartMMM Anno4j Class. In our exemplary use case, this Set would contain the Part Annotation illustrated in Figure 28.

The showcase application of the QueryService shown in Listing 21 does implement three different querying criteria that are issued. Semantically, the requirements that are defined are the following:

- As interest does only lie in the results of the animal detection extractor, the first criteria seen in line 3 requires the body node to be of the class "`mmmterms:AnimalDetectionBody`", as this is the way how extractor results are distinguished in the MMM. Therefore, the LDPath expression (starting from "`PartMMM`" nodes, so in this example from the node entitled "`part`") traverses the edge "`mmm:hasBody`" and then does a class type check for the respective class that is associated in brackets in the LDPath criteria ("`[is-a mmmterms:AnimalDetectionBody]`"). This path is visualised in Figure 29.

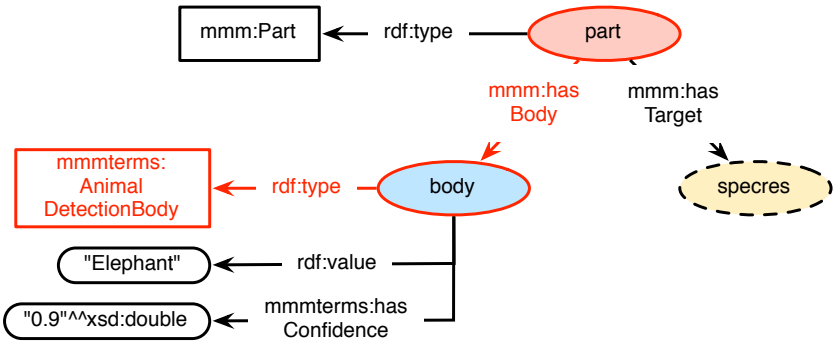


Figure 29: Illustrating RDF Graph for the First Criteria of the Query Defined in Listing 21 (Line 3), Which Supports the Semantic Feature of Searching for Results of an Animal Detection.

- The second criteria now further specifies the interest that the query owner does have in the results done by the animal detection extractor. Out of all the animals that are potentially detected on pictures, elephants are the ones that should be selected by the query. Hence the LDPath is traversed from the starting point over the relationship "`mmm:hasBody`" and then the property "`rdf:value`" in order to then select the literal that is supported there. The "`rdf:value`" of the "`mmmterms:AnimalDetectionBody`" reflects which animal the extractor has detected and therefore the addition of the criteria in line 4 of Listing 21 does support a second parameter which checks the value of the given property for equality to "`Elephant`" after adding the standard LDPath parameter. This path can be seen in Figure 30. As the check in the first criteria lies "on the path" used by the second criteria, both could be merged to the more complex LDPath expression "`mmm:hasBody[is-a mmmterms:AnimalDetectionBody]/rdf:value`" with a value check for an elephant, which would have the same semantic query requirement.

- Last but not least, a querying user could also have requirements on the picture, or more specifically the subpart of the picture,

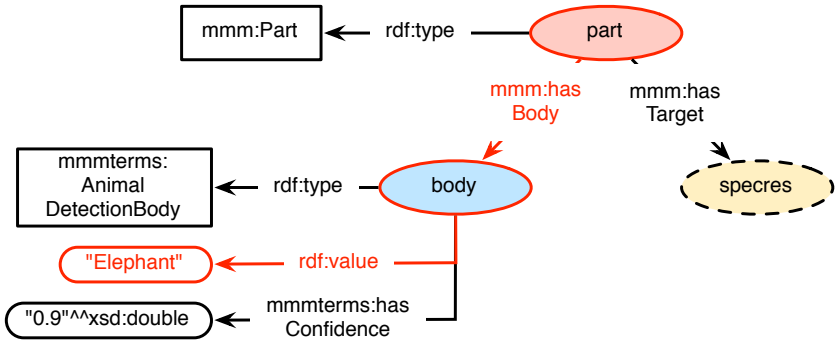


Figure 30: Illustrating RDF Graph for the Second Criteria of the Query Defined in Listing 21 (Line 4), Which Supports the Semantic Feature of Elephants Being Depicted.

that the animal detection was done on. This could for example be the requirement that the spatial fragment the elephant was found in should be "rather small". Point of interest for this criteria is the selector and more specifically the fragment value of said selector. The QueryService in Listing 21 therefore is augmented with the LDPath that traverses the relationships "`mmm:hasTarget`" and "`mmm:hasSelector`" to again read out the value that is supported for the "`rdf:value`" property of the given selector. In contrast to criteria two however, it does not check the contained String for full equality, but rather checks that it ends with "`50, 70`", so that the selection is done on a spatial fragment with width of 50 pixels and a height of 70 pixels, but it is not important where in the picture. This path is visualised in Figure 31.

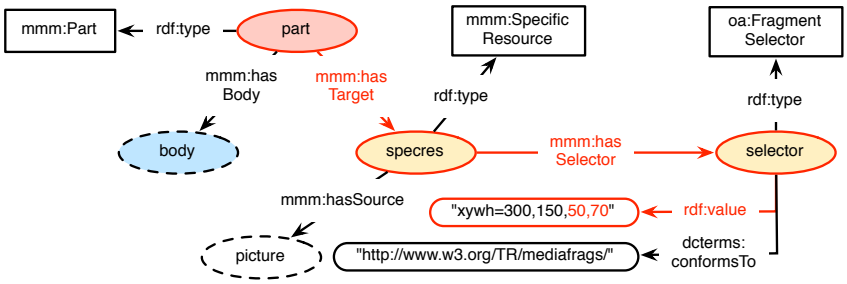


Figure 31: Illustrating RDF Graph for the Third Criteria of the Query Defined in Listing 21 (Line 5), Specifying the Detection to Be in a Rectangle of a Width of 50 Pixels and Height of 70 Pixels.

The Anno4j library supports both an easy to use basic querying functionality via Java objects, as well as a more comprehensive implementation via the path-based querying language LDPath. This suf-

fices the requirement for ORMs / ORdfMs of supporting a querying API (which was introduced in Section 4.1) that lets user fine-tune their queries for different purposes, in order to directly query for information that meets their current focus only. Path-based querying is supposedly easier to understand than pattern-based approaches like SPARQL, and therefore targets the overall-claim of the Anno4j library to reduce barriers for non-Semantic Web experts.

At this point, the essential features of an abstraction layer like the ORdfMs are discussed for Anno4j, which consist of the persistence and the querying of data. With these, users are able to both write and read RDF data from a given database. The next important requirement that needs to be covered for such interactions are database requirements that allow a convenient working with the database in the first place. These requirements ensure various criteria like parallel working or data consistency. Therefore the requirements met by the Anno4j library are discussed in the following sections.

4.5 ESTABLISHED DATABASE CONCEPTS OF THE ANNO4J LIBRARY

In the last sections, the basic mapping functionality of the Anno4j library has been covered. Java POJOs can be utilised to write RDF data to a database, while various querying functionality enables convenient requesting of RDF data from the database, which is returned in Java POJOs on the other hand.

What has not been covered are requirements that should always be present when working with a database of some sort. This circumstance exists for a very long time now, and various literature [69] [73] [51] emphasises the importance of implementing procedures or the whole system architecture to be fault tolerant. There are always sources present that can lead to different failures of the system or errors in the code and applications, even if those problems only happen in very rare situations. No matter the scenario, if a failure happens, it is essential that in particular the data and partially the actions that are currently done at the database are persisted, so that a consistent state can be ensured at any point of time and data that has been worked with cannot be lost or not reproducible. This is especially of importance the more sensitive the data gets, with the most common examples of finances or personal data.

One of the core concepts for database interaction that is mentioned in the literature above is the **transaction**. A transaction encapsulates multiple actions into a set of actions, that is either fully **committed**, so the actions do take effect on the data that is persisted currently in the database, or the set is **aborted** or **rolled back** and the actions do not take effect at all. This enables the database to be conform to the **ACID** concept [175], which is commonly known in the world of (relational)

databases. It stands for the following single concepts fulfilled by the database:

ATOMICITY The all-or-nothing philosophy described above for a given transaction. A set of actions is either fully committed, or not at all.

CONSISTENCY Consistency targets the conformity of a given database status, so the data is correct at a given state, oftentimes oriented towards a given data scheme or model. Each action executed with an underlying consistent status at the database always creates another, altered but consistent status. This consequently also holds true for transactions and their set of consistent single actions.

ISOLATION This concepts illustrates the fact, that the actions of a started transaction are not visible to other users and therefore hidden from other transactions, until it is fully committed and the change of the data is persisted. Many techniques about this concept deal with synchronisation.

DURABILITY Once a transaction is fully committed and therefore the data is altered at the database, the contained set of actions need to be fully persistent in the database from that point of time. This means that any subsequent malfunctions or errors may not have an effect on the happening of these actions. With the concept of consistency, every correct and committed transaction took part in the database, and the only way of reverting this state is issuing a transaction that exactly counteracts the originally executed actions.

The Anno4j library adapts the concepts mentioned above in order to enable ACID-like behaviour for its database communication. To give further insights into how this is implemented, Section 4.5.1 explains the basic functionality of the transactional behaviour that has been introduced to the library. Afterwards, Section 4.5.2 explains so-called **ValidatedTransactions**, which are an extension to the classic transaction. The validated variation allows to check the data that is supposed to be inserted by the transaction for validity towards the schema information that is present in the current state of the data model. Lastly, in Section 4.5.3 Java Annotations are introduced that can be added to the Anno4j Classes and Partial Classes in order to produce such validation information next to the defined metadata model in Anno4j. These additions to the Anno4j library contribute to the answer to research question 3.3.

4.5.1 *Supporting Transactional Behaviour in Anno4j*

The basic persistence setting for Anno4j is "auto-commit". This means, that every object that is created or altered during a Java process automatically gets persisted in the respective database. This workflow has shortcomings as described above, especially in the case of failures or crashes while the process is running and is not finished. In this case, only a part of the desired information would be stored, leading to an inconsistent or faulty database state.

Therefore the Anno4j library supports transactional behaviour as described in Section 4.5. This allows to bundle several Anno4j actions into sets of actions that are either committed when successful or rolled back if an error happens. Here again, Anno4j allows the developer to introduce rich Java features like error handling into his processes while supporting consistency of the database.

Listing 22 shows a short implementation example of the Transaction in Anno4j. The Transaction object is created via an Anno4j instance and hence is reflected on the database that is associated with the Anno4j instance. A Java interface *"TransactionCommands"* is implemented in the library and defines the methods and behaviour of the Anno4j and Transaction objects. Because of this, both objects behave the same way and offer the same functionality. Additionally, the Transaction class offers the methods "`.begin()`", "`.commit()`", and "`.rollback()`" for the respective essential transactional commands.

```
1  Anno4j anno4j = new Anno4j();
2
3  Transaction transaction = anno4j.createTransaction();
4
5  transaction.begin();
6  // Standard Anno4j code
7  // ...
8  Cat cesar = transaction.createObject(Cat.class);
9
10 QueryService qs = transaction.createQueryService();
11 // ...
12 transaction.commit(); // or eventual transaction.rollback();
```

Listing 22: Basic Transaction Implementation in Anno4j, Encapsulating Actions Done for Persisting and Querying Information.

Line 3 of Listing 22 creates the Transaction object via the defined Anno4j instance of line 1, which is started in line 5. Without a start to the transaction, executed commands would not result in the desired effect on the database. Line 6 through 11 now apply different persistence and querying actions that are described in Section 4.3 and Section 4.4 like creating an instance of the Anno4j Class "`Cat`" or creating a QueryService object. In contrast to earlier examples, this is done using the Transaction object instead of an Anno4j instance, which is an important detail. Last but not least, the transaction is committed

to the database in line 12 (or instead rolled back with the commented code).

4.5.2 *Validate Database Input with Validated Transactions*

The transactions implemented in Anno4j ensure the ACID criteria for database interaction via the library. This enables the database to be in a consistent and clean state at all times which is a fundamental cornerstone of today's services and applications.

Next to the consistency, another important requirement that can and always should exist for a database is the **validity** of the incorporated data. This validity is always motivated by some kind of rule set that is defined for that respective dataset and its context or use case, with the intention that the data is conform to the defined rules and therefore is denoted as being valid or correct. This validity of data is important, as the use of the data is compromised heavily if it does not correspond to the defined ruleset. If another user wants to apply the data in his own use case, they have to rely on the data being conform, otherwise the process of data application gets much more cumbersome and time-consuming.

The validity can be achieved or checked at different points of time of a process, and if it is not done before the data is inserted, there should always be processes that can verify the correctness of the data later on. In addition to the validation itself, it is always important to also propagate this information to the user, so he knows what the invalid action would be or what instances in the database are not corresponding to the defined ruleset.

As it has been seen so far, many databases in the Semantic Web support a vocabulary or ontology next to their actual data. This increases the applicability of the database, as users can learn and understand the structure of the data by parsing said ruleset, making the data itself more valuable. Therefore, the structural validity of the data and hence also the semantic integrity conveyed through the data model should always be assured. As has been discussed in Section 2.2, the main tools to implement a metadata model in combination with a specified structure by an ontology or vocabulary are RDFS and OWL. Next to the mentioned information, OWL does also support more thorough validity features like transitivity and multiplicity of RDF relationships. Anno4j incorporates some of those, originating from OWL Lite, in order to introduce validity checks, as will be seen in the next section. As the model information of both RDF and OWL is also formulated as RDF triples, the information can also be persisted at the database itself, which can be helpful at certain scenarios when a client also needs to consider this kind of information. In the Semantic Web world it is especially important to make sure that data validity is

given, as statements as discussed in Section 2.2 are always expected to be "true".

One of the most common ways of both creating a schema and even respective instance data are editors like Protégé [121]. They enable tool-supported possibilities to create and define classes with their properties and relationships, and accordingly also create instances of those classes. This process ensures complete validity of the created data, allows the persistence on the fly, and in most cases does also support information propagation to the user.

In contrast to editors, the Anno4j library introduces the validity feature to the mapping functionality, enabling a more technical approach to the requirement of data validity. This enables developers to always produce valid data and also incorporate the validation in their known way. Next to the "`Transaction`" Java interface that has been shown in Section 4.5.1, there is an extended interface called "`ValidatedTransaction`" which supports all the common transaction features with the addition of a validity check. This check is done on the schema information supported at the respective database that the validated transaction object is created on (again in relation to an Anno4j object). This check is fulfilled once the "`.commit()`" method of the validated transaction is called. Eventually, if the validity of the to be added data is not given, a "`ValidatedFailedException`" is thrown and the information is propagated to the Java process and the user. Therefore, this implementation satisfies the requirements for validity checks described above. The following Section 4.5.3 will show an example as well as the currently possible validators that can be applied in Anno4j and how.

4.5.3 *Schema Annotations for Data Validity*

Section 4.5.2 has explained what data validity is in a database, and it has also indicated how this can be achieved in Anno4j through the utilisation of a "`ValidatedTransaction`". These transactions make use of the schema information persisted in the respective database. Up to that point it was assumed, that the schema information is already present. This however is not always the case, and the insertion of this data is as important and has the same necessity to be valid as the "normal" data.

Because of this, the Anno4j library does support convenient ways of defining several possible validation requirements that are oriented towards OWL Lite and contribute to the validation of the database. Java Annotations are implemented that can be added to the Anno4j Classes, which will be automatically transformed into respective triples that are added to the connected database at startup. Doing so, the Anno4j instance can create "`ValidatedTransaction`" objects that validate those Annotations on the fly. These Annotations are called

Schema Annotations. Table 4 in the Appendix shows an extensive list of the Schema Annotations that are currently implemented in the Anno4j library.

To show an example of the Schema Annotations in combination with a "`ValidatedTransaction`" object in use, first of all the utilised Anno4j Class has to be adapted. Section 4.3 explained the basic setup of an Anno4j Class, and how getter/setter pairs need to be implemented in order to formalise an RDF relationship or property for the respective class. These can now be enhanced by adding a Schema Annotation defining the validity criteria. Listing 23 shows a code excerpt that extends the Anno4j Class "`Cat`" (seen in Listing 17) with a validation criteria on the relationship "`pao:hasCatFriend`".

```
1  @Iri(PAO.CAT)
2  public interface ValidatedCat extends Animal {
3
4      // ... Left out other setter/getter pairs
5
6      @Symmetric
7      @MaxCardinality(3)
8      @Iri(PAO.HAS_CAT_FRIEND)
9      Set<ValidatedCat> getCatFriends();
10
11     @Symmetric
12     @MaxCardinality(3)
13     @Iri(PAO.HAS_CAT_FRIEND)
14     void setCatFriends(Set<ValidatedCat> friends);
15
16     void addCatFriend(ValidatedCat friend);
17 }
```

Listing 23: Adapted Anno4j Class "`Cat`" from Listing 17, Introducing a Maximum Cardinality and Symmetric Requirement on the Relationship "`pao:hasCatFriend`".

Line 6 and 11 apply the Schema Annotation "`@Symmetric`", which semantically conveys the fact that friendship relations should always be in both directions. This enforces that a set relationship "`pao:hasCatFriend`" from instance "`X`" to "`Y`" is only valid when there also exists the same edge from "`Y`" to "`X`". The second validation criteria, defined in lines 7 and 12 respectively, issues the (only exemplary) requirement that cats can only have a maximum of three (symmetrically valid) "`pao:hasCatFriend`" relationships.

The adapted Anno4j Class "`ValidatedCat`" can be used as described in Section 4.3 and Section 4.4. Using it in combination with a "`ValidatedTransaction`" checks for the defined validation criteria shown in Listing 23. As described in Section 4.5.2, at the point of time when the implemented objects are to be persisted, the validation criteria are checked and an exception with a description of what went wrong is potentially given to the user. Listing 24 shows an example, implementing "`cesar`" as the starting point and various additional information to showcase the validation check.

```
1  Anno4j anno4j = new Anno4j();
2
3  ValidatedTransaction transaction = anno4j.createValidatedTransaction();
4
5  transaction.begin();
6
7  ValidatedCat cesar = transaction.createObject(ValidatedCat.class);
8
9  ValidatedCat cat1 = transaction.createObject(ValidatedCat.class);
10 cesar.addCatFriend(cat1);
11 cat1.addCatFriend(cesar); // Necessary symmetric edge
12
13 ValidatedCat cat2 = transaction.createObject(ValidatedCat.class);
14 cesar.addCatFriend(cat2);
15 cat2.addCatFriend(cesar); // Necessary symmetric edge
16
17 ValidatedCat cat3 = transaction.createObject(ValidatedCat.class);
18 cesar.addCatFriend(cat3);
19 cat3.addCatFriend(cesar); // Necessary symmetric edge
20
21 // Adding another, fourth cat would invalidate the MaxCardinality on cesar's cat
       friends
22 // ValidatedCat cat4 = transaction.createObject(ValidatedCat.class);
23 // cesar.addCatFriend(cat4);
24 // cat4.addCatFriend(cesar);
25
26 transaction.commit();
```

Listing 24: Exemplary "`ValidatedTransaction`", Using the "`ValidatedCat`" Anno4j Class Displayed in Listing 23.

The code shown in Listing 23 passes the validation check, as the appropriate additional relationships are added. In this case, the symmetric edges directed from "`cat1`", "`cat2`", and "`cat3`" respectively back to "`cesar`" are defined in the lines 11, 15, and 19. Lines 21 through 24 show a commented piece of code which could invalidate the max cardinality requirement on the defined relationship, as the allowed number is exceeded. Listing 25 shows an exemplary error message indicating the violation of this requirement. The error message shows, which instance is invalid (the RDF node with URI "`http://pao.com/cesar`") and also the maximum cardinality is exceeded with 4 representatives, while only 3 would be allowed. The "`ValidatedTransaction`" object is created in line 3 (with the respective Anno4j object), started in line 5 and committed in line 26, just like a classic transaction.

```
1  com.github.anno4j.ValidatedTransaction$ValidationFailedException: Property http://
       petsandowners.org/hasCatFriend of http://pao.com/cesar has only 4 values, but
        maximum cardinality is 3
```

Listing 25: Showcase Error Message if the Allowed Maximum Cardinality of the "`pao:hasCatFriend`" Relationship for "`cesar`" is Exceeded.

4.6 AUTOMATED DOMAIN MODEL GENERATION THROUGH ANNO4J RDF SCHEMA PARSING

Various features of the Anno4j library have now been discussed, on the one hand concerning the "basic" functionality of the mapping, and on the other hand also more specialised features like the ValidatedTransaction and the Schema Annotations in order to support consistency as well as validity for the underlying database status. Essential building blocks to these processes are the Anno4j Classes (the basic POJOs that enable the mapping), Partial Classes (extensions to the Anno4j Classes, allowing for more fine-grained and detailed implementation of the mapping objects), and Proxy Classes, which have been mentioned only briefly in Section 4.3 yet. To put it simply, a Proxy Class resembles a pre-compiled "building plan" that originates from the defined metadata model. From this plan the Anno4j library is able to create proxies for the respective Anno4j Class instances quickly, as it does not have to generate the proxies behaviour from scratch as described in Section 4.3. These Proxy Classes can then be provided for a given Java process to make use of. This does only work with a static model, as a change in the model definition does require a new generation of the Proxy Classes. However, the time savings achieved increase the applicability of the library, model changes should not happen to often, and the library does support an automated generation of the Proxy Classes as will be described in this section.

However, a problem can potentially exist when working with these building blocks, as especially with complex ontologies or use cases with many different classes and entities to model, the routine of creating the mapping (implementing the classes etc.) by hand can become cumbersome and most notably also error-prone. Although the mere class structure to implement is not difficult, this process needs a high amount of focus, as for example the sole assignment of an incorrect IRI to an Anno4j Class can lead to a faulty metadata model, recreating the mistake in the database status every time that respective class is applied.

To counteract this circumstance, a contribution of this thesis exists in the implementation of the so-called **Anno4j Generation Tool** for the Anno4j library. This tool is able to facilitate the process described above of creating a metadata model mapping, which is described in Section 4.6.1. Next to this, the generation tool is also the core instrument for automatically creating an object-oriented REST API for a respectively created model, see Section 4.6.3. The internals of the generation process are described in Section 4.6.2. The Generation Tool constitutes this thesis's answer to research question 3.1, while its Proxy-extension answers question 3.2 and the REST API-extension question 3.4.

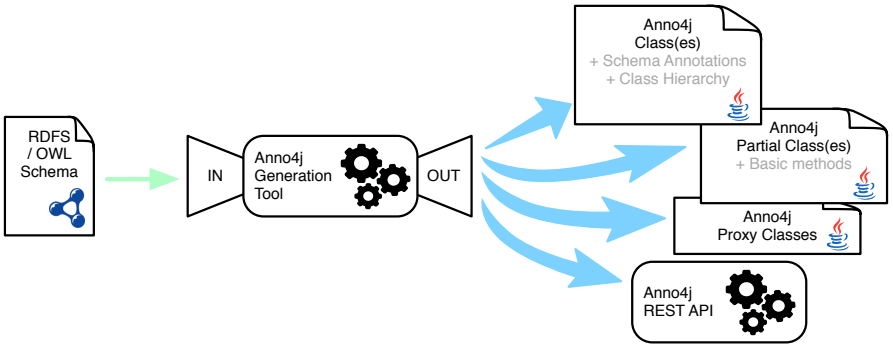


Figure 32: Illustration of the Anno4j Generation Tool. With the Input of an RDFS or OWL Schema File, Anno4j Classes, Partial Classes, Proxy Classes, and a REST API Can be Automatically Generated.

4.6.1 *Domain Model Generation Functionality*

The functionality of the Anno4j Generation Tool is illustrated in Figure 32. The single input for the tool (next to some optional configuration for fine-tuning the generation process) is a normal RDFS or OWL schema file, which can contain class and property definitions of the respective language, as well as the validation criteria that have been discussed in Section 4.5.3 (additional OWL criteria can be included, but will not be considered by the generation tool). At this point it is important to note that only RDFS and OWL Lite ontologies can be processed by the Anno4j Generation Tool. The reasons why will be discussed in Section 4.6.2.

Schema files posed a promising possibility to facilitate the metadata modelling process, as ontologies or vocabularies are mostly supported when a Semantic Web use case is implemented. Out of this schema file the generation tool is able to generate the following entities for the metadata model, next to the REST API extension explained in Section 4.6.3:

- A full list of Anno4j Classes representing the RDF classes that are included in the respective schema file. These are enhanced with appropriate IRIs for the class itself as well as all added relationships and properties that are defined. These are extended with possible Schema Annotations and basic methods (add/remove single/multiple instances), depending on the type of relationship or property and its multiplicity. Super- and subclass relations are reflected in a Java class hierarchy.

- For every basic Anno4j Class, a respective Partial Class is implemented, filling the base methods defined in the Anno4j Class with respective code. This enables a rich basic behaviour for every Anno4j object, and supports an interface for own extensions.

- Proxy Classes can be generated for the defined snapshot of the metadata model. Together, these can be included into the class path of a given Java process as a `".jar"` archive/file, skipping the behaviour gathering step of the metadata creation, resulting in a speed up of the application. Results of this increase in speed are evaluated in Chapter 6.

The configuration of the generation tool allows to handle two things: ambiguity and language. Especially in more complex schemata with a deeper class hierarchy it is possible to have clashes in the naming of Java classes as well as Java methods and their labels. The naming for both classes and methods is determined by supported `"rdfs:label"` properties, and if those are not present, the last fragment of the given IRI. Therefore the tool offers two ambiguity checks that can be enabled for the generation process. This results in a slightly longer processing time, but will prevent name clashes that would have to be dealt with by hand. As an example, consider the two IRIs `"pao:PetOwner"` and `"pao:Pet_Owner"`. As the underscore character "_" in the second IRI is not a valid character for Java classes, the tool would ignore it and both entities would be mapped to the same Anno4j Class `"PetOwner"`. With a checked ambiguity, both Anno4j Class names would be extended with a random number that does not carry any semantics, but the clash is prevented and therefore both (potentially semantically different) Anno4j Classes would be created.

In terms of language, two things are concerned in the generation process: firstly the naming for bot Anno4j Classes and their methods, and secondly the description of those in the form of Javadocs. For both a preferred language as well as a default language can be issued via configuration of the Generation Tool. Therefore the properties `"rdfs:label"` and `"rdfs:comment"` of the respective schema will be utilised, as both can be supported repeatedly with different language codes.

An extensive example for an application of the Anno4j Generation Tool with the utilisation of the PAO ontology is shown in Appendix A. The basic application of the generation tool is shown code-wise in Listing 31. Using the exemplary input schema illustrated in Listing 32 (which is oriented to the data model that is designed and used in Section 2.2), the output of the generation tool can be seen in Listing 33.

At this point it is important to mention a small tweak to the way Anno4j database objects have been created up to now. The Java constructor does also support a boolean flag that induces the respectively created Anno4j instance to also persist schema information that is associated with a given Java project at that time. This is especially important with a generated schema and the ValidatedTransactions, as both work on that stored schema information.

The technical process of generating a domain model out of a given ontology requires several procedures or algorithms in order to coun-

teract issues that arise from the different concepts that exist between RDF and OWL on the one side, and Java on the other side. These issues in combination with solutions that have been introduced to the Anno4j library will be covered in the following section.

4.6.2 *Generation Process Internals and Algorithms*

The overall process of the generation requires an RDFS or OWL Lite ontology as input. The first step consists of extracting required information in order to generate a domain model for the Anno4j library, which is done by applying a reasoning step to the ontology (reasoning in RDF has been explained shortly in Section 2.2.6). This reasoner uses several rules to deduce said information and make it usable in further processes, creating a **materialised ontology**. For the Anno4j implementation, the reasoner Openllet[32] is used, which is built on top of the reasoner Pellet [158].

With this materialised ontology, a procedure will be conducted that normalises the present RDF or OWL classes in order to streamline the RDF class concept to be used for the same purpose in a Java domain model. The above mentioned issue between both concepts exists in the interpretation of classes and their instances. While the Semantic Web concept allows the definition of equivalences between classes or properties, which symbolises that their instances are present for both classes or that they receive both classes as a type, this is not possible in Java. With this issue in mind, it also gets apparent, why only OWL Lite and not higher specifications can be utilised in order to generate Java domain models. Java does only allow inheritance relations between classes or interfaces to be modelled in a way that represents a subset-relationship in RDFS or OWL. This allows to be mapped only to those OWL/RDFS specifications that apply the "`rdfs:subClassOf`", "`owl:equivalentClass`", and "`owl:intersectionOf`" relationships at most, which can only be assumed from OWL Lite specifications, but not higher representatives. Nevertheless, a survey of Wang, Parsia, and Hendler [172], who have analysed 1275 RDF documents for their application of ontology specifications, shows that the largest part of these documents use OWL Lite or lower for their documents, therefore justifying the generation workflow in practice as described.

To solve the above issue, a procedure will be proposed that combines equivalent classes represented by the given RDFS or OWL Lite ontology, eventually making it possible to turn them into Java classes. The result of this step is called **normalised ontology**. This ontology is semantically the same with the original ontology, but meets the requirements defined above.

32 https://github.com/Galigator/openllet (last visited 03/12/2018)

Further steps that will be done on the normalised ontology increase the usability as well as the validity and applicability of the generated domain model. As it is common sense not just in the Semantic Web, but also other programming related topics, meaningful names should be used for the classes, properties, and relationships in the ontologies as well as the domain models. If this information is present in the input ontology of the Anno4j Generation Tool, procedures will be called that determine meaningful names for the combined classes of the normalising step, as well as produced identifiers for the created Anno4j Classes and their methods. Furthermore, the method signatures in the created domain model will be enhanced to eventually use the most specialised types for their methods.

NORMALISING A MATERIALISED ONTOLOGY - FINDING EQUIVALENT CLASSES The normalisation step implemented in the Anno4j Generation Tool applies its functionality in two steps. The issue between the RDF and Java class concepts manifests itself in two available relationships used in RDF/OWL in order to implement class hierarchies and equivalences, which need to be handled for Java conformance: the "`rdfs:subClassOf`" and the "`owl:equivalentClass`" relationships. The former in most cases would not be harmful, as it can directly be mapped to a Java inheritance. Nevertheless, cyclic implementations of the relationship are allowed in RDF, eventually creating a class equivalence between classes that are part of the given cycle. The latter creates a class equivalence between two classes that are associated with a respectively defined relationship. Cyclic inheritance, and thereby class equivalence, is prohibited in Java and hence these occurrences need to be eliminated.

The OWL relationships can easily be found by simple SPARQL queries, but the respective RDFS correspondent proves to be more difficult to find because of its cyclic nature. Therefore the first step of the normalisation consists of finding and "removing" existing cycles.

As has been seen throughout this work, RDF data and documents can be interpreted as a directed graph with nodes and directed edges. With this concept in mind, cycles between nodes can be found by addressing them as so-called Strongly Connected Components SCC, in which every node is reachable by every other node contained in the currently considered component. Detecting these in a graph with SPARQL queries would be inefficient, therefore the Tarjan Algorithm [162] is applied. Algorithm 1 shows its implementation[33]:

The procedure seeks all SCCs in a given graph $G = (V, E)$ that are reachable by a node v via depth-first search. In order to identify all present SCCs, the procedure can be repeated as long as there is a node $v \in V$, whose value $v.index$ is not yet defined. Next to the graph

33 Aligned to the implementation shown in https://www.programming-algorithms.net/article/44220/Tarjan's-algorithm (last visited 03/12/2018)

Algorithm 1 : The Procedure "`tarjan()`" to Identify Strongly Connected Components in a Given RDF Graph.

Data : Directed graph $G = (V, E)$, considered node $v \in V$, already identified SCCs $S \in \mathcal{P}(V)$, stack K, current index i
Result : Set $S \in \mathcal{P}(V)$ of the Strongly Connected Components in G

```
v.index := v.lowlink := i
i := i + 1
K.push(v)

// Considered neighbouring nodes of v:
foreach (v, w) ∈ E do
    if w.index is not set then
        // Continue recursively with w:
        tarjan(G, w)
        v.lowlink := min(v.lowlink, w.lowlink)
    else if w ∈ K then
        v.lowlink := min(v.lowlink, w.index)

if v.lowlink = v.index then
    // The nodes on the stack form an SCC:
    SCC := {}
    do
        u := K.pop()
        SCC := s ∪ {u}
    while v ≠ u;
    S := S ∪ {SCC}
return S
```

itself and the starting node, the input for the procedure is given by a new index i to be set for a node and a stack K.

With $v.lowlink$ the lowest index of a node reachable from v is saved, whose SCC is not yet identified. As v is not traversed yet, there is no other known edge to another node and therefore v is the only reachable and known node, and $v.lowlink$ is set initially to the index of v. That followed, all direct neighbours w of v are considered. If the node has not been traversed already, the search starts the procedure recursively starting with w. In the case a node with lower index is reachable from w, $v.lowlink$ has to be adjusted when the depth-first search returns.

If w in the other case has been traversed and hence is present on the stack, this means that w is not part of another SCC, as these nodes will be removed from the stack in the next step. Here $v.lowlink$ has to be adjusted if a node with lower index is reachable from w. The test if a node is contained in the stack can be realised in $\mathcal{O}(1)$ by adding

a flag that represents the presence for example, which will be set for the addition or removal of the node respectively.

The SCC of the node v can be returned after all neighbouring nodes have been traversed and if v then has the lowest index of all reachable nodes. As every node is reachable from every node in an SCC and v has been visited first, there cannot be any further untraversed nodes in the SCC with the return of the depth-first search from v. In addition to this, all nodes of the SCC are placed "above" v in the stack, as these have been traversed after v and therefore have been placed on the stack to a later point of time. Hence all nodes can be popped from the stack until v itself is popped. The thereby identified SCC S can be added to the already found set of SCCs G.

With the above mentioned linear-time solution of determining the presence of a given node on a given stack, the procedure shown in Algorithm 1 requires only a linear runtime of $\mathcal{O}(|V| + |E|)$. Therefore it can be used in practice in order to determine the cycles constituted by the "`rdfs:subClassOf`" relationships in an RDF graph.

NORMALISING A MATERIALISED ONTOLOGY - THE MERGING OF NAMES OF EQUIVALENT CLASSES AND RELATIONSHIPS OR PROPERTIES The Tarjan procedure shown above solves the problematic cyclic relationships of the "`rdfs:subClassOf`" relationships, which implement a class equivalence between the classes of the cycle. Therefore the next step is constituted by treating the "`owl:equivalentClass`" relationships that directly support a class equivalence. This is done by the procedure presented in Algorithm 2, which is also used to fulfil the last step of normalising a materialised ontology that is needed for the overall generation process: the merging of class names of equivalent classes.

As input, the procedure shown in Algorithm 2 receives the RDF triples of the materialised ontology specification and the SCCs that have been identified by the Tarjan algorithm shown in Algorithm 1. At first, every class c out of the SCCs is checked for existing "`owl:equivalentClass`" relationships towards another class c'. If present, the instances of c and c' represent the same instances. Let s' be the name of the SCC containing c', then both sets of SCCs can be merged together, as every contained classes of both are equivalent to each other. This means that s and s' can be removed from S, while a combined SCC $\{s \cup s'\}$ can be added. When this has been done for every class in all SCCs, then all maximal sets of equivalent classes have been identified, both in terms of RDFS and OWL class equivalence.

The next step of the procedure chooses a class name c_s for each set of equivalent classes s, which is chosen to represent the respective set in the resulting normalised ontology specification. Therefore an iteration is done over all classes c in s and all occurrences of the URI of c in triples are exchanged for the IRI c_s. Hence all statements that have

Algorithm 2 : The Procedure "`normaliseClassEquivalence()`" to Merge "`owl:equivalentClass`" Relationships and Address New Names for Sets of Equivalent Classes.

Data : RDF-Triples G of a fully materialised ontology specification, SCCs S of the class generalisation hierarchy in G
Result : RDF-Triples G of the respectively normalised ontology specification

foreach *SCC* $s \in S$ **do**
  foreach *Class name* $c \in s$ **do**
    // If an equivalence of c to another class c' has been specified:
    if $(c, \texttt{owl:equivalentClass}, c') \in G$ **then**
      // Combine the SCCs of c and c':
      $s' :=$ SCC in S, which contains c'
      Remove s and s' from S
      $S := S \cup \{s \cup s'\}$

foreach *SCC* $s \in S$ **do**
  // Choose a class name that represents all the class names of the group:
  $c_s :=$ random class name in s
  // Exchange equivalent class names in all triples:
  foreach $c \in s \backslash \{c_s\}$ **do**
    Exchange c with c_s in all triples of G

been issued for a class in s do also apply for the classes now defined by c_s. At this time, the representative name is chosen randomly, but there is the possibility of adding functionality that devises a meaningful name out of the given class names. It should however be paid caution to not increase the procedure's overall runtime.

In terms of runtimes for the procedure shown in Algorithm 2, it has been mentioned that the "`owl:equivalentClass`" relationships can be queried efficiently, as it is the same for triples that respectively contain a given IRI. Therefore the procedure can also be done in linear runtime.

At this point it should be mentioned that the presented algorithms above are also able to do the same normalising for relationships and properties, which are implemented via the relationships "`rdfs:subPropertyOf`" and "`owl:equivalentProperty`" respectively.

DETERMINATION OF JAVA IDENTIFIERS With the normalised ontology created with the processes and procedures documented above, it is possible to turn a materialised ontology specification into a nor-

malised ontology specification that is convertible to a Java domain model from the conceptual point of view between RDF and Java. However, there are still further issues that need to be clarified in order to support a clean transformation.

As has been seen throughout this work, the Semantic Web utilises IRIs to "name" their resources. These cannot directly be used in Java, especially for identifiers (for the interfaces of the Anno4j Classes as well as their method names), as they contain letters and symbols that are forbidden (e.g. brackets). Therefore in the overall workflow, a step is necessary that adapts respective IRIs that will be used in the generation process. At the same time, when choosing the names for Java interfaces, meaningful and significant names should be selected in order to convey convenient semantic understanding for the eventual developer that makes use of the resulting domain model.

The first approach to this is the "`rdfs:label`" property that is commonly defined in Semantic Web ontologies. This label in general contains a textual name that is mainly for humans to interpret and understand. These labels therefore are often composed of letters only and implement words that bear semantical meaning for humans. As described above, if Java prohibited symbols are contained, these are deleted from the label. In addition, the often used Camel-Case syntax is applied for further understandability.

If the "`rdfs:label`" is not defined for a given class or property, it is attempted to achieve the desired information out of the given IRIs of the currently considered resource. In many cases, the suffixes of such IRIs are also chosen to contain semantical meaning in order to eventually comply to Semantic Web best-practises, trying to make the use of the respective resource as easy and self-explanatory as possible. Therefore, the same rules for Java identifiers as described above - removing illegal symbols and use Camel-Case - are applied for the suffix of the IRI, which is then used as the identifier in the resulting generation step.

As the result of the generation is an in Java usable domain model for the Anno4j library, a further detail of convenience is drawn from the path of a given IRI, as these paths eventually can be transferred into a package structure in Java. For example, the IRI "`http://petsandowners.org/Cat`", which describes a feline animal and which has been used in examples throughout Chapter 2, will be mapped to a Java class with the identifier "`Cat`" in the package structure "`org.petsandowners`".

DETERMINATION OF METHOD SIGNATURES The last few passages have described, how the Anno4j Generation Tool devises meaningful names and identifiers for the eventually generated interfaces of Anno4j Classes as well as their methods. For the latter however, only

the identifiers are not enough, as Java methods for a full signature also require return types and parameters.

This information is also present in the normalised ontology and can therefore be used for the generation process. The "rdfs:range" and "rdfs:domain" relationships have been introduced in Chapter 2, which restrict both the source and the sink of a to be defined relationship or property respectively. This information can directly be mapped to a Java method signature. The "rdfs:domain" determines which resulting Anno4j Class will receive the currently considered method, while the "rdfs:range" fixes respective return types and parameters. As an example, if the "pao:isPetOf" relationship from Listing 3 with domain "pao:Animal" and range "pao:Human" would be used in the generation process, the resulting methods would be added to the "Animal" Anno4j Class with two basic methods (the automatically generated Partial Classes would add more methods, which are skipped here for reasons of clarity). The signatures for these methods would be "void setOwner(Human owner);" and "Human getOwner();", each with the added "@IRI(http://petsandowners.org/isPetOf)" Java annotation to properly map the defined methods.

The Anno4j Generation Tool does also incorporate the RDFS requirements of allowing multiple values to be set for the above relationships in its workflow. Here the issue exists again between the concept of RDF and Java, as the assignment of multiple "rdfs:range" relationships semantically conveys the intersection of all defined RDF classes. This intersection of classes however is not necessarily represented by an own class which could be used in respectively generated Java interfaces. Therefore the generation process needs to determine a class C, which has an own class name in the given ontology specification and which contains all individuals of the classes contained in the intersection. This is the case when C represents a generalisation of all those intersecting classes. Given that the RDF class "rdfs:Resource" is represents a generalisation of all available classes, it can be inferred that such a class can always be found. The "rdfs:Resource" is represented by the "ResourceObject" Anno4j Class in the library, hence the same assumption can be done from the Java point of view. Nevertheless, for a smaller error rate as well as higher usability of the generated domain model, it is desired to find a class that is more specialised than the top concept of "rdfs:Resource". The procedure shown in Algorithm 3 shows how this is done in the Anno4j library.

The procedure shown in Algorithm 3 receives a set of RDFS or OWL classes $C_1, C_2, \ldots, C_n$ as input, which have been defined as the values of a "rdfs:range" or "rdfs:domain" relationship. At first it is tested, if a class exists in the given ontology specification that equivalently represents the intersection of the classes $C_1, C_2, \ldots, C_n$ already. In OWL, this can be tested by considering the respective

Algorithm 3 : The Procedure "`getLowestCommonSuperclass()`" to Find the Lowest Common Superclass Among a Set of Classes.

Data : RDFS/OWL-Classes $C_1, C_2, \ldots, C_n$
Result : Most special common generalisation C of $C_1, C_2, \ldots, C_n$

// Check if an equivalent class exists in the cut of $C_1, C_2, \ldots, C_n$:
if *If there exists a class* C *with* $C \equiv C_1 \sqcap C_2 \sqcap \cdots \sqcap C_n$ **then**
  return C

// Identify a set S of common generalisations:
foreach $C_i \in \{C_1, C_2, \ldots, C_n\}$ **do**
  // Remove all classes from S that are not a direct or indirect generalisation of C_i:
  $S_i :=$ Set of all generalisation of C_i
$S := \bigcap_{i=1}^{n} S_i$

// Identify the most special class in S:
$C := \arg\max_{s \in S}(d(s, \texttt{rdfs:Resource}))$
return C

"`owl:equivalentClass`" relationships, which are present in the applied materialised ontology, as they are produced by the Openllt Reasoner discussed above. Therefore, the required information can be requested via SPARQL queries.

In case there is no equivalent class present, a preferably most specialised class has to be determined, which contains all the classes $C_1, C_2, \ldots, C_n$. To achieve this, the generalisations S_i of every class C_i is determined. Given that the ontology is fully materialised, S_i can be located efficiently and fully as a set of all "`rdfs:subClassOf`" relationships of C_i. Afterwards with the intersection of all S_i, the set S of all common generalisations of $C_1, C_2, \ldots, C_n$ can be found.

The last processing step $C := \arg\max_{s \in S}(d(s, \texttt{rdfs:Resource}))$ now determines a class C in the set S, which is as specialised as possible. This is the case, when C is located as "deep" as possible in the graph of generalisations and therefore it has preferably the most generalisations. Hence, for the determination of C, the distance for every class s in S is calculated with $d(s, \texttt{rdfs:Resource})$. This distance represents the path of "`rdfs:subClassOf`" relationships from s to the most common class "`rdfs:Resource`". With the preceding step of reasoning, this path does always exist, as "`rdfs:Resource`" is a generalisation of every other class and therefore this path is inferred for every class. Also, the length of this distance path is finite, since the generalisation hierarchy from the normalised ontology specification is free of cycles after the procedures shown in prior steps.

With the choosing of a class that has maximal distance to the "`rdfs:Resource`" class, a class C is determined, which is as specialised as

possible and contains the intersection of the classes $C_1, C_2, \ldots, C_n$. It is to be noted that if $n = 1$ then the class C_1 itself is returned.

PERFORMANCE OF THE GENERATION PROCESS The performance of the generation workflow that is applied by the Anno4j library to generate a domain model out of a given ontology specification is evaluated in a combined experiment described throughout Section 6.2. These evaluations apply a mocked specification that is used as input for the Anno4j Generation Tool to create a domain model, from which instances of generated Anno4j Classes are then instantiated. The runtimes for this whole process are tracked and it is shown that runtimes can be kept to an imperceptible minimum. Therefore an application of the generation can be done in practice for running systems, however the actual usage of it is designed to constitute a pre-step before an application goes online.

ROUNDUP OF THE GENERATION WORKFLOW The procedures of this section have shown how it is possible to generate a fully usable Java domain model from an RDFS or OWL ontology specification. This constitutes the centrepiece of the Anno4j Generation Tool. It has also been shown that linear runtimes are held throughout the generation workflow, which allows for an efficient utilisation and incorporation of it in real-world applications. Whenever possible, focus is laid on understandability in order to make the resulting domain model as useable and self-describing as possible, which does also align with the Semantic Web best-practises.

All of this shows that it is an effective and useful way to make use of generation processes in order to transfer models from one concept to another, which alleviates the access to these models as well as their accompanying technologies. This is done by allowing developers to use their familiar technologies, also eventually reducing the development effort of applications. Because of this, the next section will show an extension to the processes described above, which lifts the Java domain model even further to be used as a REST API via commonly known HTTP access methods.

4.6.3 *Generation of a Web Component for a Metadata Model*

Next to the generation of the technical elements necessary for the mapping of a metadata model described above, the generation tool of the Anno4j library is also able to produce RESTful interfaces. These interfaces align with the model that is created in the first generation step and therefore enable the library to be used as a REST API. This moves or transfers its "place of installation", which is illustrated in Figure 33.

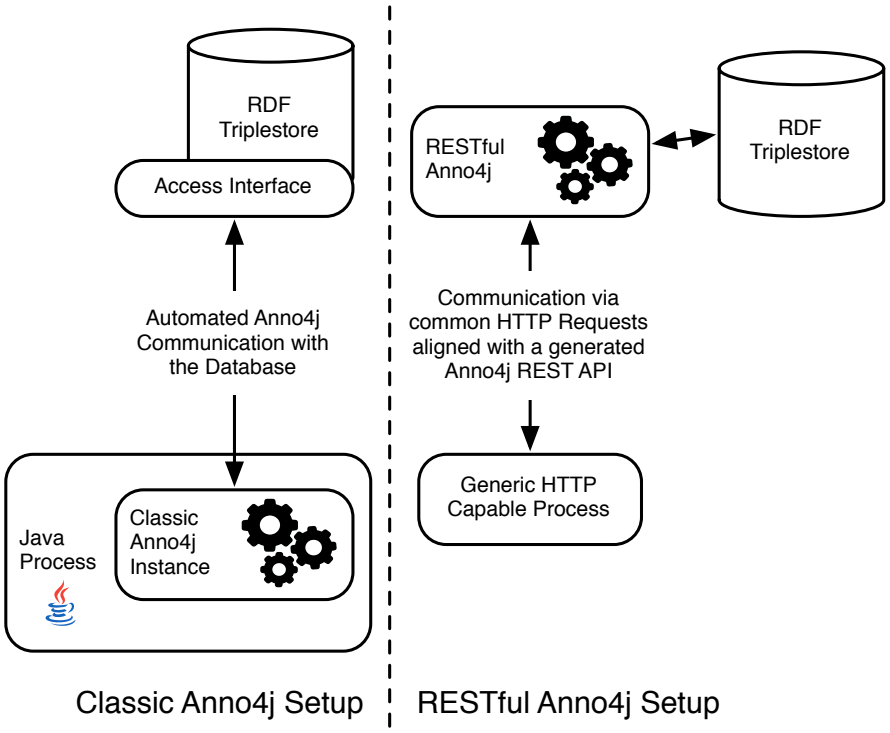


Figure 33: Places of Installation for the Anno4j Library: Classic and RESTful.

The left side of Figure 33 shows the "classic" implementation of the Anno4j library. Its functionality is embedded into a common Java process that is run on a given computer. The respective triplestore can be had locally or remotely and is connected to the Anno4j instance. Communication is done automatically via SPARQL queries that are created and processed by Anno4j.

The right side shows the RESTful application of the library. The instantiation of a respective API interface can be embedded into the generation process described in Section 4.6.1, which creates the metadata model out of a supported schema file and also creates the Web interfaces necessary. Additionally, the creation of the Web interfaces can also be triggered by hand, to incorporate Anno4j Classes that may have been created by hand after a preceding generation process (or created via generation without generating the Web component as well). Eventually, a **Web Application Archive** (a .war file) is created that contains the respective RESTful interfaces. This can be embedded next to a triplestore on a server for example, whose access interface URIs (query and update) have to be supported via configuration to the RESTful Anno4j component. This allows the triplestore to be accessed via the RESTful API that is generated by the Anno4j library via common HTTP calls, extending the accessibility of the database as shown in Figure 25.

Implementation-wise, the RESTful application of the library follows the basic Web stack of the Spring framework[34], featuring a modular controller/service/repository structure. This enables a fine-tuned implementation that can be adapted conveniently for various use cases. Throughout all of the layers, a classic behaviour is implemented for every Anno4j Class, supporting basic methods like *create*, *query* (single/all), and *delete* (single/all) instances of given input.

In summary, a convenience feature is supported by the Anno4j Generation Tool, lowering the barrier to access to the mapping between RDF data and Java objects even more. A Semantic Web expert maintaining an RDF database does in most cases also provide a schema written in RDFS or OWL next to it. This schema as sole input can be applied to support access to the database via Java objects in a convenient and automated way. From another point of view, implementing a new ontology or vocabulary can be done via Anno4j Classes or designing a schema which is then used in the generation tool. In the end, in both ways, these schema information can also be applied to foster data validity through Schema Annotations. All of this functionality can also be encapsulated into a RESTful application of the Anno4j library, which can conveniently be generated with either a schema and/or a collection of Anno4j Classes as input.

4.7 ADDITIONAL ANNO4J DATABASE FEATURES

The preceding sections of Chapter 4 have given a detailed description of the Anno4j library. It has been covered, what the library is able to do and in what fields it can have beneficial factors for the development of and in applications. This sections will show smaller additions to some of the aforementioned functionalities, which partly originate from overall RDF features that have been covered in Section 2.2 and/or increase the comprehensiveness of said features.

NAMED GRAPHS AND TRIPLE CONTEXTS Named Graphs are an RDF feature that allow the contextualisation of triples inside the same database by supporting the named graph IRI next to the commonly known components of the triple. This concept has been introduced in Section 2.2.4 and is accessed via the library in different situations. Next to the possibility of creating an Anno4j object with a defined default context which will be passed to all its functionality automatically, objects created via an Anno4j instance can have the named graph assigned with a respective method parameter. A instantiated QueryService can be created in regards to a given context, and transactions (both Transaction and ValidatedTransaction objects) can have their context set to an IRI of a named graph. Listing 26 shows a code excerpt showing the various possibilities of inserting the context.

34 https://spring.io/ (last visited 03/12/2018)

```
// The IRI used for the namespace at different occasions
URI context = new URIImpl("http://www.petsandowners.org/graph1");

// Create an Anno4j instance with default context
Anno4j anno4j = new Anno4j(context);

// Create a Cat object in the named graph, which would not be necessary if the
     Anno4j object is already contextualised
Cat cat = anno4j.createObject(Cat.class, context);

// Create a QueryService that will query in the defined named graph only
QueryService qs = anno4j.createQueryService(context);

// Create a Transaction object and set the context
Transaction transaction = anno4j.createTransaction();
transaction.setAllContexts(uri);
```

Listing 26: The Applications of Named Graphs/Context at Anno4j Instance-, Object Creation-, QueryService-, and Transaction-Level in Anno4j.

SERIALISATION INPUT AND OUTPUT Another prominent RDF feature and an important part of the Semantic Web technologies is the materialisation of triples to and from the serialisation formats that have been discussed in Section 2.2.5. Triples can therefore be written in textual form for humans to read or a parseable format for computers to understand. Supporting a serialisation for the objects that are used in the mapping via the Anno4j library opens up additional possibilities of using the overall implementation by allowing convenient usage via the commonly known triple representations.

This functionality extends the mapping of the Anno4j library with a further component, as users can **parse** triples in order to convert them to Java objects (that need to be represented in the data model of the respective Java and Anno4j process), or on the other hand **write** their existing Java objects to the commonly known RDF serialisations. Figure 34 illustrates the extended mapping concept.

The visualised mapping shows the richness of interoperability of the Anno4j library. The supported implementation opens a wide variety for users to interact and apply Semantic Web technologies in order to eventually create Semantic data. With the different conversion mechanisms, data can be converted from one file format to another, allowing users to benefit from the whole chain of data.

From an implementation point of view the writing of triples is straightforward. Every object created via an Anno4j instance has a "`.getTriples(RDFFormat format)`" method, which takes a constant value of the RDFFormat Java class - representing an RDF serialisation format - and returns the respective object as a String containing the serialised triples.

The parsing requires the creation of an "`ObjectParser`" object.This parser is contained in the Anno4j library and based on the Apache

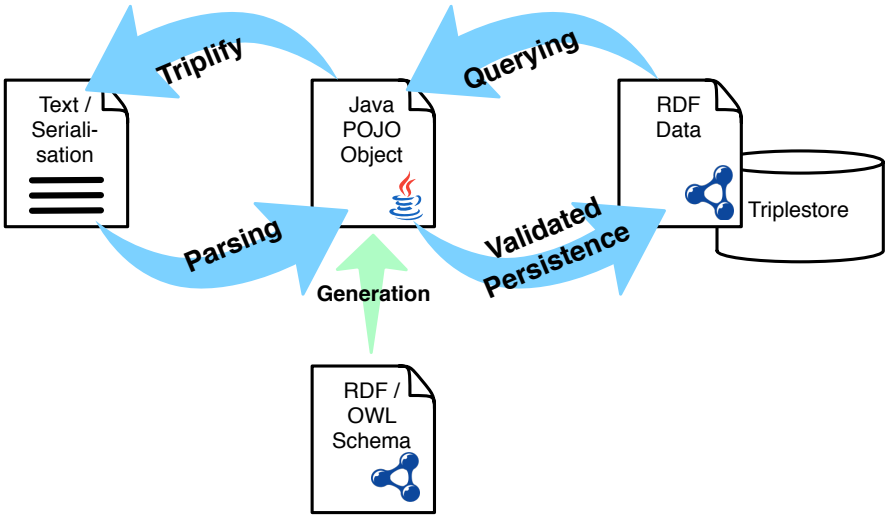


Figure 34: Illustration of the Extended Overall Anno4j Mapping Functionality, Including the Parsing of Serialised RDF Triples and Writing of Anno4j Java POJOs to Serialised RDF.

Jena RDF I/O technology (RIOT)[35]. The ObjectParser's "`.parse(...)`" method has got parameters for the input text String, a generic class or Set of classes that determines the Anno4j Class type to parse for, a default context, and the RDFFormat representing the serialisation format that the input file is written in. The result is a list of objects that are of the defined Anno4j Class(es) type and which are contained in the triples of the parsed file. These objects are then kept in a local memory store of the ObjectParser (to not accidentally "pollute" the user's data status by default), can be used as normal Anno4j objects, and can be persisted to a respective database on demand.

QUERY PLUGINS FOR LDPATH FUNCTIONS For an even more comprehensive querying capability, plugins can be added to the Anno4j library. These extend the possible LDPath querying tools, by adding an additional LDPath function[36] in combination with an associated querying logic. This function can then be used in querying criteria provided for a given QueryService, allowing for much shorter but at the same time more complex querying features and expressions. By doing so it is possible to confine requested result sets beforehand, as the querying logic is evaluated at a given database, rather than returning a much larger result set which then is normally elaborated afterwards. In order to implement a plugin, the user has to define the LDPath textual function expression, as well as the querying logic. An

35 https://jena.apache.org/documentation/io/ (last visited 03/12/2018)
36 http://marmotta.apache.org/ldpath/language.html (last visited 03/12/2018)

exemplary expression (adopted from SPARQL-MM [104]) can be seen in Listing 27.

```
1 QueryService qs = anno4j.createQueryService();
2
3 qs.addCriteria("sparqlmm:leftBesides(\"elephant\", \"lion\")");
4
5 List<ItemMMM> result = qs.execute(ItemMMM.class);
```

Listing 27: Exemplary Plugin Expression "`sparqlmm:leftBesides(...)`" in an LDPath Criteria.

The integration and application of such a plugin could lead to a criteria that is much shorter, but with the same semantics. The example shown in Listing 27 could potentially query for those "`ItemMMM`" objects that contain two animal detection results, one showing an elephant, the other showing a lion. In addition to this, the coordinates of the found results are supposed to be in a specific order, as in this case the elephant to the left side of the lion.

4.8 ANNO4J CONCLUSION, OUTLOOK, AND ENVISIONED ADDITIONS

In summary, the Java library Anno4j offers comprehensive access to RDF databases, following the commonly known idiom of an object relational mapping. This broadens the database access for developers to not only use classic Semantic Web technologies like SPARQL to directly query the database for retrieval or persistence purposes, but use an idiomatic language they are more familiar with: Java. Still, "lower" and more "complicated" access interfaces are not overshadowed for experts but rather open in parallel, enabling a user base as diverse and wide-ranging as possible.

To recuperate the library from a technical point of view and therefore show its technological benefits, Figure 35 shows the Semantic Web Technology Stack[37] and the implications of Anno4j on it, or more precisely in what layers the user is supported when applying it. The Semantic Web Stack is a collection of concepts that are utilised in the overall Semantic Web, with the addition of candidates in the form of specifications or standardisations that implement respective concepts. Concepts that do not have a candidate are envisioned in the overall Semantic Web construct, but have not yet been supported technologically.

The yellow border indicated at the bottom of Figure 35 shows which concepts are supported by the utilisation of Anno4j. With the descriptions of the library throughout this chapter it has been seen that Anno4j implements an abstraction for RDF data, therefore helping with the overall concept as well as its serialisation or syntax and

37 https://www.w3.org/2004/Talks/0412-RDF-functions/slide4-0.html (last visited 03/12/2018)

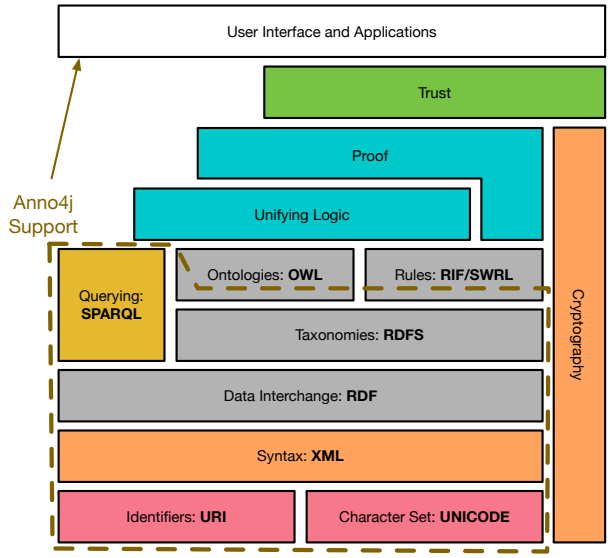


Figure 35: Visualisation of the Semantic Web Technology Stack with Concepts shown in Regular Font, While the Implementing Technologies Are Written in Bold Font. The Dotted Border Shows Which Concepts are Affected, Interwoven, and / or Supported by the Anno4j Library.

also respective identifiers and URIs (which can be freely chosen by the expert, or automatically be generated for the newcomer). Taxonomical features that are supported by the RDFS schemata is fully representable in Anno4j domain models, while querying is facilitated by different querying methods. Out of the OWL specification only the concepts similar to the RDFS features are implemented, with the addition of the validation possibilities given by OWL Lite (see Section 4.5.2). Additional rules or logics are not yet implemented, but an SWRL extension is envisioned, which is described below. In the end, Anno4j supports the implementation of applications, which is indicated in the illustration as well.

Next to the various technical features supported by the library, these applications do also benefit from design principles that are implemented by Anno4j. The foremost among these is the abstraction that is implemented, allowing users to access and produce RDF data without having to deal with the Semantic Web technologies to do so. Next to this, the library is designed in a modular way, decoupling its single features whenever possible to support fine-tuned utilisation for the user. On top of this, extensibility is offered both in ways of persisting and querying data, allowing for further user-driven extensions and implementations. The validation features of the Anno4j library support some kind of robustness to the application, as given input can be validated against existing schema information. As it has also been seen in the related approaches, development effort gets reduced when applying an ORdfM implementation like Anno4j, and Chap-

ter 6 will shows that an application is also not impeded runtime-wise by the library. Next to these general design benefits, the ORdfM concept does also enable general advantages, namely productivity, non-intrusiveness, reusability, and higher maintainability by lower error-proneness.

In Section 4.1 the core requirements by Bauer and King [16] for an ORM implementation were listed. These naturally can be adapted for ORdfM implementations as well. Anno4j supports all of them, namely CRUD API, querying language or API, mapping metadata, and optimisation functionality. In the following, the technical features of Anno4j will be recapped and it will therefore be shown, how Anno4j fulfils the basic requirements for ORMs.

The creation of metadata is encapsulated in a mapping, allowing the user to work with Java classes with very easy basic behaviour which - through the mapping - get transformed to RDF data on the fly. However, this is not restricted to this basic behaviour, as so-called Partial Classes can be supported next to the Anno4j Classes whose methods can be implemented by hand and therefore incorporate Java features and therefore a richer functionality.

Basic querying in Anno4j is supported idiomatically via Java objects, while LDPath expressions enable a thorough and rich querying functionality that lowers the barrier of the de-facto standard SPARQL to query metadata from an RDF database. Rather than querying pattern-based, LDPath is a path-based querying language that supposedly is easier to use, especially for non-Semantic Web experts. The result of an issued query via Anno4j is returned as the defined Java objects of the Anno4j mapping, which are therefore likewise convenient to use for developers. The `"QueryService"` interface of the Anno4j library implements a fluent interface, allowing to add LDPath querying criteria as well as different configurations conveniently and in modular fashion, enabling extensive querying functionalities.

Next to the mapping functionality to persist and query metadata, the Anno4j library supports data consistency- and validity-preserving features, that can be used and applied optionally. This is done via transactional behaviour, which can be extended by so called Schema Annotations. These Annotations are aligned with OWL Lite validation requirements for RDF data like cardinality and transitivity, and can be added to the defined Anno4j Classes for the mapping. When said transaction would be committed to a database, a check for the integrity of the database status is done before changes are committed. Eventual errors or validity violations are propagated to the user with a description about the incorrect action.

Code generation features further increase the applicability of the library in various other use cases. The Anno4j Generation Tool allows the parsing of existing RDFS or OWL schemata to generate the mapping in an automated fashion, bypassing the process of creating it by

hand. This is especially useful and time-preserving for more complex ontologies, while also avoiding the error-proneness of hand-written code. Next to this, the Generation Tool can also alleviate the mapping functionality to a RESTful API, enabling access through common HTTP communication.

Additional features permit the whole Anno4j functionality to be applied in common RDF named graph fashion, while a parsing component allows to read and write Anno4j objects from and to textual RDF serialisations. Already comprehensive querying functionality supported by LDPath can further be extended by personal plugins with LDPath functions.

All of this technical functionality to access semantic data next to an easy inclusion of the library via Maven facilitates its applicability in not just a simpler way of using a mapping. The scenarios described above enable whole workflow chains of large-scale applications to be supported by the library, enhancing and simplifying the steps not just for creation and consumption of the metadata, but also the exploitation as it can be used and produced much more conveniently. With the additional features like generation, the effort for developers to include metadata into their work cycle gets lowered immensely. As an application example to also evaluate these claims, Chapter 5 will describe the MICO platform, a workflow-based multimedia scenario in which Anno4j has been used to implement the metadata communication. In this scenario, efforts have also been made to provide the Anno4j functionality to other programming languages. A proof-of-concept version for C++ is implemented in the MICO platform, called **Anno4cpp** (see [5] [22]) which constitutes a proxy to the Anno4j library using Java Native Interface[38] [110]. This also reveals, that all the presented features are not just bound to Java only.

Next to this, Anno4j is provided in the active community on Github[39], has been included and mentioned in recent scientific activities [161] [130], and is listed as one of the reference implementations for the Web Annotation Data Model on the W3C homepage for the Web Annotations Group[40]. Furthermore, the currently ongoing research project ViSIT poses another promising field of application, and therefore room for further development of the library (see Section 6.3). Also, the uptake of the library in a research project can be seen as positive evaluation.

More ideas exist that further elevate the idiomatic functionality of the library and therefore lower the barrier of Semantic Web technologies even more. These ideas originate from current use cases or have been gathered during the current implementation stage, but have not (yet) been implemented as they would exceed the margin of this work.

38 https://docs.oracle.com/javase/8/docs/technotes/guides/jni/ (last visited 03/12/2018)

39 https://github.com/anno4j/anno4j (last visited 03/12/2018)

40 https://www.w3.org/annotation/wiki/Implementations (last visited 03/12/2018)

In any case, a short description of those ideas is given here in order to illustrate the expendability and especially the potential of the applied approach of Anno4j.

OBJECT QUERIES The most anticipated extension is summarised under the name **Object Queries**. As described in Section 4.4, the current status of the Anno4j library supports queries by supporting LDPath expressions. This is considerably lower in complexity than SPARQL queries, but still requires some kind of knowledge of Semantic Web technologies. Like the name suggests, Object Queries will be related to the kinds of objects that are already in use of the library. As these are again implemented in Java, the same claim of idiomatic utilisation of Anno4j holds for this feature as well. Listing 28 illustrates how this could look like.

```
1  // Left out code dealing with the Anno4j instance as well as creating potential
       other Anno4j objects, in particular eventually the Cat Cesar
2  Human bob = anno4j.createObject(Human.class);
3
4  QueryService qs = anno4j.createQueryService();
5
6  Cat queryCat = qs.createObject(Cat.class);
7  queryCat.setAge(5);
8  queryCat.setOwner(bob);
9
10 Cat result = qs.execute(queryCat);
```

Listing 28: Exemplary Object Query Application Scenario.

The known QueryService interface gets extended so that it can create Anno4j objects just like an Anno4j instance, and take objects as input that follow the defined mappings that are used for querying and persistence up to now. The execution call then transforms the object into respective queries. In the example, a query for a `"pao:Cat"` object of the age `"5"` and the owner `"bob"` would be issued. The expressiveness of this method needs to be evaluated if it can represent all querying criteria that SPARQL or LDPath have, but if this could be done successfully, the querying barrier would be reduced to a Java-conform level requiring only a little knowledge about Semantic Web technologies.

SWRL The **Semantic Web Rule Language SWRL** [86] [124] is a language that allows the definition of Horn-like rules that can be applied to a semantic dataset in order to trigger a functionality that acts like a reasoner. Reasoning has been described in Section 2.2.6 and is used to create additional metadata from already existing metadata by applying those rules. These are made up of an *antecedent* (a set of requirements that need to be met in order to trigger the consequent) and a *consequent* (a set of actions that are executed if the antecedent is fulfilled). Both of these components are built as a concatenation between core elements, like the existence check of a relationship between two

RDF entities or an RDF type check for one entity. SWRL is based on a recombination of OWL DL and OWL Lite with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language[41] [36]. It therefore enables an expressive language that allows the definition of rich rulesets. Syntaxes are supported for XML as well as RDF, and there is also a more human-readable notation supported by the authors. Listing 29 shows an example of a rule that could be implemented for the PAO use case. The rule infers semantically that if a person (represented by the variable "`?person`") owns a pet ("`?pet`") and then marries someone ("`spouse`"), then the respective spouse should also have the sponsorship to said pet associated.

```
pao:isPetOf(?pet, ?person) AND pao:isMarriedTo(?person, ?spouse)
    => pao:isPetOf(?pet, ?spouse)
```

Listing 29: Exemplary SWRL Rule for the PAO Ontology.

Beneficial factors of SWRL towards a technical application are its expressiveness as well as the easy to read syntax of a ruleset. This allows for a convenient implementation in the Anno4j library, enabling users to support defined rules that can be triggered as a conjoined process. These processes can then be used to infer information about their dataset, which does not need to be modelled necessarily via the Anno4j Classes. This opens up several new opportunities of metadata modelling. One envisioned application is called **Semantic Zoom**, which will be used in the Visit project (see Section 6.3). This zoom will work with two different metadata models that describe the same use case in two variants - one complicated and detailed, the other one more easy and convenient - automatically converting information content that is produced for one model to the other model and vice versa.

RECOMMENDATION In use cases that allow users to search through a supported set of data, for example video or music platforms at which users can search for their favourite films and music, oftentimes it is desired to have the possibility to compare these multimedia data in order to support some kind of statement like "You liked this film, so maybe you would also like these other ones!". This is generally done with recommendation engines, extending the given use case to a broader functionality as well as delivering a better user experience overall. Especially the MICO use case showed potential for such a recommendation component, and therefore the possibility to do an implementation was evaluated. Similar extraction results could be recommended, so for example two pictures with an alike animal detection result could be presented to the user for research purposes. LDPath querying criteria can be applied in the recommendation fea-

41 http://ruleml.org/index.html (last visited 03/12/2018)

ture calculation and therefore target all RDF facets of MMM Items and Part Annotations.

From a data model point of view, the RDF feature called reification has been introduced in Section 2.2.6, which is well suited to constitute recommendation results. A beneficial factor here is that the reification statement is not interpreted as 100% true, which is in the nature of recommendations, but it is rather a statement about a statement and only asserted to be true as described before. Next to this, recommendation processes are always in need of hardware resources, as well as good potential of being supported technically to not delegate more effort to the user than necessary. All of this aligns perfectly with an implementation feature for the Anno4j library, which has been done prototypically as an extractor in combination with a mapping model in MICO. Figure 36 shows an example how the model could look like.

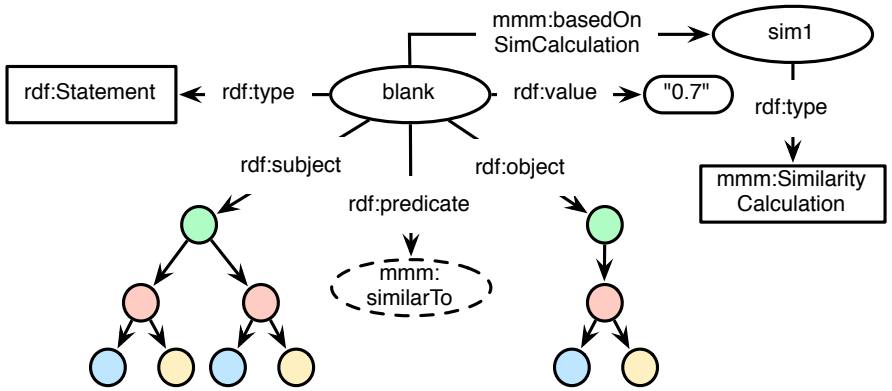


Figure 36: Exemplary Anno4j Recommendation Result in the MICO Use Case. The Two "`mmm:Item`" Instances at the Bottom (Indicated by the Coloured Tree-like Structures Which Correspond to the MMM Colours Discussed in Section 3.3) are Compared and Evaluated to Have a Similarity in Their Results to 70%.

Two items have been evaluated for their similarity based on the "`mmm:SimilarityCalculation`" with the IRI "`sim1`". An "`rdf:value`" is supported to indicate the degree of similarity between the two Item instances on the bottom of the picture, in this case "`0.7`". These are connected by an "`rdf:Statement`" as described in Section 2.2.6. With this implementation, multiple different calculation strategies can target the same RDF instance, enabling a broad field of application. The approach for the implementation can then support federated and comprehensive querying capabilities so that the recommendation results can be used effectively and without further effort.

5

A WORKFLOW-DRIVEN APPROACH FOR MULTIMEDIA METADATA APPLICATION

The preceding sections of this work have given insights into basic Semantic Web Technologies, overall metadata modelling with a focus on multimedia, and the Anno4j library, which is an implementation of an ORdfM to implement an abstraction layer on top of a given RDF database to conveniently work with semantic data in idiomatic fashion by using Java. The latter of these insights laid special focus on the fact that existing Semantic Web Technologies have an initial barrier to learn and apply, and that implementations like Anno4j exist in order to facilitate miscellaneous interactions with the database, enabling non-Semantic Web experts to produce and consume semantic data as well.

However, these implementations can not only be used to facilitate single processes or enable non-experts, but they can also contribute to a more comprehensive and thorough implementation of "bigger" applications that intend to take part in the Semantic Web. To evaluate and show this claim, the information supported in the previous sections will be concentrated and applied as a use case in the MICO Platform, a centralised software platform that is designed to analyse multimedia data by applying workflow-based processes via connected autonomously working multimedia extractors. As a result, the platform produces descriptive information that constitutes a metadata background for the analysed multimedia files. The vision or idea of the overall MICO Project as well as an initial description of the MICO Platform has been given in Section 1.2.

The MICO Metadata Model (see Section 3.3) will serve as the ontology for the created metadata, while Anno4j will be included mainly in every metadata extractor to implement the metadata communication with the central MICO Platform. Therefore, metadata at the extractors can be persisted by creating and filling Anno4j Java objects (which correspond to the model mapping defined in the platform dependency, see Section 5.2) rather than using so-called "SPARQL templates", predefined SPARQL queries with placeholders at respective positions, which were the preliminary solution for creating, querying, and communicating metadata. These templates proved to be error-prone as well as time-consuming, as next to basic templates every extractor needed templates of its own. Hence Anno4j supported a much more reliable and maintainable solution to the task of metadata management. As will be seen later, Anno4j can also be applied in other features of the MICO Platform.

For the MICO scenario described above two solutions will be presented, differing in the point of time and therefore the status of the Anno4j library, which will have impact on the possibilities of implementation space. The first solution will describe the technology present at the effective date of the end of the MICO Project. At that point, the MMM is in its final version described in Section 3.3, the Anno4j library was released with the version 2.3.0[1], which incorporated the functionality of the 2.0.0 version with an increment to the metadata model of the WADM. This version does *not* include the features about generation (see Section 4.6), validation (see Section 4.5.2, although basic transactional behaviour is available), nor any solution about the proxy problem (see Section 4.6). However, a fully functional MICO Platform implementation exists for the scenario with the facts described above.

The second setting does incorporate the Anno4j library in its latest release 2.4.0[2] and can therefore exploit its full potential. This scenario is not implemented, as a thorough implementation would exceed the margin of this work, however the respective results and given evaluations support the claim from a theoretical and partly practical standpoint. This theoretical construct is subsumed under the name **MICO+** and will be described in combination with its potential advantages in Section 5.4.

Regardless of which of the two settings is considered, it is shown that a part of the problems of the Semantic Web shown in Chapter 1 can be solved and improved. More precisely, the described problem of having a task that is supposed to be solved using the Semantic Web requires the gathering of information from various different information peers. Therefore, subproblems can be defined that firstly incorporate the understanding of the various information bits and pieces, as the peers do not necessarily communicate "speaking the same language", and secondly the recombination of all those information in order to receive a combined result. While the part about understanding supports another difficult task to accomplish, what the MICO use case in combination with Anno4j and the MMM shows is, that the communication between various information peers is possible and that their results or inputs can be combined to be commonly accepted and interpretable. Additionally, by supporting a workflow-oriented information concept of the platform that is enabled by a metadata model that can incorporate respective information, it is shown that the gathered information fragments are not only used statically, but can evolve and fuel other yet undiscovered semantics.

To give a deeper insight and explanation about this, this section is structured as follows. Section 5.1 will enlist related work in the fields

1 https://github.com/anno4j/anno4j/releases/tag/v2.3.0 (last visited 03/12/2018)

2 https://github.com/anno4j/anno4j/releases/tag/v2.4 (last visited 03/12/2018)

of multimedia platform implementations as well as workflow-centric metadata processes, in order to compare existing approaches against the work done in this thesis. Afterwards, Section 5.2 describes the MICO Platform more in-depth in the first setting describe above, especially highlighting the components and details that are necessary for this work to show its contribution. An adapted metadata workflow and lifecycle concept for for handling metadata information is described in Section 5.3. Existing shortcomings of the first described platform setting will then be evaluated in Section 5.4, and according improvements will be presented. These solutions will be concluded in a technical description and hence theoretical construct of an extended MICO Platform implementation, called MICO+.

5.1 RELATED WORK - MULTIMEDIA METADATA PLATFORMS AND METADATA LIFECYCLES

As the previous chapters have shown, metadata in general has a great impact on the applicability of data itself and especially in the fields of Multimedia, as the descriptive information allows to make better use of the multimedia content in terms of for example best display/payout or the actual content. Therefore, supporting a thorough metadata background allows the given multimedia item to be used in a wider area of application, as it gives useful information about how it is produced, how it is structured and built up technically, and especially what it is about or what the content of it is.

It has been learned up to now how metadata can be modelled and described altogether in the Semantic Web, an implementation of a specific multimedia metadata model, the MMM, has been shown, and technical implementations to create, access, query, and apply this metadata have been proposed. Following this, it is useful to highlight the different high-level stages that metadata goes through in its entire lifetime in order to potentially improve some of these steps to further increase the quality of the metadata. These steps are subsumed under the concept of a *lifecycle of metadata* by various authors. They also mention the importance of technical support in order to access many of these steps, as they are often interwoven into functionality and therefore hidden and unaccessible for users, which counteracts the idea of making space for improvement. Hence, after explaining some insights about the lifecycles of metadata, this section will describe platform implementations and their impact and/or adaption on the lifecycle of metadata.

In [102], Kosch et al. describe the lifecycle of metadata that is directly connected to multimedia. They do this, by separating the cycle into three different spaces: the *content space* contains the steps that directly concern the multimedia file itself, while the *metadata space* includes everything relating to metadata that is connected to the given

multimedia file of the content space. The third space illustrates the various kinds of users that interact in different potential ways with the multimedia file and/or with respective connected metadata. This is called the *user space*.

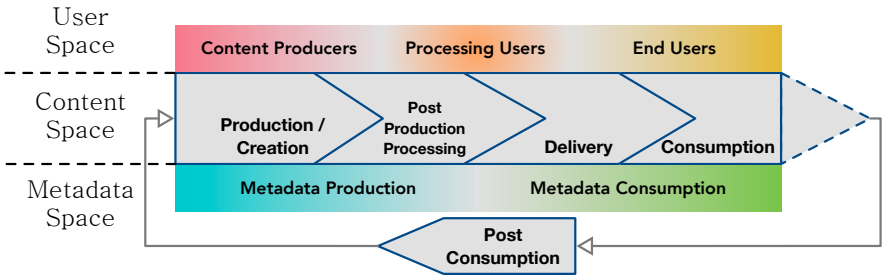


Figure 37: Visualisation of the Multimedia Metadata Lifecycle (Adopted from [102]).

Their content space encompasses the actual *creation* or *production* of the multimedia file, which is followed by the *Post Production Processing* step, enriching the created file with metadata either by automated analysis processes or human interaction. Eventually, the multimedia in combination with its descriptive metadata is then distributed in the *Delivery* step, enabling users to make use of the multimedia item and its metadata in the *Consumption* step.

The metadata space is coarsely divided into two subparts, *metadata production* and *metadata consumption*, dealing with the creation and utilisation of metadata associated in the multimedia workflow of the content space respectively. The production of metadata takes place at first during the production step of the multimedia file, when common technical information like the creator, the creation date, and the file format is addressed. Next to this, the post processing of the multimedia file oftentimes incorporates various basic analysis processes (like it has been seen in the MICO use case) that add low-level features in general, for example a color histogram or similar features from the MPEG-7 standardisation. There is also the possibility for human interaction, adding more high-level features to the metadata background that cannot be performed by the analysis processes automatically. These quite often contain information like transcoding hints, a reference to original files (if the given described file has been compressed in some way), a media profile, and possible content adaption infos for further processing.

Different user groups or roles are involved in the whole lifecycle of multimedia metadata, fulfilling and contributing to different steps of the cycle. *Content providers* are the essential starters of the lifecycle, as they create or instantiate the multimedia file that is to be processed. After this initialisation, the *processing users* are the ones who enrich the given media file with describing metadata in the post production processing step. Lastly, the *end users* induce the delivery of the media

files in order to consume them. This also encapsulates the metadata consumption step of the metadata space, as the end users filter and browse through content, find appropriate versions of media files (depending on their QoS criteria), and therefore trigger the payout of the media files via the internet by using and applying the assigned metadata.

The lifecycle of multimedia metadata in their vision ranges from the creation of the initial media file, to metadata allocation, and finally to delivery and consumption by end users. However, as the authors also hint, the lifecycle for a given file is not necessarily "over" here, as the participating roles of interacting users can overlap and therefore trigger another iteration of the cycle. In a *post consumption* step, end users can potentially allocate things like feedback, viewing preferences, device characteristics, or simply alter and adjust existing metadata in order to further improve its quality. These terms can then be seen as either a new multimedia file itself (if the respective changes to it are enough to interpret it like that) and therefore restart the cycle at the production/creation step, or the metadata allocation can be seen as post processing and therefore trigger the cycle from the second overall step. This eventually creates a reiterating circular workflow of different process steps that can be iterated for several times in order to create an always improving metadata background for the initially injected multimedia file as well as its potentially created altered versions via compression or similar processes.

As mentioned in the introduction of this section, the allocation of metadata does have a close connection to technical implementations, as these can enable better access to the steps of the metadata lifecycle, or even provide the access possibility in the first place. Kosch et al. do also support this claim in [102], as they, next to the steps of the cycle itself, introduce a project in combination with its technical foundation in order to embed the metadata lifecycle. The CODAC project[3] bases its metadata content on the well-known MPEG-7 standardisation (also explained in Section 3.1), which is stored in a Multimedia Database Management System [103]. This metadata database is complemented by a Multimedia Data Cartridge[4] [59] for multimedia files in combination with metadata, and a dedicated streaming server to play out the given multimedia files of their databases.

Authors of various other literature [11] [12] [13] [122] highlight another point of view on metadata, also in the interpretation of a lifecycle: the Lifecycle of Linked Data. Their lifecycle is designed in circular fashion, as there are eight steps that can reiterate and re-trigger each corresponding successor and even each other in a non-strictly linear fashion. They do also propose a technical foundation of the lifecycle,

3 http://www-itec.uni-klu.ac.at/~harald/codac/ (last visited 03/12/2018)
4 https://docs.oracle.com/cd/B28359_01/appdev.111/b28425/introduction.htm (last visited 03/12/2018)

called the LOD2 Stack. The eight steps implementing the lifecycle, which is also depicted in Figure 38, are namely:

EXTRACTION Data can be present in various different formats and/or persisted in databases. In order to incorporate it into the Linked Data lifecycle, the data has to be at least mapped to RDF data, or even converted at best.

STORAGE AND QUERYING With enough RDF data, the next step consists of storing the data, making it access-, manage-, and queryable.

AUTHORING This step encapsulates the possibilities for users to both create new data in conjunction to the existing RDF data, as well as correcting the already existing data.

(INTER-) LINKING Following the general idea of the Semantic Web of having increased knowledge benefits for every participating peer of interconnected RDF databases, this step induces users to link respective data of preceding steps to other knowledge bases.

CLASSIFICATION AND ENRICHMENT At this point, a lot of RDF data has been created and gathered, and Ngomo et al. [122] raise the claim that (starting) at this step, the quality of the produced data should be revised, in order to especially increase the querying capabilities and usefulness of the data altogether. In this step the focus therefore lies on the classification, structure, and schema information of the data. The mainly mentioned process to achieved this is relying on high-level structures that are allocated to the gathered RDF dataset.

QUALITY ANALYSIS By doing a comparison to the quality of content of the Web of Documents, the authors also issue that a quality assessment should also be done for Linked Data. Therefore, strategies or calculations are envisioned to do so.

EVOLUTION AND REPAIR If problems, inconsistencies, or faulty data is detected, this step needs to offer possibilities to correct those.

SEARCH, BROWSING, AND EXPLOITATION Eventually, the result of the lifecycle is to support an ideally qualitative rich, well structured, and extensive knowledge base of RDF data that will potentially be used by a broad field of users. Hence, searching, browsing, and exploitation possibilities must be supported in a convenient and comprehensive fashion.

The LOD2 Stack (a result of the LOD2 project[5]) is an integrated distribution of aligned tools and by that implements the Linked Data

5 The original project homepage is offline, a good summary of the project is found here: http://aksw.org/Projects/LOD2.html (last visited 03/12/2018)

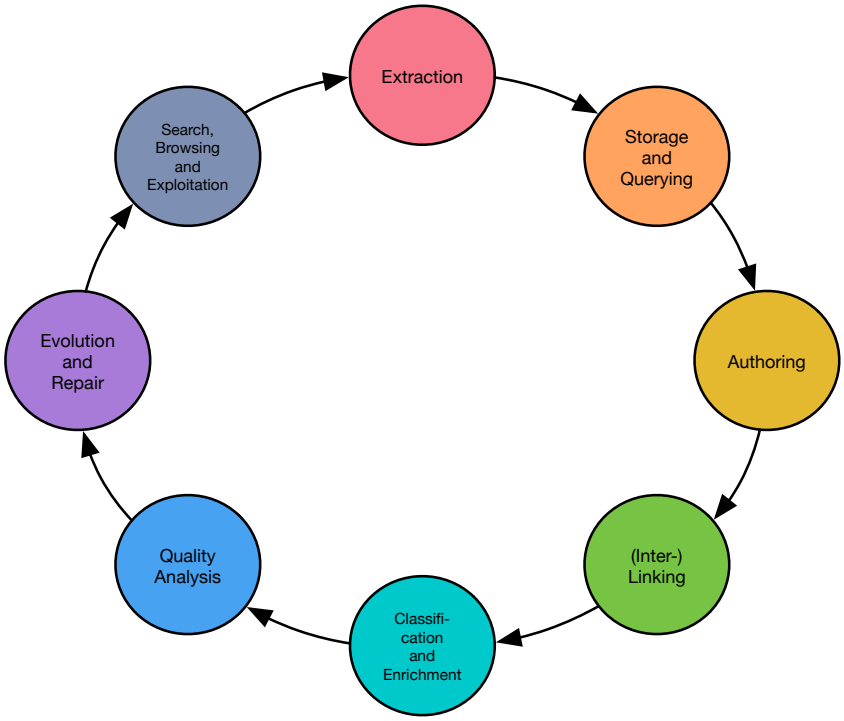


Figure 38: Visualisation of the Linked Data Lifecycle (Adopted from [11] [12] [13] [122]).

lifecycle. It enables to choose between the incorporation of tools implemented in the same project or third parties for every step of the lifecycle, in order to enable comprehensive access to every step and therefore the comprised RDF knowledge and data. Its main components are implemented as open-source to support a wide field of application as well as development and deployment possibilities. Next to this fact, the whole implementation is kept versatile, encapsulating most functionality via defined interfaces in order to support easy plugin of third-party implementations.

The whole framework therefore represents a combined accumulation of different software, working together to fulfil the described lifecycle. As both the third-party implementations as well as the own implementations are fine-tuned towards their specific use case or field of application, especially the adapted implementations are not altered in order to not compensate their usability. In fact, the LOD2 Stack offers its own access interfaces and allows the forwarding to respective third-party interfaces of the respective lifecycle steps.

With the different possibilities posed by the recombination of the application of different software pieces for every step of the lifecycle, every implementation of the LOD2 Stack is different. Hence, the LOD2 project supports a *stack configurer*, allowing users to apply the

characteristic development of the stack depending on their own requirements.

This is made possible with a modular architecture of the LOD2 Stack, basing on three core pillars:

- For a standardised, well-known, as well as versatile deployment and packaging process, the LOD2 implementation is based on the Debian packaging system. This enables easy assembly and installation of the various components. Dependencies can also be managed via the Debian system.
- The access to the underlying RDF database is supported via central standardised SPARQL endpoints that follow standardised vocabularies. The same approach is applied for the base access between the different components. All components of the LOD2 Stack are supposed to query and write back their data to the corresponding central triplestore, which allows for high interoperability between the components. This however emphasises the fact that every component, or more specifically the data that is written by the given component, should also be aligned strongly to standardised and commonly understood metadata standards and ontologies.
- As described above, the LOD2 Stack does not influence the original interface amongst the possibly very heterogenous adopted components, but outside of this, the Stack does implement REST enabled Web interfaces as far as possible in order to support commonly known access possibilities to the functionality of the underlying Stack implementation.

In summary, a versatile and modular framework is supported by the LOD2 Stack that is able to create Linked Data applications. By complying to the structure of the defined Linked Data Lifecycle, every of the eight steps can transparently be implemented either by own implementations or by third-party software, allowing to construct transparent applications that are fine-tuned to their respective use cases and needs.

Altogether, the two described employments of lifecycles show that it is beneficial for the whole technical application to have concise definitions of the interlinked steps and what the purpose of every step is. Therefore, every sub-process can be designed directly fit for its purpose, highlighting its advantages to enable easier and more comprehensive access for the given user.

While Kosch et al. [102] highlight their own designed implementation next to the lifecycle for multimedia metadata, Ngomo et al. [122] support a framework that can be self-designed for different purposes. Both approaches however show the importance and the connection

between the theoretical lifecycle and the underlying technical implementation that realises the lifecycle. Therefore having the lifecycle of metadata in mind when designing an application is important.

Because of this technical importance in the course of metadata, the remainder of this section will highlight approaches and solutions to this aspect. As it has been seen before, many different steps are necessary for a thorough lifecycle of metadata, and some of them even need user interaction in order to function. Amongst those, the most impactful steps of the lifecycle are especially the creation, extraction, and possibly the application of the metadata and hence will be focused. Different approaches in terms of semi- and full-automated platforms, often times with Web features, cover the landscape of todays metadata and annotation scenery.

Kavasidis et al. [97] propose a semi-automated Web-based collaborative platform that is designed to aid users to build large scale and diverse ground truth datasets that are supposed to be utilised for object detection, segmentation, tracking, and classification. Their vision is a resulting collection of annotated data that can be utilised as training data to increase the performance of classifiers in different fields of application. With a use case of underwater fish-videos, they provide proof of their concept in a very difficult domain, as animal monitoring is already difficult in general, and potentially harder in underwater domains [160]. The tool allows users to upload video files, which will then be segmented automatically into video frames. The user is then prompted to utilise pencil- and rectangle tools to easily place annotations on a given frame. Additionally, by presenting several preceding and/or successive frames, annotations on different frames can be copied and therefore be combined and interlinked in order to create associations and easily relate these annotations. Sharing mechanisms between users allow for even better ground truth creation, as annotations will be under concurrent supervision and therefore quality control is ensured.

With CAMO[6], Hu et al. [88] cover the problem that missing metadata for multimedia hampers its usability severely. They issue that there are various datasets describing multimedia available, especially in the wake of the LOD. However, these are oftentimes focusing only on single multimedia types, although a combination and interlinking would be very valuable, elevating the usefulness of the multimedia metadata and consequently the multimedia item itself. Next to this fact, they also claim that there are descriptions of a multimedia item present in different data sources, but frequently these are heterogenous in terms of metadata modelling as well as information coverage. Thirdly, as they focus their metadata descriptions on the RDF standard, useful "legacy data" stored in relational databases is challenging to interlink with their existing RDF knowledge base. CAMO

6 http://ws.nju.edu.cn/camo/ (last visited 03/12/2018)

solves these problems by applying a well-known ontology, the DBpedia ontology[7], as their base ontology, to which other information is to be mapped in terms of a mediation model. Additionally, with methods like ontology matching and instance linkage they can create aggregated knowledge bases with combined information about different media types as well as combined descriptions emerging from various (originally heterogenous) data sources. For the legacy data, Hu et al. implemented a procedure that is able to connect the relational data to their domain model. Lastly, they implemented an Android app that is able to query the CAMO database in order to make use of the created multimedia metadata.

The most similar solution to the MICO Platform approach is given by De Meester et al. [56], who, like the approach described in this work, address the issue that comprehensive retrieval of multimedia has become a necessity in the Web world of today. They also exemplify this by the uprising quantity of easy possibilities to create and share multimedia content, mainly emphasising this point with the convenience of Web access and the rising amount and quality standards of portable devices. Efficient retrieval goes hand in hand with the quality of the attached multimedia metadata, which is in the focus of the authors. Creating these metadata by hand is unfeasible, therefore De Meester et al. propose a domain- and problem agnostic framework that is able to apply automated analysis processes by currently available multimedia analysis methods that are incorporated as Web services. Once a service is registered at the central platform, it supports descriptive metadata about its analysis process, which can later on be used to apply the respective service in an analysis chain. This is automatically determined by the platform by making use of such descriptive metadata. The metadata is persisted as RDF and mainly contains an easy representation of the input and output that is supported by a given analysis service. This semantic description of the services can then be used to meet the semantic description of a request to the platform. The whole analysis implementation is based on a reasoning cycle (inspired by [128]) consisting of three steps, which are visualised in Figure 39.

The reasoning cycle consists of three phases, *observation*, *hypothesis*, and *prediction*, between which respective processes called *abduction*, *deduction*, and *induction* transpose between the phases. Observations can be seen as the results of analysis services on a given task, therefore posing the initial analysis step as well as intermediate ones. With every iteration of the cycle, potential new observations are added and combined to the up to now discovered results. With every variation of the observations, hypothesises can be applied to the current state of the observations. These are in general combined with knowledge about the analysis domain in order to find inconsistencies in the ob-

7 http://wiki.dbpedia.org/services-resources/ontology (last visited 03/12/2018)

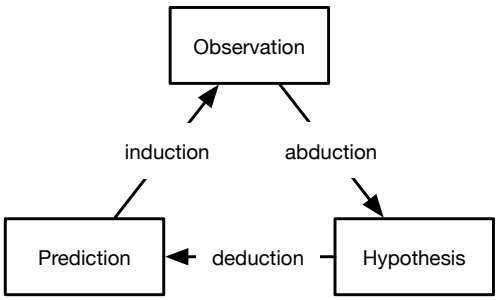


Figure 39: Visualisation of the Reasoning Cycle Applied in the Framework of [56] (From Which the Image Has Been Adopted).

servations. Then in the next step, predictions are created which reflect how the current set of observations can be improved, based on the given hypothesis of the iteration of the cycle. These predictions can then trigger respective analysis processes that are able to process the current input, which will in term create new observations that are combined with the previously created ones, hence closing the current run of cycle. It is important to note, that a given analysis service will never analyse the same part of a given multimedia item twice, so at some point, no further predictions with associated observations can be made, therefore ending the reasoning cycle for the respective input. The overall result is then the combined observation result.

In [56], De Meester et al. do also introduce a proof-of-concept as well as an evaluation that their approach is feasible and that their results achieve the promised results. In this proof-of-concept, the authors implemented several components of their approach in two different use cases: a face detection use case as well as an optical character recognition use case. Their results show that their framework can produce equally good if not better results as state-of-the-art methods and approaches, in addition to the fact that their framework can also incorporate these methods in order to further benefit from their input. For these components, further information about the semantic description of the analysis services and how these are constituted is given in [169] [170]. More information about the problem-solving platform is given in [168], which describes the component that is responsible for the allocation of tasks to the registered Web services. In so-called execution plans, the consecutive services are called, and respective results are gathered and combined. Furthermore, the implementation is able to find alternative routes if errors occur, in order to produce equivalent results, if possible. Technically, the platform itself implements a blackboard architectural pattern [50], which is composed by several components. The *blackboard* serves as an institution that collects and holds received information. *Services* are the working instances of the whole platform. A *supervisor* in combination

with a *service composer* triggers services accordingly to the requested processes and finds alternative problem solutions in case of errors.

LESSONS LEARNED AND SUMMARY - METADATA LIFECYCLES AND ITS IMPLICATIONS ON UNDERLYING TECHNOLOGY The pervasive focus point of the related work enlisted in this section is metadata, descriptive information pieces about a given file or data, with the emphasis lying on the Multimedia domain. In this domain, metadata bears an even more important role, as multimedia files are mainly useable to a good extent when descriptive metadata is present, which for example gives further information about the file's content, best playback modes, processing information, and so on. This metadata is expressed in different levels of detail and complexity, however, only a combination of several different levels leads to a thorough metadata background.

While the creation of such high-level features contains complexity to achieve on its own, the recombination of various produced metadata features poses an obstacle that needs to be addressed. Therefore, several related approaches have been mentioned that mainly focus on software platforms and frameworks that target this problematic. With this kind of software support, these approaches manage to create solutions to the metadata issue mentioned above that deliver results in a robust and especially more efficient, valid, and more accurate fashion. Next to this fact, the platforms and frameworks do also enable a higher and better degree of convenient usage of the production of results as well as the eventual usage of those respective results.

For the different steps that interact with the metadata like production, consumption, or exploitation, lifecycle interpretations exist that implement these steps in a more fine-grained fashion. Aligning one's application to these delimited steps supports better transparency and offers better fine-tuning of the sub-components as well. Therefore a modular structured implementation is possible, supporting a best possible fit to the respective use case.

Altogether, this Section 5.1 shows related work in the fields of multimedia applications and metadata lifecycles and therefore emphasises their respective impacts and eventually their importance. To integrate this into the course of this work, the following Section 5.2 will describe the MICO Platform from a technical point of view, which has been shortly introduced in Section 1.2. The MICO Platform constitutes an application that aligns with the related approaches and therefore, after its infrastructure and core components have been highlighted, Section 5.3 weaves the insights from this section and explains how the MICO Platform fulfils the presented requirements.

5.2 EMBEDDING THE MULTIMEDIA METADATA WORKFLOW INTO A TECHNICAL ENVIRONMENT - THE MICO PLATFORM

With the scientific background and the MICO Project explained in Section 1.2 and Section A.1 in the Appendix to form the theoretical basement for this work, this section will now give more insights into the technical implementation of the whole MICO infrastructure. This is necessary to understand both the implications of the results and contributions shown in the preceding sections as well as the beneficial outcomes of the overall platform. Therefore, after an overview of the general platform, specific components of the MICO infrastructure will be described in detail with focus on the topics covered in this work, namely metadata handling in the form of creation, querying, communication, and the application of the MMM, as well as the idiomatic application of Semantic Web technologies. Figure 40 shows an overview of the MICO infrastructure:

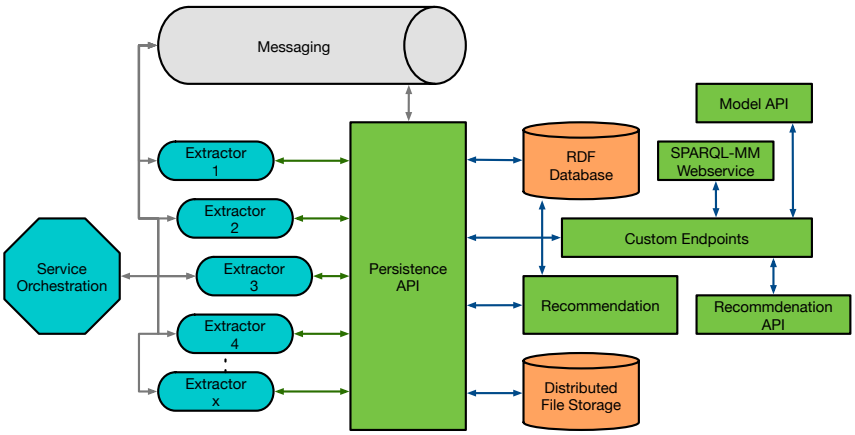


Figure 40: Setup of the MICO Platform.

The components of the MICO Platform in addition to their respective interaction with other components are the following:

MESSAGING The central piece of communication for the MICO infrastructure, managing all communication calls between components that can not be done directly (as for example calls to the database) and which are also possible to be queued. These are the messages that are sent between the core persistence API, the service orchestration unit, and the autonomously working extractors. Various different intertwined queues exist at the side of the MICO platform, and every extractor does have a queue of its own. The queuing system is implemented with RabbitMQ[8]. The queueing concept enables a fluent working environment, which does also alleviate some timing constraints in the overall

8 https://www.rabbitmq.com/ (last visited 03/12/2018)

infrastructure. Additionally, the concept does also allow different technical features. For example for the same extractor purpose or task, multiple instances of the same extractor type can be operated in parallel, as they listen to the same queue together. Every instance then can pull single tasks from that respective queue in a round robin like fashion.

EXTRACTORS The core "workers" of the MICO infrastructure, extractors receive ingested media files of the MICO Platform in order to produce results in the form of metadata or possibly new (sub-) media files. This information is then propagated back to the platform for potential further analysis processes. In general, an extractor implements one specific task and therefore defines a respective input and output format. Extractors can run on other computers than the platform itself and therefore work autonomously by only being connected to the platform via the messaging component. Further information and configuration of extractors will be given later in Section 5.2.2.

SERVICE ORCHESTRATION This unit manages the possible pipelines of extraction processes and therefore creates the workflow-driven analysis concept of the MICO Platform. When an extractor is registered at the platform, by supporting input and output formats, the orchestration unit can determine which ingested media files can be analysed by which extractor and/or extractor chains. This information is synchronised with the persistence API and respective calls to the autonomous extractors are issued. By doing this orchestration, different paths through the extractor processes can emerge for input multimedia files, and possibly unknown paths - the beneficial wiring and combination of extractors in a way that the developers had originally not planned - can originate through their recombination. The implementation of the orchestration unit is based on Apache Camel[9].

PERSISTENCE API This API supports the functionality for everything related to persistent data, and is therefore the main component connected to both the RDF and the file storage or database, next to the recommendation engine which has access to the RDF database, too. Hence this API implements methods that enable the creation, querying, updating, and accessing of any RDF or multimedia data in the MICO platform. The API is therefore called mainly by custom endpoints and the recommendation engine that will be described below, and indirectly by the extractors via the messaging component.

9 http://camel.apache.org/ (last visited 03/12/2018)

RDF DATABASE With respective modular implementation, any triple-store that implements the SPARQL 1.1 language can be applied here. The standard MICO Platform implementation comes with a supported Apache Marmotta[10] instance installed.

DISTRIBUTED FILE STORAGE For the persistence of multimedia items that are to be analysed by the MICO Platform as well as intermediate file results, an HDFS[11] implementation is supported in order to provide efficient access to these items.

CUSTOM ENDPOINTS Rather than seeing the MICO Platform from the perspective described in this work, the platform does also encourage "external" developers to take part in an existing installation of the platform in order to benefit from its produced results, as it supports various endpoints following classic REST and HTTP concepts. Some default APIs and Web services like the Model and Recommendation API or the SPARQL-MM Webservice are supported and therefore interact with the basic endpoints, and the modular structure of the MICO Platform allows for the addition of any custom endpoint extension.

MODEL API AND SPARQL-MM WEBSERVICE These access interfaces support the basic functionality of the standard ways to work with the MICO Platform results. These possibilities will be explained further in Section 5.2.1.

RECOMMENDATION AND RECOMMENDATION API A prototype recommendation engine based on Apache PredictionIO[12] [101] has been implemented for the MICO Project, supporting the functionality as well as an API to access the results. Recommendations will mainly be used in order to request "similar" results, given one result, so for example a video that is closely matched to another video based on the analysed metadata backgrounds of both. Although prototypic ideas and implementations have been incorporated for the MMM and Anno4j (see Section 4.8), the recommendation feature of the MICO Platform will not be described further, as it is not in the focus of this work.

The MICO Platform is distributed as free to use virtual machine image[13] in the Open Virtualisation Format (OVF) [93]. This image does come with a baseline of freely available extractors, namely "Audio Demux", "Speech-to-Text", "Kaldi2RDF", "Kaldi2TXT", "MediaInfo", "MediaTags2RDF", "Animal Detection" in two different variations (the first one using Deformable Part Models DPM [66], added with

10 http://marmotta.apache.org/ (last visited 03/12/2018)
11 http://camel.apache.org/ (last visited 03/12/2018)
12 https://predictionio.apache.org/ (last visited 03/12/2018)
13 https://www.mico-project.eu/pages/documentation/#document-7 (last visited 03/12/2018)

a second option using Deep Neural Networks and the YOLO framework [135]), "Face Detection", "ObjectDetection2RDF", "Diarization", "OpenNLP NER" with language models for English, German, Italian, and Spanish, and "OpenNLP Text Classifier". All the extractors can be enabled and disabled as required in the platform, and there are various predefined extractor pipelines that can also be activated. These can be loaded via the platform's basic interface by choosing respective pipelines and inducing the Platform to load them. An overview in the MICO Orchestration Service shows a graph of enabled extractors, also indicating the possible extractor chains that are constituted by matchings of extractors' inputs and outputs. Both the platform and the free to use extractors are implemented as open-source software under the Apache License Version 2.0[14] and can therefore be applied and extended freely. Software releases have been done frequently during the MICO Project, the final version of the platform is its version 3.0.4 (which is an upgrade from the downloadable version 3.0.1). A rich and detailed description of how to get the MICO Platform, how to install and initialise it, how to use it, and a row of internal details can be seen on the MICO Platform Overview homepage[15].

The following sections will highlight some of the details of the MICO Platform that are necessary as input for the workflow explanation of Section 5.3 and especially the concept of the MICO+ description in Section 5.4. Therefore, Section 5.2.1 will explain the different methods that the MICO Platform supports to both ingest and then afterwards access data. Then, Section 5.2.2 and Section 5.2.3 give insights about the applied extractor conceptualisation and how to implement own extractors, which is then concluded with the description of the MICO Orchestration Service approach in Section 5.2.4, which is the central component that manages the synchronisation of multiple extractors to eventually create chains of extractors.

5.2.1 *Accessing the Produced Results of the MICO Platform*

At the very start of every extraction workflow or metadata creation is the injection of a multimedia item into the MICO Platform, on which the overall analysis will be based on. These are represented as the "`mmm:Item`" instances that have been discussed in Section 3.3 and therefore represent the "root" of the whole respective metadata construct. To achieve this, the MICO Platform comes with a simple interface that supports the upload of any file. Next to the file, the user is prompted to define the file's format or file type. This is especially important, as this file type will be used by the MICO Orchestration Service in order to determine which extractors or extractor chains are able to process the injected media file. At this point, it is important to

14 http://www.apache.org/licenses/LICENSE-2.0 (last visited 03/12/2018)
15 https://www.mico-project.eu/pages/documentation/ (last visited 03/12/2018)

note that not only commonly known file types can be used here, but also own "self-made" semantic meta file types which can represent data constructs of one's own use case which then are eventually processed by own extractors that have been implemented for that given use case.

Once injected into the MICO Platform, each item will be represented by a respective URI, and it will be enlisted in the overview of items tab. There, all the items are enlisted and do have either of the states "In Progress", when the item is still processed by an extractor, "Finished", when the analysis process has finished successfully, or "Failed", in the case an error happened during the whole processing step. Figure 62 in the Appendix shows such an enlistment at the bottom, while also depicting a "Service Dependency Graph", which corresponds to the currently enabled extractors and their transitions between one another. Items whose analysis processes have failed do support further information by hovering the "Failed" indicator. An example can be seen in Figure 63 in the Appendix.

By this view, the basic inspection of an item is possible. A detailed view of the item can be accessed, which shows further information about the item itself like injection time, creator, as well as its type(s) and associated file. In addition, every `"mmm:Part"` that has been added by an extractor as intermediary or final result is enlisted here. These do also show technical details like the ones mentioned above for the intermediary steps. They also display their respective "source" of the analysis process, which has been explained semantically in Section 3.3.5. An example how this looks like in the MICO Platform is shown in Figure 64 in the Appendix.

From the item overview, as well as the detailed item view it is possible to access the item's metadata by pressing the buttons for "Inspect", "Metadata", or on an URI of either the item or a part, which automatically forwards the user to the triplestore's metadata view at the respective metadata access point. As already mentioned, for the MICO Platform the default triplestore is Apache Marmotta with its rich and diverse access and querying potentials. The basic metadata view allows the user to navigate through the RDF graph by clicking and following respective URI links between the RDF nodes, leading to an enlistment of the details of the currently viewed node.

Next to the basic view, the MICO Platform does support three of Apache Marmotta's querying interfaces to enable more comprehensive querying of the produced results. Those are the following:

SPARQL QUERY INTERFACE An interface that bases on the SPARQL 1.1 standard, using the standalone open-source project Squebi[16]. Next to full SPARQL 1.1 query and update functionalities, this interface does also support bookmarking of result pages, auto-

16 https://github.com/tkurz/squebi (last visited 03/12/2018)

creation of URI prefixes, and autocompletion for well known ontologies.

LDPATH QUERY INTERFACE LDPath has already been introduced in Section 4.4. This interface supports a comprehensive possibility to issue queries of the same specification towards the dataset that is currently present in the given MICO Platform instance.

SGVIZLER DATA VISUALISATION A visualisation interface is implemented that allows to visualise queried SPARQL results in various visual representations like charts, maps, or tables. To achieve this, the Marmotta framework incorporates the SGVizler framework[17], which is also supported conveniently by the basic MICO Platform installation.

Altogether the MICO Platform supports comprehensive and rich features to analyse, inspect, and make use of multimedia items in combination with their metadata background. This is already supported by the default setting of the platform, which then can be enhanced further with own functionality and extractors, leading to an even wider application scenario. Important for this adaption is the implementation of own extractors, for which a description is given in the following section.

5.2.2 *MICO Extractor Description*

The basic functionality of a MICO extractor encapsulates a certain task, in general a multimedia analysis task, that is executed autonomously for the MICO Platform. Apart from the registration process, the extractor does mainly listen to task jobs that are pushed to the extractor queue from the MICO Platform via the communication component described above. Results are communicated back in a similar fashion. To perform this, the extractor registers at a MICO Platform by supporting its input and output format, next to some configuration details. These have already been discussed in Section 3.3.5 and are illustrated in Figure 60 in the Appendix. The extractor by itself is not aware of being incorporated into a workflow chain of the MICO Platform, as this is done solely on the side of the platform by the Orchestration Service.

Next to communication and configuration, each extractor is supporting some kind of processing functionality, depending on the use case of the extractor. An extractor is triggered by being messaged by the platform. Therefore, every extractor holds its own messaging queue. The respective message contains information about the item that the extractor is to analyse. Results in the form of metadata information and potentially intermediary multimedia files are synchro-

17 https://github.com/mgskjaeveland/sgvizler (last visited 03/12/2018)

nised with the platform during the analysis process. Once this process is finished, the extractor sends a finished signal back to the platform, issuing that further analysis processes can be triggered or that the item has finalised its complete potential overall analysis.

As mentioned earlier, the functionality of creating and querying metadata as well as its communication among the extractors changed between three variants. The most basic and rudimentary implementation was supported by SPARQL Templates, predefined SPARQL queries that contain wildcards which could be filled accordingly. The templates were used for both querying and persisting at extractor-level and were "hosted" and supported via the platform dependency. However, this kind of approach had some shortcomings. The solution was primarily error-prone, as every single template had to be written by hand and especially the URIs used in the Semantic Web are sometimes intricate and drawn-out, therefore further impeding the definition of the templates. In addition, every different use case or extractor combination needed its own set of templates to query specifics out of the data storage. This setup required some extended knowledge about Semantic Web Technologies of the extractor developers or increased support of the metadata team, both leading to increased required manpower altogether.

One of the most relevant contributions of this work consists in an improvement of this shortcoming of the SPARQL templates, which is done by the Anno4j library, described extensively throughout Chapter 4. Using Anno4j, the effort of writing RDF data for the developers can be reduced to filling and connecting simple Java objects that are supported by the library. The data model to perform this was covered in Section 3.3. In terms of querying, Anno4j enables two approaches (see Section 4.4): the basic querying can be done by solely supporting the Anno4j Class to query for, filtering through all returned objects by hand and/or by code, which lets the extractor developers reduce the need for Semantic Web knowledge to a minimum. With a little more effort, the advanced querying of Anno4j via LDPath can also be used to fine-tune the respective query to only request the desired result objects.

The functionality described above is based on the Anno4j features of its version 2.0. For an extended application in the MICO universe, Section 5.4 will show how the functionality can conceptually be improved when incorporating the extensions supported by the 2.4 version of the Anno4j library.

5.2.3 *Implementing Own MICO Extractors*

To implement an own extractor, the MICO Project supports interfaces and guidelines on how and what to do on the MICO Extractor Bit-

bucket homepage[18]. Both Java and C++ are supported as programming languages, following slightly different procedures of creating an own extractor. As Java is mainly relevant for this work, it will focus on those corresponding explanations.

In order to support a convenient solution for integrating own extractors, a Maven parent can be incorporated in the Java project that is to implement one's own MICO extractor. Doing so, all respective dependencies and build plugins will be managed automatically. Via the platform dependency that is included in the Maven parent[19], there are also the necessary objects present for the metadata modelling, creation, and querying, and most importantly the interface for implementing an extractor. The "`eu.mico.platform.event.api.AnalysisServiceBase`" interface (with two further implementations "`AnalysisService`" and "`AnalysisServiceAnno4j`" with an automatically incorporated Anno4j transaction for RDF communication) supports five configuration Strings and a "`call`"-method that need to be implemented for a fully functioning MICO extractor. The former define the extractor ID, the extractor mode ID, its version, as well as textual representations of the format for input and output. The call-method is triggered whenever the respective extractor receives an event from the MICO Platform in order to analyse the included Item (or more specifically the Item that is referenced by a provided Item URI).

The MMM supports a modular structure that is used to facilitate the persistence and querying of produced metadata during a MICO workflow. The necessary components from Item to Part with its body and target implementations are always well defined and applied, no matter how the analysis process looks like in detail. This way, developers can always follow the same paths in the RDF graph in order to retrieve desired information. Various convenience functionality methods of the Anno4j library increase this unobstructed functioning even further. For example, rather than having the user deal with supporting correctly formatted Date Strings, a helping method implemented in the respective Anno4j Partial Class can have single parameters for every necessary Date parameter (year, month, day, etc.) and with those create the correct Date String.

However, there exists a minor shortcoming in terms of metadata exchange when it comes to introducing "new" metadata into a workflow system, as potential follow-up extractors need to have knowledge about respectively created metadata and its structure. This can be broken down to the pre-knowledge of the particular Anno4j Classes, whose implementations need to be forwarded to subsequent applying other extractors. With the setup of the MICO Platform described above incorporating the Anno4j 2.0 version, the exchange is

18 https://bitbucket.org/mico-project/extractors (last visited 03/12/2018)
19 https://maven.apache.org/guides/introduction/introduction-to-the-pom.html (last visited 03/12/2018)

only possible by adding the new classes to the MICO Platform dependency model, which would require the developer to alter two places of code, in addition to the required publishing of the new MICO Platform version. Another way would be to support the extended MMM metadata model of the own use case in an own dependency or repository, which would not reflect the extendable workflow philosophy of the platform. Especially this can be dealt with much more conveniently with the implementation of Anno4j 2.4, which is depicted in Section 5.4.

Altogether, a MICO extractor does have a defined lifecycle when interacting with a platform instance and its orchestration service, which will be described in the next section. Described by Aichroth et al. in [4], these are a finite set with the start consisting of *extractor preparation*, which constitutes the packaging of the developed extractor and providing it with registration information so the extractor "knows" the address of the MICO Platform to register with. After that, the extractor can be deployed in the *extractor deployment* step, utilising the registration information to connect to a given platform. If supported, testing data is then forwarded to the platform to be used in following testing steps. If the deployment and registration are successful, then the extractor can be included in *workflow creation* steps, in order to be applied in analysis processes of the platform. These can either be for testing purposes or already be real workflows. Once included in at least on of these workflows on the platform side, the platform issues an *extractor process startup* call in order to "activate" the respective extractor, which then will be used in the *workflow execution* step when items are injected into the platform and require processing.

5.2.4 *MICO Orchestration Service - The MICO Broker*

The orchestration service called the **MICO Broker** constitutes the core unit that enables the workflow-centric functioning of the MICO Platform. Autonomously and independently working extractor implementations register at the respective platform instance and get aligned after one another in order to form pipelines of loosely connected processing workflows. Doing this, a transparent analysis is supported, as not just the final results, but also intermediary steps consisting of both pure metadata, as well as potential binary files, are added to the whole metadata background that is produced. With this kind of fine-grained analysis, many different links to applications and use cases can emerge and therefore make the initially ingested media file useable in a much broader field of application.

A more detailed description about the development cycle of the MICO Broker, which also highlights the incorporation of the multimedia analysis extractors, is shown in Section A.1.2.

5.3 FROM A MULTIMEDIA METADATA WORKFLOW TO A SELF-SUSTAINING METADATA CYCLE

With the detailed description of the MICO Platform in mind, a more in-depth view about the metadata that is incorporated is given in this section, highlighting the single steps that eventually constitute the lifecycle of metadata. These steps range from the initial creation of both the multimedia file and metadata, to further metadata allocation, to the point where the metadata is applied in order to profit from its descriptive information about the data that it is about.

In Section 5.1, two different metadata lifecycles have been introduced: the Lifecycle of Multimedia Metadata invented by Kosch et al. [102], describing the metadata that is associated in the multimedia domain, and the Lifecycle of Linked Data by Auer and Ngomo et al. [11] [12] [13] [122], describing a more general view on the metadata overall that is associated with Linked Data. Both of them give detailed insights about the origins of metadata in their respective domain, and thereby create a transparent documentation about incorporated steps, which generally deal with the creation, consumption, and/or exploitation of the metadata. By supporting this documentation, it can also be seen which of those steps are in the biggest focus and which are generally "optional", while having the greatest impact when it comes to the usefulness of the resulting metadata. Next to this, it is also highlighted that the technical implementation of applications that try to align with these lifecycles can orient themselves at these steps, allowing for higher modularity and finer-grained implementations.

In the MICO universe, actually both of the lifecycle principles described above do have impact on the MICO metadata, as the MICO Platform in the first place analyses multimedia data, which is published as RDF and hence Linked Data. Therefore, it is reasonable to devise a combination of both lifecycles, in order to properly attribute both of them and fine-tune the corresponding MICO implementation to the intersection of both. Figure 41 shows a merged combination of the lifecycles for multimedia and linked data. At this point it should be mentioned that the merging was done with a less complex multimedia lifecycle application in mind, as there surely exist applications that optionally incorporate a subset of the steps of authoring, quality analysis, and/or evolution and repair.

The baseline of the combined approach is given by the Lifecycle of Linked Data, seen in Figure 38. Therefore the base steps of extraction, storage and querying, authoring, interlinking, classification, quality analysis, evolution and repair, as well as search, browsing, and exploitation are incorporated, as these steps do always have a role when it comes to linked metadata. Highlighted with the top five circles from top to middle of the figure (with the less complex view

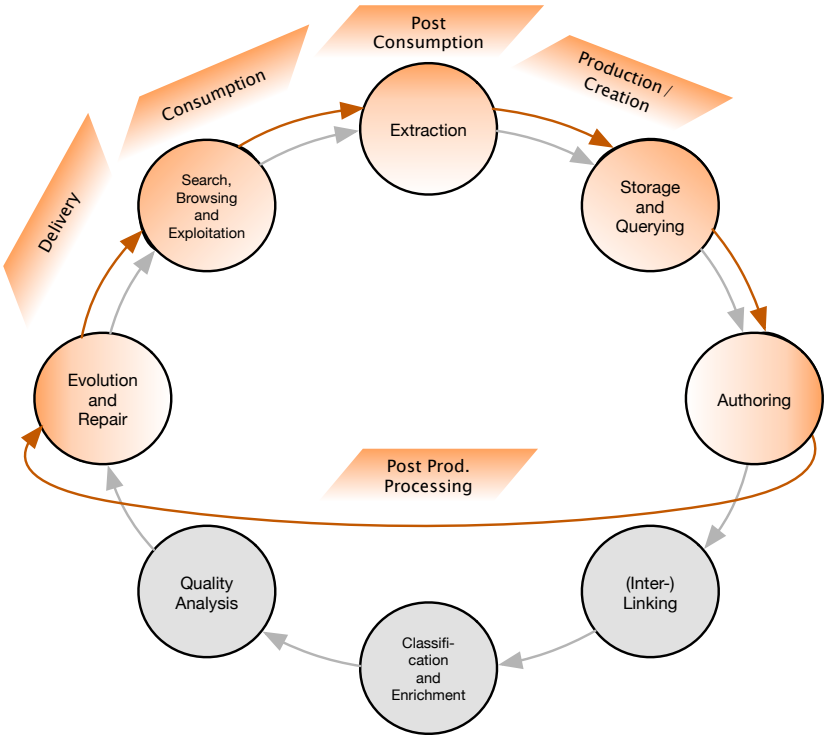


Figure 41: Visualisation of the Merged Lifecycles for Both Multimedia and Linked Data, Resulting in a Generalised Lifecycle for Linked Multimedia Data.

of multimedia applications described above) are the steps that are generally incorporated in multimedia workflows or lifecycles of their respective metadata, adapted to the step descriptions of the Multimedia Metadata Lifecycle. This is possible, as the generalised steps of creation, querying, storage, data processing, and search or utilisation of the metadata are applied in both lifecycles.

By viewing Figure 41, the most important aspects from which the multimedia metadata can benefit from are the missing interlinking and classification steps. The latter is somehow present in common multimedia metadata formats like MPEG-7 (which was introduced in Section 3.1), which introduce structural criteria with their XSD schemata for example. However, these features are by far not as rich as the possibilities proposed by RDF metadata modelling like class and relationship hierarchies, which have been discussed in-depth in Section 2.2. These features can improve for instance querying capabilities by a great deal and therefore influence and benefit all subsequent steps and following iterations of the whole lifecycle.

The feature of (inter-) linking metadata is a point that was not considered in former multimedia applications, as the linking of metadata got its upcoming rise especially in the Linked Data world. Linking to descriptive concepts that are present in other, potentially more spe-

cialised databases, linking results to unique RDF instances, or simply linking various results among each other, are only examples from which the whole application can benefit from.

Relating this back to MICO, the project's metadata workflow was devised around three (more rudimental) core steps: creation, consumption, and subsequent exploitation of the queried information. What was initially seen as a linear set of steps, is actually devising a lifecycle of its own, as in most cases, the implemented extractor-oriented workflow of the MICO Platform can reiterate these steps throughout a whole process chain. With the exploited information of an extractor result, a subsequent extractor can add its own insights and results, therefore supporting query- and exploitable information for another extractor, and so on. This is visualised in Figure 42. In between, querying mechanisms are supported in order to fine-tune queries. As it has been seen, different querying mechanisms supported by the Anno4j library enable a broad variety of access possibilities to the database.

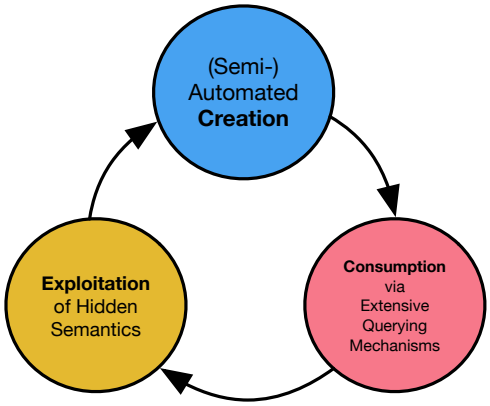


Figure 42: Visualisation of the Core MICO Metadata Lifecycle.

As an example, imagine a use case of detecting animals not just in pictures like it was done before, but rather in whole videos. After a video is tagged as containing animals, it is supposed to be processed further in the envisioned overall workflow. Assuming that the task of detecting an animal in a video is not possible with a single extractor, as it is a very complicated task [3] [5], it will be separated in several steps.

Following this exemplary workflow, the first task would be carried out by a Temporal-Video-Segmentation TVS extractor, which divides the video into meaningful subparts, meaning that the video is separated into several successive scenes (which could be detected by blend ins and outs, scenery changes, etc.). TVS was already mentioned and explained a little bit more in Section 3.3.3. The scene splitting information can now be queried and exploited by the follow up extractor,

which finds the most representative single frames of a given scene (which is oftentimes already done by the TVS extractor as well, but for reasons of clarity it is split in two steps). The identification of a key frame for a whole scene is useful, as the further analysis can be done on this single frame rather than on multiple or all frames of the scene, which not only yields results much faster, but in general contains the most useful information right away in the single frame. The frame detection can also incorporate a direct extraction of the frame as an image on which the following processes can be based on.

When representative frames of the video are extracted, the "original" task of the described use case can begin: the potential detection of an animal. Therefore the animal detection extractor can, again querying and exploiting the created metadata by its predecessor, run its analysis on the images extracted from the overall video. Its results of detecting an animal on a frame can then be semantically applied to the whole scene, or if better or more thorough results are desired, trigger further analysis processes on frames that are contained in the remaining subparts of the scene. Once again, with this potential information of a detected animal, the follow-up extractor can exploit this metadata to trigger its further processes of the given workflow.

Overall in the described exemplary use case, the core MICO metadata lifecycle is re-iterated several times. This allows the concise querying and exploitation of exactly the respective information pieces that are needed by a given extractor instance, and also shows that a recurring exploitation with enrichment of the metadata is not only feasible, but also desirable, as modular and fine-grained metadata is the result that can be utilised in various use cases while also fuelling successive analysis processes.

For the MICO Platform, the merged lifecycle for linked multimedia data shown in Figure 41 and the core MICO Metadata Lifecycle shown in Figure 42 are interwoven, constituting the **MICO Metadata Lifecycle**. The result is visualised in Figure 43.

The general idea of the core MICO Metadata Lifecycle is already contained in the steps that have been seen in the Linked Data Lifecycle, namely extraction, storage/querying, and search/browsing/exploitation, which can directly be mapped to creation, consumption, and exploitation. These will represent a mandatory basis for every iteration of the overall lifecycle.

The other steps of the bottom half of the lifecycle image in Figure 43 are potentially optional, but should always be included when possible or the necessity for better metadata quality exists. Additionally, with the MICO Platform architecture and processing workflow, the optional lifecycle steps should also be implemented following the steps of the core MICO Metadata Lifecycle, interweaving small iterations of the core lifecycle in every step of the overall lifecycle. Therefore, these steps can all be implemented as an autonomously working

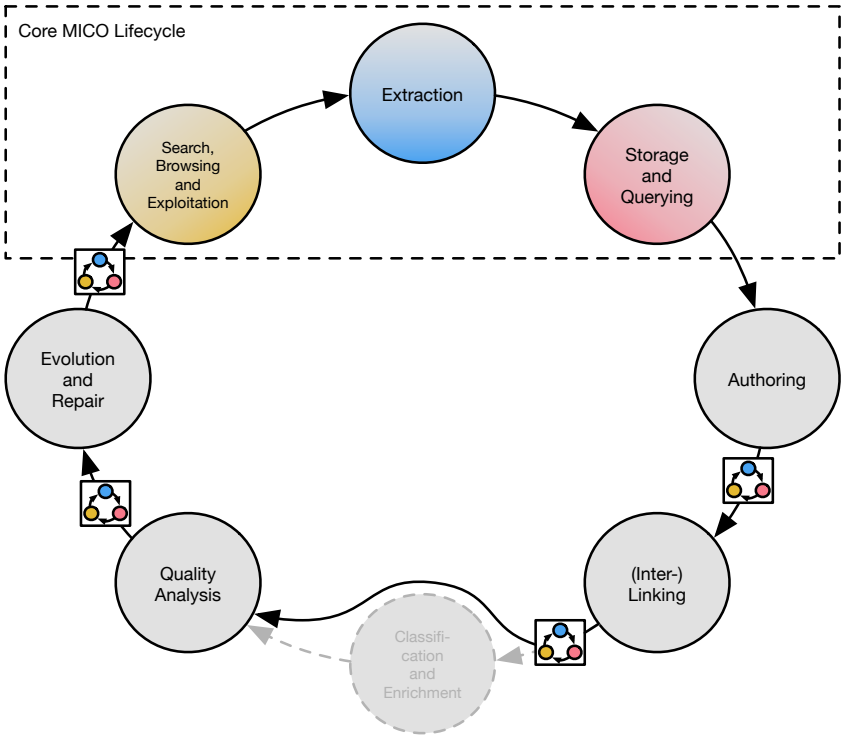


Figure 43: Visualisation of the MICO Metadata Lifecycle.

dedicated extractor. The classification step has been greyed out as the MMM already enforces a classified structure onto the results that are produced by every extractor.

The result is a transparent, modular, quality-variable, and efficiently working (by applying parallel processing) setup for metadata allocation. The realisation of the lifecycle steps as extractors, which in terms apply the steps of the core lifecycle, allow multiple access points to the modular implementation of workflows. Every part of metadata that is added to the database is traceable through provenance features and therefore allows fine-grained querying of results. As every extractor is working autonomously, a given workflow process chain does not run one step after another, but rather in parallel, allowing different steps of the original workflow to be analysed nearly at the same time. As described in the MICO Platform description, multiple instances of the same extractor can be added to even further increase the productivity.

There is also variability in terms of the level of metadata quality, which can be achieved by concurrent optional reiterations of the overall metadata lifecycle combined with quality analysis. Many multimedia analysis processes can run in different configurations, generally allowing to alter between accuracy of the results and shorter calculation times. Inaccurate results or calculations can be done quicker, while more precise results require more training or higher calcula-

tion times. This process can be utilised in the overall lifecycle in a way that in the first instance, possibly inaccurate calculations are performed that can be compared against a threshold value in the quality analysis step. If the result does already suffice the given threshold, no further calculations are necessary or respective following processing can be triggered. In contrast, either by default or because the threshold is *not* met, another workflow chain with the same general use case can be triggered, which applies more thorough analysis processes in order to achieve more accurate results that in terms take more time as described above. This allows both the benefits of described analysis configurations, delivering quick inaccurate results while also calculating the better ones to optionally swap out later iteration inputs.

From a technical point of view, the foundation that is created by the lifecycle approach described in this section is perfectly met by the Anno4j library described throughout Chapter 4. Ranging from the extractor implementations (see Section 5.2.2) to the incremental steps of the core MICO Metadata Lifecycle, Anno4j supports developers as well as users with various functionality to make more convenient use of the respective step's intention. As shown in Section 4.2, the application barrier for the possibilities of both creating and requesting metadata from the underlying database gets lowered by supporting technologies the user is more familiar with. Also, by supporting different access technologies, users have more possibilities of communicating with respective technology. Because of this, potential users and developers can more easily support functionality for example in the form of extractors, analysis processes, or workflows, that can be incorporated into the overall MICO workflow. Also, by lowering the barrier to persist and retrieve data from the respective MICO database, the exploitation gets facilitated. Therefore, further fuelling of the overall workflow by supporting new information is initiated more conveniently. These insights are based on the "basic" implementation of the Anno4j library and the MICO Platform and already show, what useful implications can be done by their combination, while also incorporating the theoretical foundation laid out throughout this work. The following Section 5.4 explains the extended version of the MICO Platform with enhanced functionality that is better suited for the overall use case by applying the latest version of the Anno4j library and some corresponding technical adaptions of the platform.

5.4 EXTENSION OF THE WORKFLOW ENVIRONMENT OF THE MICO PLATFORM BY TECHNICAL FEATURES OF ANNO4J 2.4

Chapter 5, up to now has described the MICO Platform, given concepts and principles about metadata allocation in terms of lifecycles, as well as the combination of both. Additionally, related work has been enlisted in order to learn further details about these topics and

give limitations and additions for further clarification. By describing the functionality of the MICO Platform backed up by its technical information, all of the cornerstones that have been described throughout this work have come together to form a multimedia metadata platform that allows the distributed analysis of multimedia content by applying autonomously working extractors in a workflow-centric fashion. By utilising frameworks or libraries, like the Anno4j library in this case, it was shown that the barrier for the development and therefore the contribution to such Semantic Web-oriented applications can be reduced significantly, allowing non-experts to take part more conveniently, while supporting richer development possibilities for developers that are already familiar with Semantic Web technologies. This overall creates an abundant and potent basis for Semantic Web applications.

All of the descriptions of this Chapter 5 have been based on the core implementation of the Anno4j library with version 2.0. However, the library's incremented version 2.4 introduces new features as well as adapted and enhanced existing features that allow for a much more thorough and technically richer implementation of the overall MICO Platform (throughout Chapter 4 both versions of the library and their differences are described).

To show the possibilities, this section will highlight the optional extensions to the MICO Platform with the extended functionality of the Anno4j library. In the further work, this extended implementation of the platform is called **MICO+**. It is important to note that this can only be done on a conceptual basis. However, the applied features of the Anno4j library are implemented, published, and for critical features also evaluated, supporting a practical background for the claims stated above.

The enhanced features that will be described here give an extended functionality for extractors that mainly focuses on a better way to exchange the metadata model amongst each other, the introduced validation features of Anno4j in order to preserve data consistency for the whole platform instance, as well as the incorporation of the RESTful Anno4j implementation, which eventually does widen the applicability of the overall platform implementation. This again does promote the various access mechanisms that then can be utilised in order to communicate with the platform and therefore contribute to its workflow functionality. Doing this, non-experts as well as experts are able to work with the platform more conveniently, creating a broader field of possible users and developers.

EXTENDED SCHEMA FUNCTIONALITY FOR EXTRACTORS A drawback in the extractor connection that is presented in Section 5.2 is the exchange of the actual metadata schema. The core structural information about the metadata of the MMM (so the "`mmm:Item`", "`mmm:Part`",

their relationships etc.) are incorporated in the MICO Platform Maven dependency, which every extractor has to incorporate in order to be able to produce and understand the basic structure of the MICO metadata. This is convenient and the way how it should commonly be done. The disadvantage however comes into play when the metadata is considered that is extractor-specific and therefore not known to others in advance.

The schema for the core extractors that are automatically incorporated with the basic installation of the MICO Platform is included in a sub-schema of the MMM, the MMMTerms vocabulary, which is incorporated in the MICO Platform dependency, so these extractors can be used out of the box. Incorporating an entirely new extractor however does pose some obstacles.

Every newly incorporated extractor generally introduces some new metadata schema. With the "old" platform implementation, this information at the first point is only available and known to the project of the extractor itself. The respective developer then has to make sure that the respective descriptive schema is accessible to further extractor developers, in order to enable access and understandability for the prospectively created metadata of the first new extractor. It is also possible to incorporate the structure of the new metadata at the dependency for the MICO Platform, as it is open source and overall contribution "from the outside" of the MICO universe is desired.

However, both of these possibilities require the developer to touch at least two different "places" of code. This is partly inconvenient by itself as the synchronisation can pose difficulties. Especially when multiple new extractor instances are incorporated to one platform, the synchronisation among several different places in order to have a common and decided upon metadata schema becomes unsolvable.

To counteract this, the Anno4j Generation Tool (see Section 4.6) allows a promising solution. The sole input of the tool is an RDFS or OWL schema, a file that is easily share- and extendable, while the functionality of the Generation Tool is by default incorporated in the Anno4j Maven dependency, a dependency that is automatically required for every extractor implementation. Envisioned is a Git[20]-like [45] distribution, which is managed at the respective MICO+ Platform instance. The workflow is visualised in Figure 44.

On the left side of Figure 44 a productive and running MICO+ Platform instance is depicted, having several already registered extractors (`"extractor 1"` through `"extractor n"`) with various different own schemata, which is also depicted in a combined manner. From the middle to the right side of the picture an extractor (`"extractor x"`) is shown, which is supposed to be a new extractor that is to be incorporated into the overall platform workflow. Therefore, the extractor sends an already existing register message to the platform instance,

20 https://git-scm.com/book/en/v2 (last visited 03/12/2018)

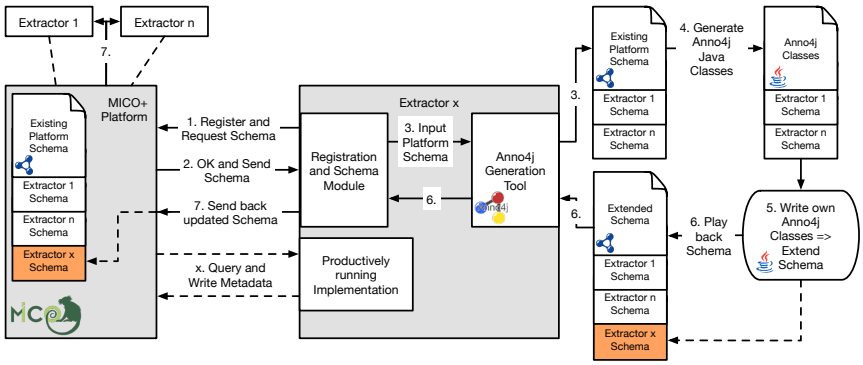


Figure 44: Workflow of Registering and Exchanging Schemata for the Extended Extractor Implementation at the MICO+ Platform.

combined with a request for the currently present schema of the platform. A desirably successful registration message and the respective RDFS schema information is replied to the extractor, which can input this information into the Anno4j Generation Tool, to fully generate all necessary Anno4j Classes inside the Java project of the extractor. This allows the developer to fully utilise the basic MMM model, as well as all the other supported extractor metadata.

Therefore, the extractor developer can directly "understand" the metadata that is produced by other extractors, allowing him to insert his new extractor in between an existing workflow process chain, or tie it at the end. Furthermore, it is of course possible to add new Anno4j Classes and therefore contribute to the overall metadata model. These changes and new additions are firstly implemented as Anno4j Classes, which will be transformed to respective RDFS metadata, extending the currently existing metadata model. These changes are then automatically propagated back to the platform instance, allowing them to be merged with the overall metadata model. These changes then are also forwarded to the other already existing extractors. This enables all extractors to always know every result that is produced in the given MICO+ Platform universe. This workflow does also not interfere with the existing extractors, as they do not necessarily have to react to updated metadata models. After the initial synchronisation has been done by the newly added extractor, it can enter the state of commonly working with the platform and therefore receive tasks by the platform, query and write metadata to its database and so on.

The scenario described above poses most of the processing work in terms of the metadata model onto the client, as every extractor instance must run its own Anno4j Generation Tool in order to create the necessary Anno4j Classes. Runtime evaluations for this process are shown in Chapter 6. This is straightforward, as the client itself

does have the clear view of how long this process is taking, he does not have to wait for the platform to respond.

There is however another possibility: transferring the code generation to the MICO+ Platform instance. To perform this, the Anno4j Generation Tool would run at the platform, triggered with every update to the overall metadata schema that can be done by a newly incorporated extractor. The output are Java classes, which can be serialised and then also be transferred to the respective extractors. This does relieve the extractor instances of the process of generating the Anno4j Classes, but it does also pose bigger message bodies to the HTTP messages that are sent between platform and extractor. Also, with bigger and always growing schemata (e.g. the CIDOC CRM used in the Visit Project, see Section 6.3), this possibility might be unfeasible.

This creates a convenient working environment for extractor developers, who are then only required to implement and adapt their own extractor project rather than two code places as described above. No extractor code synchronisation is necessary to retrieve and contribute to the metadata model.

VALIDATION OF PRODUCED EXTRACTOR RESULTS Another important aspect of the overall MICO Platform that can be extended by Anno4j features is the validation of extractor results. With the implementation documented in Section 5.2, there is no validation at all. So consecutively, the input for the knowledge base of a platform instance is not controlled. Therefore, clients and their extractors are basically not obliged to produce their extractor results in the way that the MMM proposes. This does counteract the intended benefit of RDF schemata, allowing a common understanding of how metadata is supposed to be structured in order to produce collaboratively interpretable results.

The validation feature of the Anno4j library can be used on the client and extractor side and there be interwoven into the code that would create metadata. This process then directly precedes the step of sending the created results to the MICO Platform, for example allowing the possibility of throwing exceptions, if the validation criteria are not fulfilled, or even incorporate these exceptions into the respective extractor program code, opening even more possibilities. Next to this, the validation features of Anno4j would also allow to fine-tune the overall platform metadata schema, so more precisely the `"mmm:Body"` class, in order to enforce the structure for created metadata by the extractors. For example, a cardinality requirement addressed at the body components could cause the metadata model to be flat at this point of the graph, enabling an easier to understand structure. This would induce an automated quality check when an extractor attempts to create metadata.

RESTFUL ANNO4J TO EXTEND PLATFORM DATABASE ACCESSIBILITY As described throughout Chapter 4, the Anno4j library allows for a RESTful application, opening a REST API that is accessible via commonly known HTTP messages. This does automatically extend the accessibility of the database, allowing extractor developers not only to communicate their metadata via the basic way of the "classic" Anno4j implementation, but rather via commonly known HTTP calls sent by their extractors. Similar as with the overall Anno4j concept, this enables a diversification of access technologies, broadening the potential users and developers that contribute to the MICO Platforms. In addition, by being able to write and query metadata via HTTP calls, the implementation of extractors generally is not restricted to the Java programming language (or C++ with the Anno4jCPP extension), but can be done in any language that allows for the HTTP calls. Using this possibility however does compromise the richness of the implementation, as with Java several interfaces and code snippets are already supported, which alleviates the developer's process of implementing an extractor.

In order to generate the REST API, the sole input necessary is the RDFS schema file of the current metadata model, so in terms the MMM with the added extractor specific vocabulary. Interwoven with the idea of schema synchronisation issued above, the extractor schema information can be exchanged with the platform instance as central process piece. After receiving a new registration of a new extractor, respective API functionality can be generated and then be applied to the overall existing REST API.

5.5 MULTIMEDIA METADATA APPLICATION CONCLUSION

In retrospective, Chapter 5 describes lifecycles of metadata, gives details on platforms that mainly deal with multimedia data, as well as the combination of both. The lifecycles allow to raise the awareness about which process steps are applied in the life of metadata. Therefore, applications can utilise this knowledge in order to fine-tune their utilisation of respective functionality in the way that the given use case requires. The cyclic concept dictates that with the iteration of a lifecycle, metadata is created, refined, requested, applied, and in the end, it potentially can re-trigger the whole lifecycle by sprouting new eventually more specialised and detailed metadata. Hence, the lifecycle does not necessarily "end" at some point, always increasing the generated metadata background for the initial input.

This information was mainly derived from lifecycles of the Linked Data domain. But there was also a lifecycle out of the multimedia domain, depicting important steps that are undertaken for multimedia metadata creation. The Lifecycle of Multimedia Metadata introduces important processing steps, while also implementing various meta-

data spaces that are traversed by the eventually produced metadata. Next to this, different user groups are identified that take part in the metadata lifecycle.

After the lifecycles, the related work of Chapter 5 shows similar work related to multimedia platform applications in order to compare it against the MICO Platform afterwards. A multimedia platform instance resembles a central running software application, whose general task is to manage multimedia analysis processes that overall create metadata of ingested multimedia files. These files can then be utilised in combination with the analysis results and therefore be more useful, as the metadata supports descriptive features about the given media file. There exist various approaches for this problem, with different strategies, incorporation of external extractors, communication protocols, and so on.

With the insights attained from the related work, Section 5.2 gives a detailed description of the MICO Platform, a multimedia analysis platform. This platform represents the accumulation of the contributions described throughout this work. Therefore, next to a general description of the overall platform infrastructure, extractor implementation, communication, and orchestration, this chapter highlights the mutual benefits of this work and the MICO Platform. Especially the factors of the incorporation of the Anno4j library are shown.

Then, the MICO Platform documentation is combined with the insights gained from the related lifecycle descriptions, which were combined to develop the Lifecycle for Linked Multimedia Metadata. The MICO Platform generates metadata that is based on Linked Data principles, and therefore creates an interlinked and traceable metadata background for input multimedia data. The basic workflow for the platform consisted of three steps - creation, consumption, and eventual exploitation - which could be interwoven with the overall metadata lifecycle, enabling comprehensive and transparent interactions at various occasions. This concept once again could be backed up by the Anno4j library implementation.

This chapter is concluded with an outlook for the MICO Platform, whose implementation was ended when the MICO Project ended in 2016. In contrast, the implementation for the Anno4j library did not stop, and therefore the library at this point of time supports richer and more comprehensive features that can alleviate the MICO Platform implementation with the state that is described in Section 5.2. These extended features are described at the end of the chapter on a theoretical basis.

Overall, this chapter shows that metadata is a very important cornerstone in the domain of multimedia, without which a thorough usage of the multimedia files would not be possible, especially in the world of today that lives through convenient production and consumption of multimedia content. Throughout the whole chapter and

related work, it is also apparent, that the process steps learnt from the lifecycles are in need of technical support to make these steps transparent and accessible on the one hand, but also rich on features and functionality on the other hand. The results of this work achieve that: the MMM poses a modular and extensive metadata model for the multimedia domain, the Anno4j library implements a thorough possibility to interact with produced metadata at various points of metadata lifecycles, while the MICO Platform poses a perfect fit for the before mentioned features and eventually creates a platform for multimedia analysis. In the end, the connection of all these concepts and implementations produces a multimedia analysis platform that allows for convenient collaboration, eventually producing a metadata background in the form of linked multimedia metadata that allows the input multimedia files to be utilised in a broader field of application and potentially yet unseen scenarios.

Part IV

EXPERIMENTS AND EVALUATIONS

6

EXPERIMENTS AND EVALUATIONS

This chapter picks up selected contributions of this work and tries to substantiate them by evaluations in order to further foster those issued contributions. It is structured as follows. Section 6.1 will explicate some of the experiments and results obtained by Quasthoff [132], as the author examined and researched ORdfM libraries and frameworks from a more general perspective.

Afterwards, the term complexity in relation to Semantic Web ontologies and their pendant of domain models in Anno4j will be picked up. Several ontology parameters will be enlisted and a complexity measure is defined in Section 6.2. Then, three evaluations are explained, which apply these parameters in different fashion. The evaluations also implement the Anno4j Generation Tool to create respective domain models, from which instances are instantiated and the runtimes are tracked.

The first evaluation shown in Section 6.2.1 targets the ontology parameters in isolation, fixing the other five to a mean value devised from an analysis of openly accessible LOD ontologies. This allows to get a first estimation about the impact of the parameters on overall runtimes. Section 6.2.2 describes the second evaluation, which is similar to the first one, with the difference of incorporating pre-generated Proxy Classes (which have been introduced in Section 4.6). It is evaluated, if the promised benefit of these Proxy Classes to reduce creation times of domain model objects holds. The last evaluation shown in Section 6.2.3 examines the recombination of the mentioned ontology parameters in order to determine dependance on each other, as there might be increased impact on overall runtimes.

Another way of evaluating the usefulness of an implemented piece of software is showing its real-world application and adoption. The ViSIT Project (also already referred to throughout this work) is a research project at the University of Passau in the fields of cultural heritage, working with Semantic Web metadata and therefore applying the Anno4j library. Section 6.3 will give a short overview of the project and will highlight the beneficial factors that are introduced by the utilisation of Anno4j, therefore again accentuating its usefulness in such applications. Additionally, an envisioned feature called "Semantic Zoom" will be discussed, which can potentially enrich the ViSIT use case by introducing semantical abstraction layers to the underlying semantic data. This will eventually increase the data handling potential of the overall use case on the one hand, as well as diversify and facilitate data access on the other hand.

6.1 RELATED WORK - OVERALL ORDFM EVALUATIONS

This section cannot be seen in the same way as the other related work sections of this thesis, as it will not depict related approaches to a given topic in order to establish delimitations or enlist insights that had impact on our own work. Instead, it will show evaluations conducted by related work that refer to the overall concept of ORMs / ORdfMs. The concepts are already well-known and established, therefore a conceptual evaluation is not necessary. However, this section will first pick up some of the insights that have been already discussed throughout this work, then add some related work evaluations, and finally discuss these results and their impacts on the presented contributions in respect of their usefulness and applicability.

GENERAL OBSERVATIONS: ORDFM IMPLEMENTATION EFFORT AND OVERALL RUNTIMES Some quantitative statements that concerned the overall concept of ORdfM implementations have already been presented throughout this work, especially in Chapter 4. There, the prevailing general observations targeted two topics: overall implementation effort of an ORdfM vs. implementing respective features by hand, and the runtime drawbacks that potentially arise with the application of an ORdfM.

The advantages of ORdfMs have been described on many occasions now, but it should be remembered that the actual incorporation of an ORdfM into an application environment takes a considerable amount of development time and effort, especially if compared to the implementation of respective features (so for example querying and persisting a given data model) by hand. This timely effort can potentially be reduced, as there are many already existing ORdfM frameworks and libraries out there, however, also the familiarisation with them can still require more time. A quantitative parameter that can represent the deciding factor on this decision is the size of the data model that is supposed to be utilised. Small models can be implemented and handled by hand, as only a considerable amount of queries needs to be established to cover necessary functionality. But already medium- to large-sized models already suggest the incorporation of an ORdfM.

However, although the incorporation and familiarisation requires some extra effort, the application of an ORdfM comes with a series of benefits. An increase in productivity, reusability, maintainability, design liberty, as well as a decrease in error-proneness are only some of them. Therefore the concept allows for a more thorough design and implementation of an application, that is not only increased qualitatively in the implementation phase itself, but also as a sustained yield at a later stage and state of the application.

6.1.1 *Quasthoff: General ORdfM Comparison*

The first evaluation conducted by Quasthoff [132] in 2011 compares different (at that time prevailing) ORdfM implementations. These are namely RDF2Java[1] (implemented originally for a management system of weak workflows [63]), RDFReactor [171] (or RDF2Go which has been covered in Section 4.1), SuRF[2] (a Python based ORdfM), and Elmo/Alibaba.

The evaluation is held as a series of telephone interviews with several respondents and based on the setting of Paré [125], answering two main questions:

- Why does the respondent apply the ORdfM concept?
- Why did the respondent implement an own solution, rather than using an existing implementation of the concept?

Quasthoff posed some other technical and more detailed questions about the respective implementations of the respondents, which will not be covered here entirely, but rather single answers considered to be important towards this work will be pitched on.

Quasthoff has also taken some requisitions into account. The questionnaire is only done with the main developers of the ORdfMs, as they are the main applier of their respective implementation and they are aware of the requirements that have been brought into the development. Additionally, the evaluation reveals that ORdfMs are rarely picked up by the community and therefore they are mostly used in own projects of the developers. Therefore a user study would not be useful. The evaluation also mentions that all of the respondents think that the low reusability of an ORdfM implementation is unavoidable. The results for the first evaluation can be subsumed under the following categories:

META-PROGRAMMING Statically typed languages like Java are compared with dynamically typed ones like Python. Advantages and disadvantages are enlisted and contrasted for both possibilities, but the overall insight gained is that there are only slight differences and therefore the chosen programming language does not have an impact on the implementation of an ORdfM. Hence, the user can choose the programming language freely, only depending on his preferences.

COMPLEXITY AND TASKS OF AN ORDFM A second result targets the complexity of ORdfM implementations, as well as the intended tasks that it is supposed to fulfil or support. The general opinion was that basic RDF libraries like Jena (and also mostly mentioned

1 http://rdf2java.opendfki.de/ (last visited 03/12/2018)
2 https://pythonhosted.org/SuRF/ (last visited 03/12/2018)

Elmo/Alibaba itself) are unnecessarily complicated to use. The respondents also agreed that it is not possible to have a single ORdfM implementation which can serve every use case in a sufficient fashion, as every use case poses varying requirements. As a side note the respondents also mentioned that ORdfM implementations should also stick to their main task: the translation of RDF data and representative objects. Everything that exceeds this task should be ignored and solved by other components. Examples are mentioned in the form of the connection to distributed databases, licensing and privacy, and inferencing/reasoning.

MOTIVATION TO USE THE ORDFM CONCEPT The motivations and reasons to implement an ORdfM differ and range from pure concept implementations to being part of larger projects. However, all of the respondents issued that they would always use an ORdfM implementation whenever they are in need of RDF data handling, regardless of the overall use case or the underlying data model. Some of them even used learnt insights gained from one concept implementation in order to develop an increased version. Also the general observation prevailed, that documentation of an ORdfM implementation should be made public, and collaborations between different peers are desired in order to make the best out of the concept applications.

6.1.2 *Quasthoff: ORdfM Implementation and Runtime Evaluations*

In contrast to the evaluations in Section 6.1.1, which focused on existing ORdfM implementations and their developers' opinions, this section will describe the second set of evaluations done by Quasthoff [132] that target the potential increase of productivity when utilising an ORdfM library or framework. This is done by an experimental evaluation with software developers using both an ORdfM implementation compared to an approach without ORdfM, backed up by an evaluation on the runtimes of applying an ORdfM in an application.

EXPERIMENTAL EVALUATION WITH SOFTWARE DEVELOPERS In this evaluation, Quasthoff, Sack, and Meinel [134] evaluate the increase in productivity that is potentially gained by using an ORdfM in your Semantic Web application. Several IT students of the Hasso-Plattner-Institut at the University of Potsdam were asked to solve two tasks in mixed setup combinations, supporting their opinions and experiences afterwards in a questionnaire. The tasks consisted in requesting different information persisted in an RDF graph, which are semantically composed of *Documents*, written by *Persons* that have friendship relations amongst each other. The first task demanded to retrieve the authors of given Documents, the second task demanded friendship relations between persons of the second degree, given an-

other Person. The participants were using OTMj [133] (an ORdfM implementation developed by Quasthoff and Meinel themselves, aligned to the results the authors gained by their evaluations, which are partly described in Section 6.1.1) and Apache Jena as a more "basic" approach to access RDF data. The group settings alternated in terms of which task had to be done using what technology and their respective order, constituting four different group settings. With this setup and especially the tasks done by students rather than experts, the authors claim that the results conducted cannot be projected on any generic Semantic Web application or project, but the results still support thorough insights on the productivity of ORdfM implementations and emerging issues with respective solutions that can arise with a Semantic Web application.

The metrics that have been evaluated in the experiment were rated by the participants on a scale ranging from "very easy" to "very challenging". They had to assess the *difficulty of the task*, the *maintainability* of the code they produced, as well as the difficulty to use *alternative technology* (like XML via HTTP or relational databases) for the same task. Next to these features, the participants were also asked to support free feedback concerning their *perceived biggest difficulties* in regards to the issued tasks.

In addition to these qualitative metrics supported by the participants themselves, it was tracked how much *time*, how many *programming attempts*, and how many *lines of code* were required by each participator. For further technical insight and understanding, these lines of code were also categorised in terms of the high-level functionality that they fulfilled. These are *structural constructs*, *initialisation*, *data access*, and *business logic*.

The results show, that the participants assess the tasks much easier when using an ORdfM implementation rather than Jena. In addition, they also estimate their written ORdfM application much easier to maintain, and they claim that using Semantic Web technologies is easier than alternative technology. These estimations were achieved regardless of which of the two tasks.

In regards of time requirement and the amount of tries to retrieve the correct information for task one, the participants on average required significantly much less effort for both metrics when using the ORdfM implementation. For task two, the amount of tries surprisingly went up, while the average time to fulfil the task went down. The tries were potentially motivated by the fact that the task was more demanding in terms of recombination of the results, but the reduced time for the solution still indicates a real benefit of the ORdfM utilisation.

The lines of code do also experience a distinct positive tendency towards the ORdfM implementation. While the lines for structural constructs and business logic are roughly the same for the solutions

with and without ORdfM, the required lines of code for the remaining functionality are reduced significantly for the ORdfM solutions. Next to this fact, the utilisation of an ORdfM allowed the participants for a much better separation of respective code categories, positively influencing the code's overall maintainability.

The results of the general feedback about experienced difficulties revealed insightful details. A lot of the participants issued that Jena and its functionality by itself is hard to understand and apply. Several incorporated classes and some implemented features were deemed confusing, and especially as a non-Semantic Web expert, hard to understand and therefore to use. In conjunction with this issue, the participants also added that without the supported code examples (which were supported by Quasthoff, Sack, and Meinel, not the framework) the finding of a solution for given Jena tasks would take even more time. As a second argument it was noted that the interviewees oftentimes had problems understanding the respective RDF vocabularies.

RUNTIMES OF ORDFM IMPLEMENTATIONS While the above evaluations and experiments targeted convenience factors as well as development efficiency in the first place, it is also important to evaluate the resource efficiency of related ORdfM implementations. Having an inefficient implementation resource-wise could render any application unusable, no matter how convenient or well-developed in terms of other features it is.

Quasthoff [132] indicates that evaluations about the requesting of RDF data is explored by Hartig [77], so his own evaluations will be targeted at RDF data that is held in the main memory. This experiment is partially based on the *Berlin SPARQL Benchmark BSBM* by Bizer and Schultz [31], as the SPARQL requests used in the benchmark are much more complex than general ORdfM interactions. Therefore, Quasthoff instead runs through the whole graph by traversing every edge supported by the test data, utilising 30 different RDF graphs that contain between 40000 and 970000 triples which describe a fictional domain about persons, products, traders, offers, and ratings.

Again, different test setups are used in order to generate best possible insights, applying the general Jena implementation against the own developed OTMj implementation. These are: accessing the edges without ORdfM support with object-oriented techniques, access with ORdfM with its interactions translated into object-oriented techniques, access via pure SPARQL, and access with ORdfM support whose queries are translated into SPARQL queries.

Quasthoff reinforces his experiments with quantitative numbers, amongst which the results are to be recapitulated here. The major result, being the most impactful for this thesis, is that the utilisation

of an ORdfM implementation in a Semantic Web application does only have a slight impact on the runtime and it is therefore beneficial to add the abstraction layer for a diversified access to the RDF data. It is also explained that this outcome holds true whether SPARQL is used or not. Nevertheless, it is shown that the incorporation of SPARQL requires a considerably low amount of extra time in the SPARQL tests. Overall, this does still not invalidate the assumption made above, however for a maximised implementation on in-memory data it is reasonable to bypass SPARQL when possible. When working with decentralised data, which does generally not allow for the approach that is applied for the in-memory data, the SPARQL time constraint does carry less weight, and therefore the ORdfM application does still hold true. As a side note, the author mentions that this result does align also with the statement of the interviews of the ORdfM developers (which have been recapped in Section 6.1.1), who issue that there cannot be a single best ORdfM implementation: depending on the given use case you can fine-tune your at best very slim concept implementation. The same case applies for the various runtime possibilities.

LESSONS LEARNED - IMPLICATIONS OF OVERALL ORDFM EVALUATIONS FOR THE ANNO4J LIBRARY While the remainder of this work up to Chapter 6 explained the concept of the Semantic Web, metadata models and multimedia, ORdfM libraries and frameworks with the specific representative Anno4j, and the junction of all this in the MICO Platform, a central software framework to analyse multimedia, the concept of ORdfMs and their advantages have been applied whereas some prerequisites have been left out. Mainly from a development and runtime perspective, ORMs in the world of relational data have been publicly accepted and evaluated. Quasthoff's results which have been explicated in this section confirm the same assumption for the ORdfM concept.

The insights to take away for the Anno4j library as an ORdfM implementation are the following. On the one side there are implications that can be expected and adopted for the library, and on the other side, with the results made in the related work of this section, assert confirmations for some of the claims made throughout this work.

The Anno4j library does align with the opinion of the consulted ORdfM expert developers about the main task, which a representative implementation is supposed to fulfil: deal with the RDF data communication while refraining to solve other associated problems in order to keep the implementation as concise and potentially convenient as possible. Therefore Anno4j focuses on the accessibility and therefore different possibilities to both produce and consume RDF data which is supported by the Anno4j Generation Tool in order to facilitate the access to an underlying domain model. This point is es-

pecially highlighted by the expert developers to be an issue of great importance and hence poses a thorough contribution by this thesis. Next to this, supportive features like the validation allow for further broadened and extended application of the library.

Another more general argument is given by the developers' opinion that they would always apply the abstraction layer supported by an ORdfM implementation whenever they want to develop a Semantic Web application. This shows that, although research about this topic seems to stagnate, ORdfM implementations still have impact and therefore are an essential cornerstone in the Semantic Web. Making the Semantic Web and its technologies a more present and represented entity in the overall Web of today is only possible with approaches like the ORdfM concept.

In terms of runtimes with ORdfM applications, Quasthoff's evaluations have shown that the utilisation of the abstraction layer does not have an impact and therefore their application is always beneficial. This does hold true whether SPARQL queries are applied in between or not. His evaluations have been based on his own ORdfM implementation OTMj and the underlying Apache Jena. Although Anno4j is based on Alibaba and therefore Sesame/Rdf4j, comparisons of the respective triple stores and functionalities have shown that Sesame does perform at least as good as Jena, if not better [48] [166]. Therefore Anno4j does not impose any runtime issues, backed up by the real-world application of the library in the MICO Project (see Section 5.2) and the ViSIT Project (see Section 6.3), which showed no runtime problems as well, and the evaluations seen in Section 6.2.

Overall, the experiments in this field of research indicate that the overall application of the ORdfM concept is not just practicable, but also beneficial in terms of development efforts. Additionally, the offered possibilities allow for more sophisticated and especially accustomed or familiar access to RDF data, while not imposing any (or only slight) drawbacks in terms of runtime and hence can be applied without any concerns. These assumptions support the Anno4j library and therefore the contribution of this work from an overall conceptual background. What is left to show is that the features proposed in this thesis do not contradict the above experimental results, which is done in the following section.

6.2 ANNO4J AND ONTOLOGY STRUCTURE EXPERIMENTS

This section will show evaluations of overall creation runtimes of Anno4j Class instances, as well as benchmarks for the main novel feature introduced by the Anno4j library - the Generation Tool. These evaluations will determine the impact that the utilisation of the library can have on the runtime of an application using its functionality. If the overall runtime would be increased by a to large of a margin,

it is not possible to use the library or the respective feature in a real-world application. Previous to the evaluations, ontology parameters will be introduced, which can be altered in order to implement the configurations for the experiments. In addition to this, these parameters will define a complexity measure for the input ontologies and the pendant of domain models.

The experiments done here will evaluate the overall runtime of Anno4j in combination with its generation feature in Section 6.2.1 with a simple test setting, which is complemented by an evaluation using pre-created proxies in Section 6.2.2. Afterwards, Section 6.2.3 describes the application of a more thorough test setting, which varies multiple ontology parameters in order to achieve more profound results and insights. The Anno4j generation feature has been introduced originally in Section 4.6.

6.2.1 *Runtime Experiments with Generated Anno4j Domain Models*

Throughout this work it has been mentioned at different occasions that complex ontologies and respective domain models can induce longer processing times for applications utilising the ORdfM concept, like the Anno4j library. This does become noticeable at the point of time when metadata is to be created via Anno4j. To be more specific, with an existing domain model, an instance of one of its representative Anno4j Classes is to be created. It has been determined that the runtime to perform this is influenced by the complexity of that given Anno4j Class. In order to achieve insights about the runtime efficiency of the Anno4j library, this section will show benchmarks that evaluate the creation runtime of different configurations of Anno4j Classes with underlying domain models. As it was also mentioned next to the runtime issues, a solution for increasing runtimes has been implemented for the Anno4j library as well. By being able to assess the complexity of a given ontology, decisions can be made if an initial proxy generation step is useful or not, as the generation imposes a relatively fixed pre-timing interval, but at the same time reduces the runtime of creating objects, which will be shown in Section 6.2.2. However, before the evaluations can be explained, it is important to gain insights about the notion of "complexity" for ontologies and domain models.

RELATED WORK - SOFTWARE AND ONTOLOGY COMPLEXITY In the fields of computer science, estimating the complexity of various measures is a commonly known process, which poses advantages and insights about further utilisations of said measure. One of the major domains is software development and software structure.

Different approaches are to be found in related work, which have their common factor based on software complexity. Khoshgoftaar and Munson [98] relate their complexity measure to the possibility of aris-

ing errors in the evaluated software. They try to use a statistical tool in order to use regression analysis to identify the possibility of modules creating errors, once the whole application is used live. Their analysis is done before the test phase, allowing to do adjustments and reduce error rates at early phases of the overall development cycle.

Another similar approach is given by Kafura and Reddy [95], who determine the relationship between software complexity and the effects of maintenance activity. It has been proven, that maintenance costs outweigh the actual costs of development [35], and therefore the results of the authors can lead to eventual cost savings. This is even aggravated by the fact, that with increasing size of the overall application, associated maintenance activities increases even more. In order to determine the complexity of software parts, the authors apply various known metrics, with which further adjustments can be made to the software, or arising and expected maintenance activities can be estimated, adapted, and focused, which eventually leads to a possible reduction of said activities and costs.

For choosing their technique for the evaluation, Kafura and Reddy ponder between objective and subjective possibilities. While the former tries to correlate the complexity metric of a component to the times when an actual maintenance task needs to be performed for that component, subjective techniques relate the estimated complexity to the opinions and judgements of experts, who are familiar with the reviewed system. The authors' choice is a subjective technique, motivated by the fact that they wanted to see if their complexity measurement can support some sort of guide to avoid poorly performed maintenance, applicable by a maintainer or maintenance manager.

Their complexity measure is targeted at source code and is constituted by seven different metrics, which they divide into **code metrics** and **structure metrics**. Their code metrics are given by:

- McCabe's Cyclomatic Complexity Number [116]
- Halstead's Effort Metric E [74]
- Lines of Code

Amongst the structure metrics are:

- Henry and Kafura's Information Flow Metric [82]
- McClure's Control Flow Metric [117]
- Woodfield's Syntactic Interconnection Measure [180]
- Yau and Collofello's Logical Stability Metric [182]

These metrics were chosen because of their difference between the respectively measured factors. Next to their study and its results, Kafura and Reddy also support an analysis tool that is able to calculate

above metrics for a given Fortran source program. Their analysis is done on a single mid-sized system, of which several consecutive versions are considered. They calculated the metric values for each version by itself, and then tracked the changes in complexity and maintenance activities over the various system versions, with and under the guidance of two expert developers.

As their conclusion, the authors see their issued claim about the usefulness of determining the complexity in regards of potential maintenance confirmed. Furthermore, they mention that changes in software code to reduce a given metric does definitely have impact on eventual maintenance activities. Also, they have identified that outliers in terms of the metrics are rare, but they need definite attention, as they can have grave impact on eventual maintenance.

With these approaches from general software design and the importance of ontologies in the Semantic Web, which has been explained and mentioned throughout this work, recent research has also adapted the process of quality of complexity measurement for ontologies. Similar to the software pendant, measuring these factors for ontologies does also promise beneficial input for the design, the improvement, eventually arising maintenance activities, as well as the choosing of right ontologies for a given use case [181].

One related approach is given by Yao, Orme, and Etzkorn [181], who focus their ontology complexity measurement on cohesion metrics. With again the counterpart of software cohesion in mind, their approach relates to the desired circumstance of having a good separation of responsibilities, independence of components, and control of complexity between object-oriented classes. The authors adapt this definition and see ontology cohesion as a way to symbolise the degree of relatedness between OWL classes, which are conceptually as well as semantically related by the defined properties between them. They also claim that if the entities of an ontology are thereby strongly related, then the given ontology has a high cohesion, which is motivated by the idea that ontologies should be related in terms of a domain. The parameters which the authors consider in order to determine ontology cohesion are the following:

NUMBER OF ROOT CLASSES The number of RDF classes without an ingoing `"rdfs:subClassOf"` relationship.

NUMBER OF LEAF CLASSES The number of RDF classes with no outgoing `"rdfs:subClassOf"` relationships.

AVERAGE DEPTH OF INHERITANCE LEAF NODES A path is defined by an inheritance chain going from a root node to a leaf node, while the length of the path is the amount of classes on the path. The total number of paths of an ontology is the total number of distinct paths from each root node to each leaf node. The value

of this metric is the sum of depths of all paths divided by the total number of paths.

In their empirical evaluation, Yao, Orme, and Etzkorn gathered a set of ontologies and gathered a group of eighteen evaluators, from both the fields of software development and knowledge based systems with three to five years of experience, in order to let them evaluate these ontologies in terms of cohesion. With a subsequent step of statistical analysis, the authors indicate that their cohesion complexity measure holds for the analysis of ontologies. Furthermore, their calculated values of their metrics for the ontologies match with the opinions of the invited experts.

Zhang, Li, and Tan [184] do also see and value the relatedness of software complexity and ontology complexity in a way that they issue that software design features experiences about the relation between software complexity and software quality. Both of these metrics symbolise the difficulty for humans to apply and make use of the given software, which manifests in the activities of coding, debugging, testing, and modifying. This comparison is also drawn for ontologies of the Semantic Web, as ontology complexity and quality is also seen closely related to one another. Therefore the same claim is made, that the more complex an ontology is, the harder it is for humans to use, which is also illustrated in the processes of developing, using, and modifying the ontology.

To assess the complexity or quality of a Semantic Web ontology, Zhang, Li, and Tan constitute a list of eight metrics, which originate from two levels: **ontology-level metrics** and **class-level metrics**, which either focus on the entire ontology or rather a single class of the ontology. The ontology-level metrics are the following:

SIZE OF VOCABULARY Amount of classes and relationships/properties defined in the ontology. The more of these two factors, the more complex is the ontology.

EDGE NODE RATION The ratio between nodes and edges of the ontology. More edges in relation to nodes makes the ontology more complex.

TREE IMPURITY This metric measures how far the inheritance structure of an ontology deviates from a tree-structure. If an ontology is less tree-like, it is estimated to be more complex.

ENTROPY OF GRAPH The probability of a node having i edges. A lower entropy symbolises that the ontology has more structural patterns and is therefore less complex.

In contrast, the class-level metrics are seen to have effects on other entities of an ontology, when and if they are altered, and therefore

contribute on this level to the ontology complexity. They are the following:

NUMBER OF CHILDREN The number of direct subclasses of a node. A higher value in the number of children makes the ontology more broad and thereby more complex.

DEPTH OF INHERITANCE The length of the inheritance path of subclass relationships from a root class to the currently considered class. The longer the inheritance paths are, the deeper and therefore more complex is the ontology.

CLASS IN-DEGREE The number of edges that point towards a currently regarded class node. The higher this metric is, the more classes are dependant on the given class and the more complex is the ontology.

CLASS OUT-DEGREE The number of edges that point away from a currently regarded class node. The higher this metric is, the more classes are dependant on the given class and the more complex is the ontology.

As an analytical evaluation, Zhang, Li, and Tan enlist the properties defined by Weyuker [177] and evaluate their metrics against these properties. The results indicate, that the metrics suffice all but two of the nine defined properties, which makes them generally usable in practice. Then, they also show an empirical evaluation, in which the authors gathered several ontologies with a data collection tool that was also able to calculate the values for the defined metrics in order to have a discussion basis. With these results the authors claim that their metrics are able to differentiate between the various complexities of a set of analysed ontologies and therefore their metrics are usable to assess the complexity of a Semantic Web ontology.

DOMAIN MODEL COMPLEXITY AND AN OWN COMPLEXITY MEASUREMENT With the insights about both software and ontology complexity, it is reasonable to apply similar approaches for the evaluations explained in this work. The domain models of the Anno4j library constitute software code, but the software metrics shown above seem not applicable or useful, as domain models are built out of rather simple Java POJOs, which would not create meaningful values for metrics like the control or information flow. Nevertheless, the impact and underlying concept does carry over to the Semantic Web ontology complexities and therefore also for the domain model complexity.

The results and metrics defined by the related approaches of ontology complexity are very interesting and impactful for the research done in this thesis. It is however important to note that the related

works' perspective on the ontology complexity is different compared to what is needed for the domain model evaluations of this work: while the related approaches try to estimate an ontology's complexity and thereby its further implications with different parameters or metrics *after* an ontology has been created, the evaluations of this thesis require parameters that can be configured in order to create an ontology with representative domain model and its Anno4j Classes in the first place. These parameters are thereafter evaluated on their impact on creation times of domain model representatives.

Nevertheless, for the complexity of the Anno4j Classes and thereby the domain model, six different parameters, which are partially influenced by the insights gained from related work enlisted above, were identified that do have potential impact on the runtime of creating a respective instance. The benchmark will therefore relate back to the parameters and give insights and explanations about their interpretations. Out of the six parameters, four originate from common metadata modelling concepts that closely relate to the metrics of Zhang, Li, and Tan [184] (which in terms also suffices the claims made about the parameters of Weyuker [177]), while two are additions that have their source in the Anno4j library. The six parameters are the following:

DIRECT SUBCLASSES OR CHILDREN DEGREE The amount of classes that are direct subclasses of the given class and therefore inherit its behaviour.

DIRECT SUPERCLASSES OR PARENT DEGREE The amount of classes that the given class itself is a subclass from, which in terms defines the gathered inherited behaviour for the given class.

INHERITANCE DEPTH The amount of classes that implement a direct inheritance chain that leads to the given class.

AMOUNT OF RELATIONSHIPS AND PROPERTIES The number of relationships and properties that are defined for the given class.

DEFINED PARTIAL CLASSES The number of Partial Classes that are implemented to extend the functionality of the given class.

DEFINED METHODS IN PARTIAL CLASSES The number of methods defined in all supported Partial Classes for the given class.

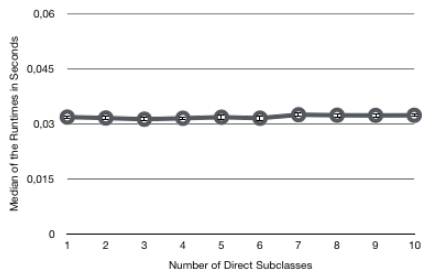
It is important to note that the benchmark will only consider the first creation of an instance of a given Anno4j Class, as the library will "remember" the proxy behaviour that it created with the initial creation. As the actual semantic content of the analysed ontologies in the benchmark is irrelevant, mocked ontologies were generated and varied in their respective parameter configuration. To obtain the domain model that can be utilised for the evaluation in Anno4j, its Generation Tool is utilised in order to automatically generate the different domain models.

BENCHMARK SETUP An initial parameter configuration was set, whose values were inferred from respective average values of vocabularies and ontologies that are currently present in the LOD. This approach created an initial setting for the to be instantiated Anno4j Class. An **inheritance chain** with depth two is assigned, while all of the Anno4j Classes had four different defined **properties/relationships** and one implementing **Partial Class** which also implemented the four methods of the Anno4j Class as **Partial Methods** respectively. In the first round of experiments, each parameter was varied on its own, while the other five parameters were fixed to the initial configuration. From this parameter configuration an RDFS schema was generated, which in terms was used with the Anno4j Generation Tool to generate the domain model. This model is then compiled with an OpenJDK Java-Compiler and packaged as a Java Archive. The Java file together with the benchmark code is loaded into a Java Virtual Machine JVM in order to initiate the benchmark.

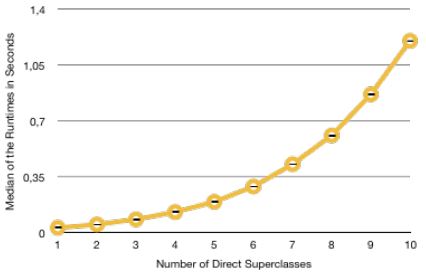
Therefore, in order to allow optimisations via the JIT-Compiler, 25 warmup iterations are done. Afterwards, the next 50 iterations are used in order to evaluate the runtime to create the respective instance of the focused Anno4j Class. After every iteration, a new Anno4j object is created to avoid existing proxies. When the cycle of 50 evaluated iterations is finished, the respectively current JVM is terminated and a new JVM is started for the same parameter configuration. This is done ten times overall for every configuration in order to randomise the influence of memory layouts, caching, and garbage collection. All of this results in 500 measures for every parameter configuration. The configuration of the test system that is used for the basic benchmarks is enlisted in Table 5 in the Appendix.

BENCHMARK RESULT DISCUSSION AND INTERPRETATION Figure 45 visualises the results for the benchmark described above in six sub-figures. Therefore, every figure illustrates the runtime evaluation when a given parameter is varied while the other five parameters are fixed to the values of the initial configuration. The results will be discussed by each single parameter. Additionally, their impact will be assessed and weighted against insights that have been gained by analysing 633 Linked Open Data Vocabularies.

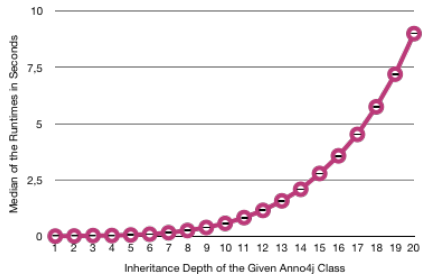
The first benchmark shown in Figure 45a visualises the impact on having multiple **subclasses** associated with the given to be produced Anno4j Class. Varying between one and ten associated `"rdfs:subClassOf"` relationships it can be seen, that this parameter does not have negative influence on the runtime, as the required time to create an instance of the Anno4j Class stays constant. It can therefore be inferred, that the amount of associated subclasses is irrelevant for the creation runtime.



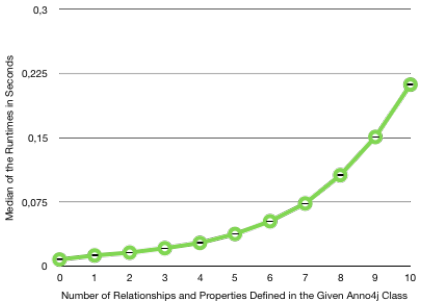
(a) Runtimes in Terms of Direct Subclasses.



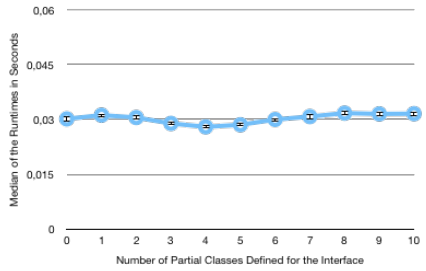
(b) Runtimes in Terms of Direct Superclasses.



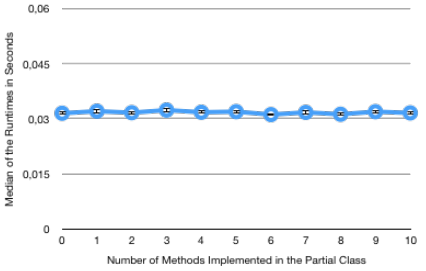
(c) Runtimes in Terms of Inheritance Depth.



(d) Runtimes in Terms of Relationships and Properties.



(e) Runtimes in Terms of Partial Classes.



(f) Runtimes in Terms of Partial Methods.

Figure 45: Benchmark Results for Single Variations of Ontology Parameters for the Anno4j Evaluation.

In contrast, it can be seen in Figure 45b that the opposite factor in the form of **direct superclasses** does have a negative impact on creation runtimes, as a polynomial growth can be assumed. In terms this raises the assumption that this circumstance imposes difficulties relating to the applicability of Anno4j and general ontologies, as branching and diverse inheritance hierarchies are often assumed. Nevertheless, the investigation of the 633 LOD vocabularies in terms of direct superclasses showed that 94,5% of the respectively contained classes only had three or less associated superclasses. At the same time, 76% only had exactly one superclass. Therefore this negative influence can generally be ignored in practical applications.

The same investigation does also show that **inheritance hierarchies** can go deep. In this regard, about one third of the considered classes of the 633 LOD vocabularies were incorporated as the lowest class in an inheritance chain with the length of five. This circumstance, in combination with the benchmark shown in Figure 45c, indicates a polynomial growth and therefore marks this parameter to be treated carefully. As an example, the numbers point out that an instance of an Anno4j Class with five superclass predecessors takes seven times as long as a Class without superclasses.

A similar behaviour can be seen in Figure 45d, which illustrates the runtimes with increasing amounts of **defined relationships and properties** for a given Anno4j Class. Here, one can also see and assume polynomial growth in terms of the runtimes. This circumstance is even aggravated by the fact, that inheritance hierarchies propagate their relationships and properties to their subclasses as well, so not only the directly implemented ones of the Anno4j Class itself do have impact here. The evaluation of the 633 LOD datasets showed that one fourth of the found classes do at least support four different properties or relationships. Therefore, this parameter requires careful attention in more complex ontologies.

The last parameters that have been considered by the benchmark originate from the Anno4j library, namely the possible additions of **Partial Classes** and respective **Methods** of those, which can be implemented next to a given Anno4j Class. The diagrams shown in Figure 45e and Figure 45f indicate, that both parameters do not have an impact on the runtimes for instantiating an instance of the considered Anno4j Class.

With the results of this section, it is possible to achieve domain knowledge about the parameters and the impact that they introduce to the instantiation of different Anno4j Classes. The benchmarks show that it can be problematic to work with the Anno4j library in conjunction with complex ontologies, especially if the ontology defines a deep and broad class hierarchy with many different relationships and/or properties assigned. Nevertheless, the Anno4j library is able to solve this circumstance by pre-generating the Proxy Classes for a given domain model, inducing a relatively fixed starting time interval, but at the same time reducing the initial creation of Anno4j Class instances to an imperceptible minimum.

6.2.2 *Runtime Experiments with Pre-Created Proxy Classes*

In order to give deeper insights into the problematic of runtimes for the initial creation of Anno4j Class instances described in Section 6.2.1, this section will highlight the reason behind the potentially problematic runtimes. Afterwards, the Anno4j solution to this is de-

scribed and a benchmark is conducted that shows the optimised efficiency when using the respective feature.

In order to narrow down the source of the issue, the tool VisualVM[3] is used in order to examine the factors that are involved in the process of creating an Anno4j Class instance. As an example, Figure 46 shows the breakdown of runtimes of processes that are involved in the initial creation of an instance of the "`crm:E21_Person`" class of the CIDOC CRM ontology (for further information about the CIDOC CRM see Section 6.3).

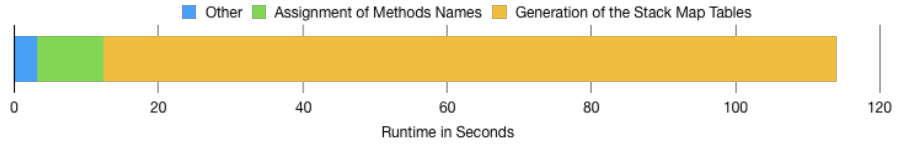


Figure 46: Breakdown of the Runtimes of Subtasks Necessary to Create the Initial Instance of an Anno4j Class Implementing the "`crm:E21_Person`" RDF Class of the CIDOC CRM.

Overall the instantiation of the person object took 113,9 seconds. In this time interval, 101,61 seconds were required by the "`MethodInfo.rebuildStackMapIf6`" method, which is part of the software library Javaassist[4]. This library is used by the underlying Alibaba and is applied for the generation of bytecode for the Proxy Class. It generates so called Stack Map Frames, which are stored in the *Stack Map Table SMT*. The SMT is a data structure that was introduced by Java 6, which is mandatory since Java 7.

It is used by the JVM in order to implement a quicker type checking. For every bytecode instruction that induces a hop or is the target of a hop at the point of compiling, a Stack Map Frame is inserted into the SMT. This frame specifies the types of the local variable and of the stack of operands to the point of time of the hop instruction. When a class is loaded, the bytecode instruction of nearby Stack Map Frames are read and their code effects in terms of the variable types and the stack are inferred. If an instruction is read whose associated Stack Map Frame is conflicting with the inferred types, then the verification fails and the class is not loaded. With this it is prevented that the JVM is compromised by the given code.

As Anno4j presumes Java 7, an SMT has to be created via Javaassist in order to ensure that the Proxy Classes that are generated via Alibaba can be read by the JVM. Therefore the types of local variables and the stack of operands have to be identified, and respective Stack Map Frames need to be created for the respective locations of the created bytecode. Checks are necessary for the type inferencing of Javaassist, if a primitive datatype or a null reference is present, or if the field has already been initialised. The profiling of the object

3 https://visualvm.github.io/ (last visited 03/12/2018)
4 http://jboss-javassist.github.io/javassist/ last visited (03/12/2018)

creation via VisualVM shows that numerous recursive calls are necessary for this in complex inheritance hierarchies, which at the end leads to the observed long runtimes.

As mentioned earlier, only the initial creation of an object instance is problematic in terms of runtimes, as Alibaba and therefore the Anno4j library is able to "remember" the bytecode that is created for the instantiated class. Therefore, after the first creation, further creations of the same Anno4j Class do not require the same timely effort, but are rather created with very low timing constraints. Nevertheless, although this does help with a running system, it can still be problematic if the domain model changes often or the system and its underlying JVM has to be restarted frequently, as the initial creation of Anno4j Class instances has to be redone with each slight change of the domain model.

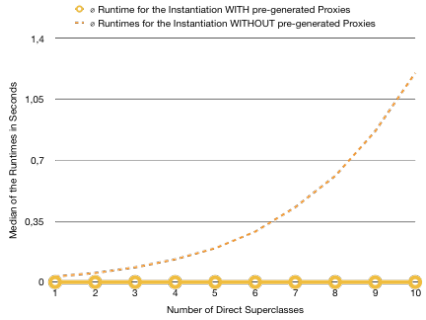
To counteract this circumstance, the created bytecode was examined and it turned out that for an unchanged domain model, it stayed the same for a given class. The created Proxy Class implements all methods of the given Anno4j Class and all its superclasses and their names are inferred from the names of the respective Anno4j Classes and their methods. Using this, the Anno4j library supports a command line based tool that can pre-create the Proxy Classes for a given domain model. The output are two JAR files, one of them containing the compiled Anno4j Classes of the supported domain model, the other containing the bytecode that is necessary for the generation of the Proxy Classes. Adding these two JARs to a Java project that intends to use the underlying domain model allows the creation process to use the contained Proxy Classes and therefore skip the timely effort to create respective bytecode on its own.

This command line tool itself needs some time in order to create its output. However, this process can be done initially by the developer before he deploys his application, therefore alleviating the actual user of the costly runtimes to create initial instances of the Anno4j Classes of the domain model. In combination with the insights gained from the parameter evaluations in Section 6.2.1, estimations can be done if the pre-generation step is useful or not for a given RDFS or OWL schema.

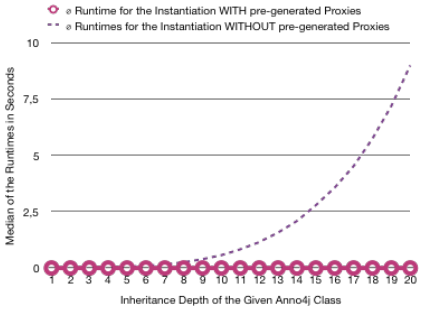
EVALUATION OF RUNTIMES WITH PRE-CREATED PROXY CLASSES

In order to show and evaluate the effectiveness and the applicability of the possibility of pre-creating the Proxy Classes for a given domain model, a benchmark was done with the same settings and measurements as the experiments shown in Section 6.2.1, with the only alteration of adding the created JAR files of the Anno4j command line tool to pre-create the Proxy Classes. The evaluations will only be done with the "problematic" ontology parameters of **subclasses**, **inheritance depth**, and **number of properties and relation-**

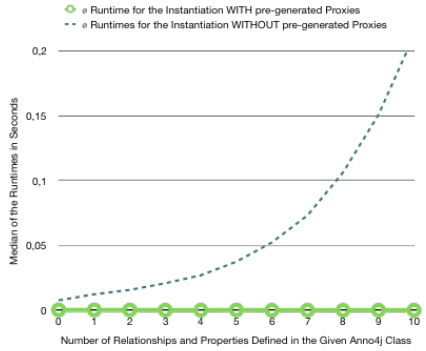
ships. Nevertheless, the runtime savings are applicable for the parameter of **subclasses**, **Partial Classes and methods** as well, but their constant runtime behaviour evaluated above is not considered as impactful for applications. The results of the benchmark are shown in Figure 47. In contrast to the actual evaluation numbers, the dotted lines indicate the runtimes without pre-created Proxy Classes.



(a) Runtimes in Terms of Direct Superclasses with Pre-Created Proxy Classes.



(b) Runtimes in Terms of Inheritance Depth with Pre-Created Proxy Classes.



(c) Runtimes in Terms of the Number of Relationships and Properties with Pre-Created Proxy Classes.

Figure 47: Benchmark Results for Problematic Parameters with Pre-Created Proxy Classes.

The results depicted in Figure 47 show that with pre-created Proxy Classes in place, the originally polynomial growing runtime numbers can be reduced to a constant behaviour. With the parameters of **superclasses** shown in Figure 47a, **inheritance depth** in Figure 47b, and the **number of relationships and properties** in Figure 47c, it can be seen that the runtimes can be reduced to an imperceptible minimum. Turning the results into numerical statements, the median of the runtimes for the superclasses had an constant average value of 56 µs with a standard deviation of 26 µs. For the inheritance depth an average value of 81 µs with standard deviation of 55 µs is calculated, while the properties/relationships benchmark showed values of 80 µs to 51 µs standard deviation.

Altogether, this benchmark shows that with the application of the Anno4j command line tool to pre-generate Proxy Classes for a given domain model allows convenient and efficient working with even large and complex ontologies or vocabularies. A one time process with the input of an RDFS or OWL schema needs to be conducted that creates two JAR files that support the domain model and the associated bytecode that is necessary for the initial creation of an instance of an Anno4j Class. Therefore Anno4j can apply a complex domain model while still keeping low runtimes that do not impede the applicability of users of respective applications.

6.2.3 *Runtime Experiments with Generated Anno4j Domain Models and Multiple Varying Parameters*

Prior evaluations allowed to gain some domain knowledge in terms of the defined ontology parameters, which is not only interesting for evaluation purposes, but these insights can and should also be applied when designing ontologies that are supposed to be used in real-world applications. It has been seen that **direct superclasses** (or **Parent Degree**), the **inheritance depth**, as well as the **number of defined relationships and properties** of a given Anno4j Class are impactful in terms of runtimes when creating an instance of this class. These runtimes are increased in polynomial fashion the higher the respective parameter is set. Defined **Partial Classes** with their **methods** and defined **direct subclasses** (or **Children Degree**) are considered as non-harmful for runtimes.

Nevertheless, although this domain knowledge helps to raise the awareness for parameter definition of ontologies, it is not thoroughly sufficient, as only single parameters are considered and altered. When used in combination (which is the general case), parameters could be dependant on each other, possibly having unexpected and grave impact on application runtimes. To also achieve awareness and insights for this kind of situation, this section enlists an evaluation that considers multiple varying ontology parameters.

EVALUATION SETUP AND APPROACH This evaluation considers the same ontology parameters that have been mentioned before, with the difference of varying multiple parameters in contrast to the evaluations described in Section 6.2.1 and Section 6.2.2. No pre-created Proxy Classes are in place in order to calculate the timely constraints for the initial creation of Anno4j Classes. A grid search-like approach (which is a concept mainly found in the fields of Statistics [164] [108] and Machine Learning [87] [159]) is applied that is restricted to check pairs of parameters for their allocation. It turned out quickly that a full grid search with parameter settings above the configurations found in general ontologies of the Semantic Web require heavy timely

workloads and therefore instantly qualify for the pre-generation of proxies, as evaluated in Section 6.2.2. However, the evaluation of parameter pairs delivers meaningful insights that further extend the gained ontology parameter knowledge from above evaluations.

Just like for the evaluations of single parameters, the same kind of workflow has been applied in order to generate numerical results. The setting of parameters is applied to generate a Semantic Web ontology, which is then used in the Anno4j Generation Tool to generate an Anno4j domain model. From that model a considered class is picked and a respective instance is created. The required runtime for its instantiation is measured in the evaluation. The technical benchmark setup in terms of warmup runs, real measured runs, etc. is exactly the same as described in Section 6.2.1. The most important difference is the hardware setting that is used. As this evaluation requires a lot more resources, it is conducted on a server, whose technical details are enlisted in Table 6 in the Appendix.

With the application of two varying parameters at a time, the results of this evaluation will be displayed in the form of heatmaps [179]. Heatmaps are two-dimensional matrices, whose axes will convey two respective parameter settings, while the values of the matrix itself as a third "axis" represent the calculated runtime values for the respective setting. These runtime results will be semantically and visually increased by the "heat" value of the map in order to better illustrate the gradient of the values, ranging from blackish colours for low values to whitish colours for high values.

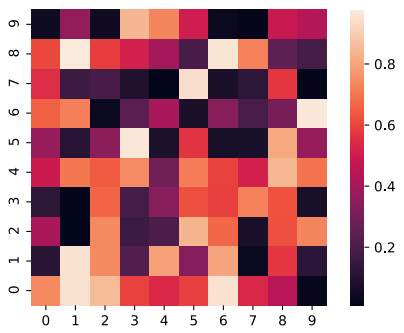
The combination of plotting the results as a heatmap and the domain knowledge that has been gained for single parameters - from now on called **soft** and **hard** parameters, depending on their constant or polynomial impact on application runtimes - , it is expected that the heatmap patterns will converge towards certain "pattern classes" in terms of a visual representation. The patterns that are identified are the following, which are also illustrated in Figure 48:

SCATTER PATTERN A scatter pattern (seen in Figure 48a) infers that the two currently considered parameters are not dependant on each other. This is most likely seen, when two **soft parameters** are utilised in the evaluation and therefore the calculated runtime values are roughly the same for every setting of the parameters. As a result and because the whole range of the heatmap is utilised, the created heatmap receives its scattered visual pattern of heat values.

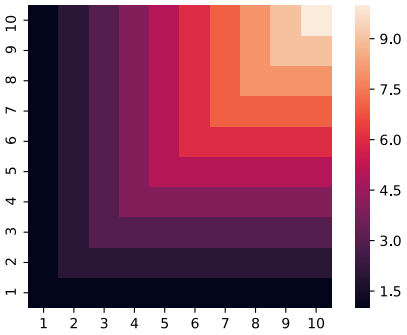
PEAK PATTERN The "opposite" to the scatter pattern is visualised with the peak pattern (seen in Figure 48b), which conveys a high dependance between the two considered parameters. The shape of the peak pattern orients its highest values towards the top right corner, which conveys that the combination of parame-

ters and therefore a higher setting for the parameters has grave impact on application runtimes. This observation can be seen with any combination of **soft or hard parameters**, as the recombined increase in runtimes is much higher in comparison than the influence of a single parameter by itself. This can be seen by the two dark edges of the heatmap pattern that are positioned directly next to the axes and a single whitish peak value in the top right corner.

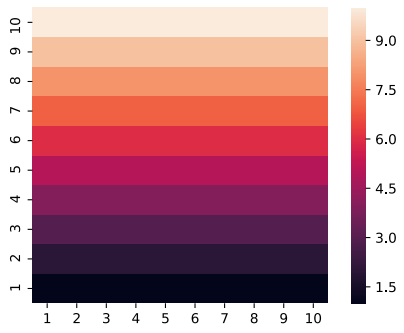
EDGE PATTERN The last type of heatmap pattern does not necessarily infer the existence or non-existence of dependance between the two considered parameters, but does symbolise that one parameter has more influence than the other in the given parameter combination. The increase of the one parameter does impact the overall combined runtimes, while the increase of the other does have a considerably low to no impact at all. This kind of pattern can be directed towards two directions, given by the two parameters, and therefore an increasing gradient of heat values can be seen towards the top or right edge of the heatmap pattern.



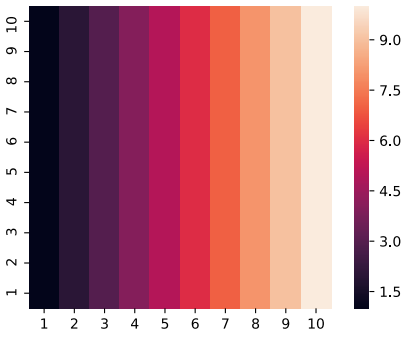
(a) Exemplary Scatter Pattern.



(b) Exemplary Peak Pattern.



(c) Exemplary Edge Pattern, Directed Towards the Parameter of the y Axis.



(d) Exemplary Edge Pattern, Directed Towards the Parameter of the x Axis.

Figure 48: Heatmap Plot Patterns with Exemplary Heat Values, Highlighting the Color Gradient or Scattered Distribution.

Inferring from the domain knowledge of the single parameter evaluation and the defined pattern types, an expectation can be done in terms of what kind of pattern the recombination of soft and hard parameters will create. These expectations can be seen in Figure 49.

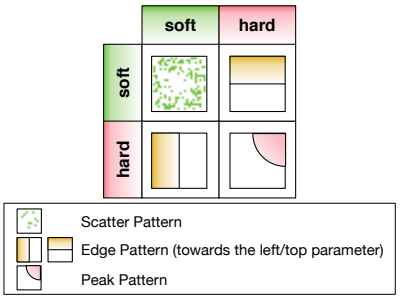


Figure 49: Expected Pattern Types Created by the Recombination of Soft and Hard Parameters.

The recombination of two **soft parameters**, as described above, does most likely lead to a scatter pattern of the heatmap, as both parameters on their own behalf do not increase overall runtimes regardless of their setting. When a **hard parameter** is combined with a **soft parameter**, edge patterns are expected that are oriented towards the hard parameter, as a polynomial increase in runtimes is introduced by the hard parameter on its own. The other soft parameter is in general considered to not be impactful in their recombination. The last possible combination of two **hard parameters** is expected to result in a peak pattern, as two already impactful parameters are combined, which eventually leads to a multiplied increase of overall runtimes.

EVALUATION OF RUNTIMES WITH VARYING PARAMETER PAIRS

In the following, the results conducted of the evaluation of ontology parameter pairs will be enlisted, described, and interpreted. Therefore, first a general table and some expected results will be shown. Unexpected results and met difficulties while doing the evaluation will be explained afterwards.

The concrete evaluation results can be seen in the heatmaps illustrated in Figure 65, Figure 66, and Figure 67 in the Appendix, while the calculated numbers (runtimes with mean and median, standard deviation, standard error, and 95% confidence interval) of the evaluations are enlisted in the tables reaching from Table 7 to Table 20 in the Appendix.

Figure 50 shows the recombination of all given parameters applying the measured results and assigns them the pattern types that have been described and defined above and in Figure 49.

In order to further explain the generated plots, three representative results that fulfil various pattern types are described here. Among those, Figure 51 shows the recombination of the **Children Degree** and **Partial Methods** ontology parameters. The plot illustrates a scat-

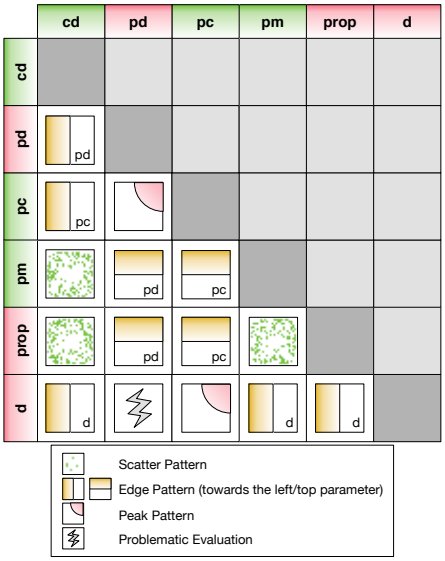


Figure 50: Pattern Allocation of Runtime Experiments with Varying Ontology Parameter Pairs.

ter pattern, issuing that the parameters are not dependant on each other and therefore do not affect overall runtimes when used with higher values and in combination. The creation runtimes take values in a small range from around 22.5 milliseconds to 28.5 milliseconds and are scattered throughout the whole parameter setting range. The explanation for this circumstance could be the fact that the parameters do not necessarily influence each other, as the Partial Methods defined for the given class are not necessarily passed to the subclasses, and that the parameters especially do not interfere with creation runtimes of a given class by being soft parameters.

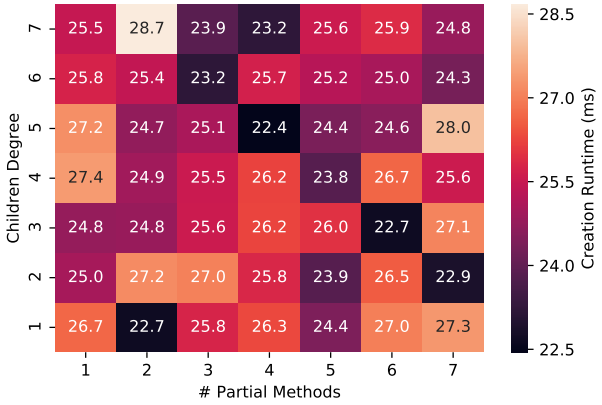


Figure 51: Exemplary Scatter Pattern Result for the Combination of the **Child Degree** and **Partial Methods** Parameters.

In contrast, Figure 52 shows a representative edge pattern that is derived from the combination of the **Children Degree** and **Inheri-**

tance Depth parameters and therefore a combination of a hard and a soft parameter. The pattern is oriented towards the Inheritance Depth parameter, which illustrates the fact that the parameters do not have a dependance on each other, as the increase of the Children Degree parameter does not have any impact on creation runtimes, while the inheritance depth parameter does show its previously known behaviour of having an increasing impact with an increasing value setting. Creation runtimes start with around 26 milliseconds for the lower settings of the Inheritance Depth parameter, and increase to around 60 milliseconds for the highest setting.

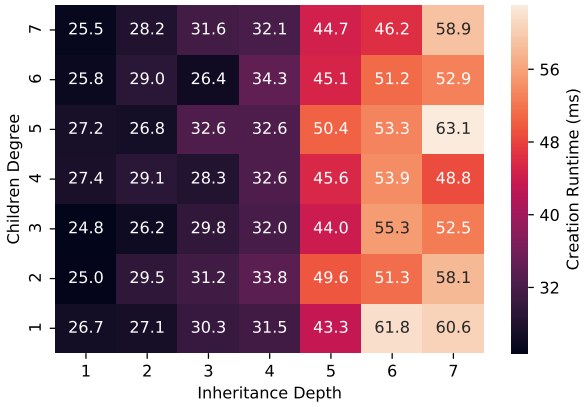


Figure 52: Exemplary Edge Pattern Result for the Combination of the **Children Degree** and **Inheritance Depth** Parameters.

The last category of patters is supported by the peak pattern, which for example is created by the combination of the **Parent Degree** and **Partial Classes** parameters. Their respective evaluation plot is shown in Figure 53, illustrating the characteristical shape with the highest heat value located in the top right corner of the heatmap. This result is unexpected, and will be discussed further in the next paragraph, as this combination features a soft and hard parameter. The creation runtime values have their lowest values around 30 milliseconds and increase drastically up to a maximum of 6 seconds with higher parameter configurations. This can be motivated by the fact that defined superclasses as well as defined Partial Classes for a given class require the Anno4j library to merge several behaviours of different natures together into one resulting behaviour, which imposes multiple time consuming merge processes.

UNEXPECTED ONTOLOGY PARAMETER PAIR EVALUATION RESULTS

Several unexpected results have been discovered with the evaluation of paired parameters. The first peculiarity is illustrated in Figure 54 and concerns the **Partial Classes** parameter. The respective interesting pattern entries are highlighted in the matrix.

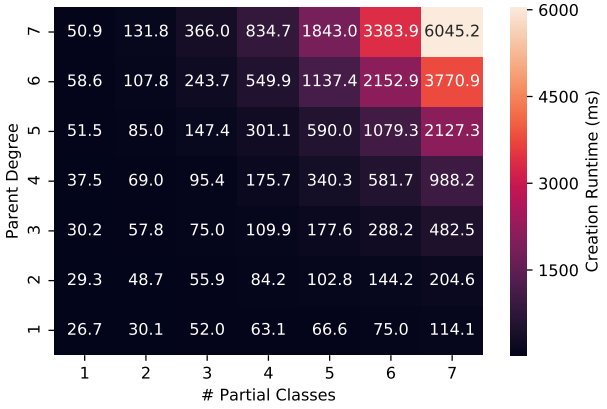


Figure 53: Exemplary Peak Pattern Result for the Combination of the **Parent Degree** and **Partial Classes** Parameters.

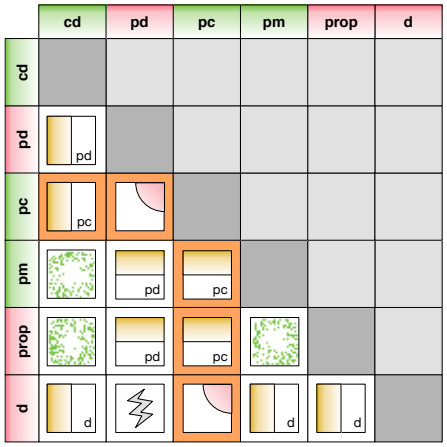


Figure 54: Highlighted Evaluation Results for the Combination with the **Partial Class** Parameter.

Prior evaluations resulted in the assumption that this parameter can be considered as a **soft** parameter and therefore uncritical in terms of application runtimes. In combination with the other five parameters however, it turned out that this is only true when regarding the parameter "in isolation". When combined with other soft parameters, earlier assumptions lead to the expectation that scatter patterns generally are the result of the combination. In combination with a hard parameter, an edge pattern should emerge that is tilted towards that respective hard parameter. Figure 54 however shows that this is not the case. In combination with soft parameters, the **Partial Class** parameter takes on the "dominant" role of the pair and therefore creates an edge pattern towards its axis. When combined with hard parameters, a peak pattern is created. Both of these observations lead to the conclusion, that the Partial Class parameter cannot be seen as soft parameter, because in real-world ontologies it is generally not ap-

plied alone. This result is especially interesting for the contributions of this thesis, as the considered parameter is a concept that is introduced by the Anno4j library, and therefore needs proper attention and utilisation.

The second unexpected observation applies to the **Property and Relationship count** for a given class. The preceding single evaluations and therefore gained domain knowledge showed that this parameter has an increasing impact on creation runtimes, the more properties and relationships are implemented, exposing this parameter to be assessed as a **hard** parameter. The pair evaluations however show the opposite, which is illustrated in Figure 55 and the respectively highlighted result patterns of the heatmaps.

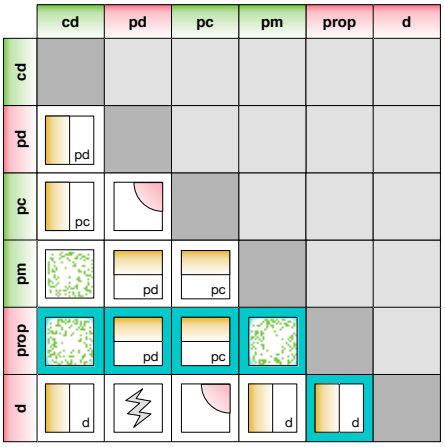


Figure 55: Highlighted Evaluation Results for the Combinations with the Property/Relationship Count.

In contrast to the Partial Classes parameter which turned from a soft to a hard parameter, the opposite holds true for the **Property and Relationship** count. When combined with soft parameters such as the Children Degree, the Partial Classes, or the Partial Methods, the currently considered parameter induces a scatter pattern, which symbolises non-dependance between the combined parameters. This is unexpected, as a hard parameter should always impose at least an edge pattern towards its values. The combination with other hard parameters, namely Parent Degree and Inheritance Depth, the Property and Relationship count acts as "passive" part rather than introducing expected impact on runtime values, resulting in an edge pattern towards the combination partner. All of this leads to the observation that the Property and Relationship count only acts as a hard parameter on its own, but does not induce any increases in combined creation runtimes when applied in combination with other parameters.

The last observation of the ontology parameter pair evaluation concerns the combination of the **Parent Degree** and **Inheritance Depth** parameters. No evaluation results could be computed for this combination with higher parameter values, as computation times took

several days for a single run of the evaluation, from which every setting requires 75 (25 warmup and 50 benchmark) in order to achieve meaningful results. The evaluation was cancelled for this configuration after the currently active twelve threads each on their own took over 180 hours and were still not finished. This unexpected and problematic behaviour starts at a parameter configuration of 4 and 4. Because of this, no result table nor a heatmap could be generated for this combination.

The reason for this behaviour has yet not been fully discovered, but from a technical point of view, a high amount of superclasses in combination with a long inheritance chain delivers a complex ontology - which is both deep and broad - and the Anno4j library needs to merge down many different and possibly overlapping behaviours into one place. It is also possible that better merging algorithms need to be implemented in order to enhance this process. This however opens an interesting point for future work.

Nevertheless, this circumstance does not render the Anno4j library to be unusable with such ontologies, as the pre-generation of proxies (as shown in Section 4.6) opens the possibility to pre-create the domain model for the given complex ontology and then instantiate respective instances in no time. Furthermore, as described in Section 6.2.1, commonly found ontologies rarely exceed the setting of having four or more superclasses with an inheritance depth of above four siblings per class.

6.2.4 *Conclusion of the Anno4j Evaluations*

In summary, several evaluations have been conducted that target the creation of Anno4j Class instances which fulfil certain ontology criteria in terms of defined parameters. This is combined with the Anno4j Generation Tool, which is used in order to generate the respective Anno4j domain model out of a mocked ontology that implements the respectively considered ontology parameters.

These evaluations have shown that there are parameters that need to be observed when an ontology is designed that is supposed to be worked with Anno4j in the first place. The gained insights however do also stick to general ontologies and suggest a complexity that can be assigned to the ontology. Furthermore, attention should be paid when using certain parameters in combination and excessive configuration. The conducted experiments allow to gain insights and awareness for these kinds of problematic combinations, as they might have impact in application runtimes which potentially surpass commonly accepted waiting times.

With the evaluated runtimes an estimation can be made about the complexity of a given ontology, therefore determining if a preparatory step that pre-generates necessary proxies for the Anno4j library

is reasonable. The respective generation requires some time upfront before an application can go live, but it reduces instantiation runtimes to a minimum and hence allows an uncomplicated implementation of said application.

6.3 VISIT - A CULTURAL HERITAGE USE CASE FOR THE ORDFM LIBRARY ANNO4J

The last few sections have shown various ways of evaluating the work and the contributions enlisted in this thesis. This was initiated by an enlistment of related work and approaches that analysed the ORdfM concept from a more general point of view, while the subsequent computations evaluated novel features implemented by the Anno4j library.

Another potential way of evaluating the work done in this thesis is showing the appliance and therefore the impact that the contributions have on follow-up projects and other implementations in the field of the Semantic Web. It has already been mentioned that the Anno4j library has been mentioned in related work by Suhrbier et al. [161] and Prabhune et al. [130] as well as the W3C Web Annotations Group, who enlist Anno4j as a reference implementation[5].

In terms of scientific impact, next to these mentions, another research project at the University of Passau called **ViSIT - Virtuelle Verbund-Systeme und Informationstechnologien für die touristische Erschließung von kulturellem Erbe**[6] (which is roughly translated to "Virtual Network-Systems and Information Technologies for the Touristic Exploitation of Cultural Heritage") poses a perfect use case for the Anno4j library and has therefore picked up its application.

The basic premise of the ViSIT Project is to virtually connect multiple museums to share their cultural content in the form of exhibition samples. In general, these museums are expected to tell a story in an exhibition about roughly the same topic. As they however do usually have different samples, each of them can only back up their respective exhibition with the samples they have on-site, the rest of the story must be covered by text only, which is of course possible, but not desirable, as the samples produce a much more colourful and impactful representation of the historical facts. Figure 56 illustrates this situation. The encapsulating grey area symbolises the whole historical content there is to tell about a given Person "x". Puzzle pieces in the figure depict an exhibit sample that has been found, which backs up certain facts about the history of that given person. Two museums, "a" painted on the left and "b" on the right, are in possession of some of the samples, namely "1" through "6". Exhibitions held in one of

5 https://www.w3.org/annotation/wiki/Implementations (last visited 03/12/2018)
6 http://www.phil.uni-passau.de/en/dh/projekte/visit/ (last visited 03/12/2018)

these two respective museums can now only display those samples they have on-site, allowing them to only cover about below half of the history that is potentially been told about "x" (incorporating the fact that there are two other examples, which are in neither of the houses).

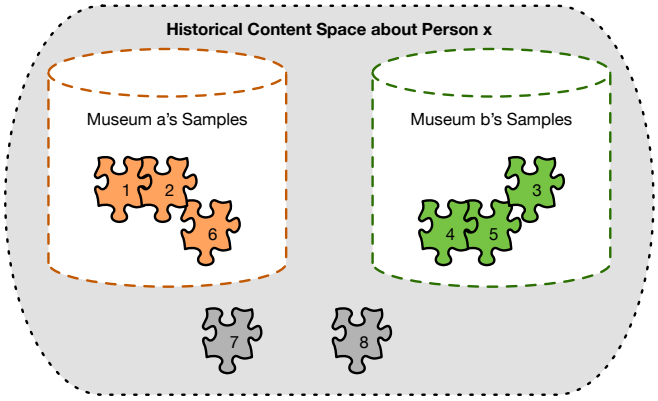


Figure 56: Exemplary ViSIT Scenario with Two Museums "a" and "b", both Issuing an Exhibition about Person "x".

This circumstance is targeted by the ViSIT Project. Various digital representations of the museums' samples are produced, which can be shared between the participating museums. These representations can then be applied in different multimedia installations in order to fill up the "bare" textual representations with not real samples by itself, but rather digital replacements. Representations and illustrations like 3D models, RTI scans, or interactive maps and videos can be incorporated in modern applications like operable computers, tablet stations, or virtual- and augmented reality, in order to fill in the gaps of a given museums's inventory while also creating a technically modern and interactive experience for the visitor. Figure 57 illustrates the scenario of two museums telling roughly the same historic narration about the same topic, person "x", with the utilisation of the ViSIT application. While Figure 56 showed two museums that were not able to synchronise exhibit samples on the fly in order to allow both houses to tell the narration, ViSIT facilitates this circumstance. By creating the above mentioned digital representations of the samples, a museum can request these digital samples and incorporate them into their narration. This does not induce a house to lend some of the samples, as the digital samples can fill up missing pieces of the narration. In Figure 57, museum "a" on the left side complements its narration of own exhibits "1", "2", and "6", with digital representations of the samples "3" through "5", attained by utilising the ViSIT infrastructure.

With the contributions of this thesis in mind, it quickly gets obvious how the resulting implementations can benefit the use case of

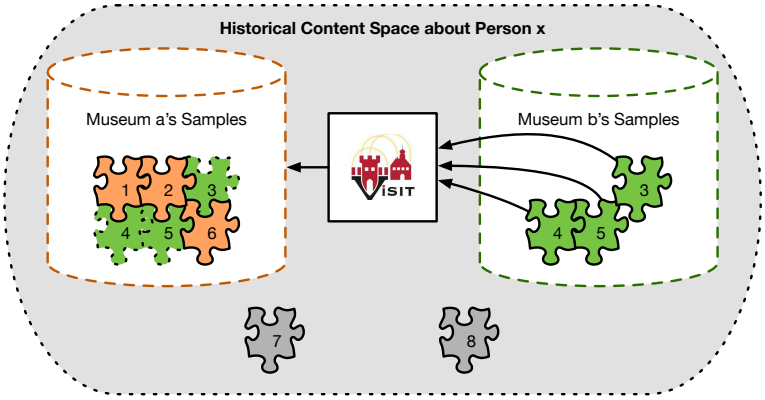


Figure 57: Exemplary ViSIT Scenario Applying the ViSIT Infrastructure to Digitally "Lend" Foreign Samples in Museum "a".

the ViSIT Project. In almost the same manner as it is for multimedia data, descriptive metadata plays an important role in the overall scenario described above. In order to correctly communicate and then synchronise the available samples on demand, they require definitive metadata that allows to search and respectively query them.

One of the best-known and prevailing ontologies for the domain of cultural heritage is the **CIDOC Conceptual Reference Model (CRM)** [58], which has been referred to already throughout this work. The CIDOC CRM is classified as being a complex ontology, originating from the fact that it covers the whole domain of cultural heritage by not being specialised to smaller topic domains, as well as simply its large amount of available RDF classes and their respective hierarchy, which are stocked by even more unique relationships and properties.

Their data model has received substantial development and effort and therefore has now gathered 10 years of experience. The CIDOC CRM is implemented by the CIDOC Documentation Standards Working Group[7] and CIDOC CRM SIG[8], which are both working groups of CIDOC[9]. It has been accepted as working draft by ISO/TC46/SC4[10] in 2000, and as official standard since 2006 [91], revised in 2014 [92]. Its purpose is to serve as an ontology to which other (if not every) other possibly heterogenous and specialised cultural heritage information can be mapped to in order to eventually represent a commonly understandable knowledge base of the whole domain knowledge, therefore aligning with the general idea of the Semantic Web and Linked Data described throughout this work.

7 http://network.icom.museum/cidoc/working-groups/overview/ (last visited 03/12/2018)
8 http://network.icom.museum/cidoc/working-groups/crm-special-interest-group/ (last visited 03/12/2018)
9 http://network.icom.museum/cidoc/ (last visited 03/12/2018)
10 https://www.iso.org/committee/48798.html (last visited 03/12/2018)

With its current major version 6.2[11] released in May 2015 (and the from there adapted latest minor version 6.2.3[12]), the model incapsulates 89 classes and 149 unique relationships and properties (which in most cases do also have the direct inverse relationship that does semantically convey the opposite of the given relationship) that are aligned in an RDF class-hierarchy with multiple in- and outwards branching "`rdfs:subClassOf`" relationships.

As indicated before, although the model is very dynamic and multifunctional and therefore allows to cover any topic or domain in the cultural heritage domain, issues arise quickly when it comes to actually implementing the model. Even with smaller use cases, the resulting RDF graph grows quickly, as the recombination of many different nodes and relationships is necessary. Additionally, when considering RDF classes that are positioned deeper in the hierarchy, developers are confronted with a plethora of various relationships to implement, even aggravated by the fact that some semantic relationships between two RDF instances are not directly modelled via a single relationship, but rather a path over possibly several nodes. This poses the problem of semantically finding the fitting RDF constructs to convey one's use case, paired with the problem of actually serialising the eventual RDF triples.

Anno4j supports this kind of scenario in various ways. The first, and probably most impactful benefit, is supported by the generation capabilities of the library. As stated above, the CIDOC CRM is composed of many RDF classes and relationships, making it hard to write instance data of the respective model. With many different naming conventions, also incorporating numbers to enumerate their objects in ascending order, errors are bound to happen when the RDF data creation is done by hand. Therefore technical support with the Anno4j functionality is desirable.

The Anno4j Generation Tool takes the RDFS schema of the CIDOC CRM (or a representative OWL version of the Erlangen CRM/OWL[13]) which results in the generation of Anno4j Java Classes that represent the CIDOC CRM domain model. With that, developers and users are able to create triples that are conform to the CIDOC CRM model by using simple Java Classes, rather than writing the triples by hand. The direct benefits of this method are more convenient handling, automated structural integrity of the produced data as the Java Classes enforce a structure, as well as a more convenient creation of the domain knowledge, as the Java Classes present only those relationships that are directly supported by the given RDF class or inherited from their superclasses. This way, the user does not have to determine the allowed relationships and properties on his own.

11 http://www.cidoc-crm.org/Version/version-6.2 (last visited 03/12/2018)
12 http://www.cidoc-crm.org/Version/version-6.2.3 (last visited 03/12/2018)
13 http://erlangen-crm.org/ (last visited 03/12/2018)

A second benefit from applying Anno4j to create cultural heritage data lies in the validation of created data. As indicated above, a first step of structural validation happens as the Anno4j Classes dictate a direct structure on the created triples. This however only affects the validity in terms of correct RDF class instances and their relationships and properties amongst each other, while further validation criteria, like multiplicities or inverse relationships that have been seen in Section 4.5.2 and Section 4.5.3, are not considered. As described throughout this work, data validity is especially important in the Semantic Web and Multimedia domains, and the domain of cultural heritage is no exception. Stating cultural and historical facts is especially dependant on correct data, things like dates need to be as correct as possible. From a technical point of view, the circumstance of the CIDOC CRM and data validity is even aggravated by the fact that one cannot assume the same general user as was done throughout this work: the "non-Semantic Web expert" is estimated to have only few to no knowledge about Semantic Web technologies, but is expected to have programming knowledge of object-oriented concepts, Java at best, or HTTP and REST.

Nevertheless, Anno4j alleviates this circumstance. Its validation features can be included to check for the adopted validation requirements of the CIDOC CRM in order to enforce their fulfilment *before* data is persisted. Otherwise the process of data creation is prohibited and the user can be prompted to correct the respectively incorrect data. The technical knowledge barrier in this case cannot be eliminated entirely, but the supported technological access methods allow for more convenient ways of constructing input interfaces that encapsulate the Anno4j functionality in order to also eventually create valid CIDOC CRM data.

These facts show that the utilisation of the Anno4j library can help developing applications in the domain of cultural heritage. In addition, the arguments enlisted here support the claims of the contributions done throughout this work. The following section will show an outlook of a feature called "Semantic Zoom", which allows to convert the complex CIDOC CRM metadata into a "fuzzier" but more simple domain model using the SWRL features of the Anno4j library shortly described in Section 4.8.

A "SEMANTIC ZOOM" FEATURE TO INTRODUCE AN EASIER MODEL ABSTRACTION LAYER FOR THE CIDOC CRM With the CIDOC CRM being one of the most common metadata models in the domain of cultural heritage, it is reasonable for a project like ViSIT to mainly stick with that respective metadata structure. However, as indicated in this section, the CIDOC CRM allows both very thorough and fine-grained possibilities to dynamically model a respective use case, but at the same time it is a very complex model and sometimes compli-

cated to apply. This holds true especially when the use case itself is very shallow. To counteract this, an idea is envisioned that allows to abstract from the complex syntax of the CIDOC CRM while at the same time conveying roughly the same semantics with the loss of only some information.

The **Semantic Zoom** is a feature that can be implemented using the SWRL features of Anno4j (see Section 4.8). With this feature it is supposed to be possible to have two models, the CIDOC CRM itself and a much easier, condensed, semantically similar model, which are able to be converted to one another on the fly. This is done by supporting SWRL rules that transform CIDOC CRM triples or even whole paths into the easier model and vice versa. Figure 58 shows an illustrative example.

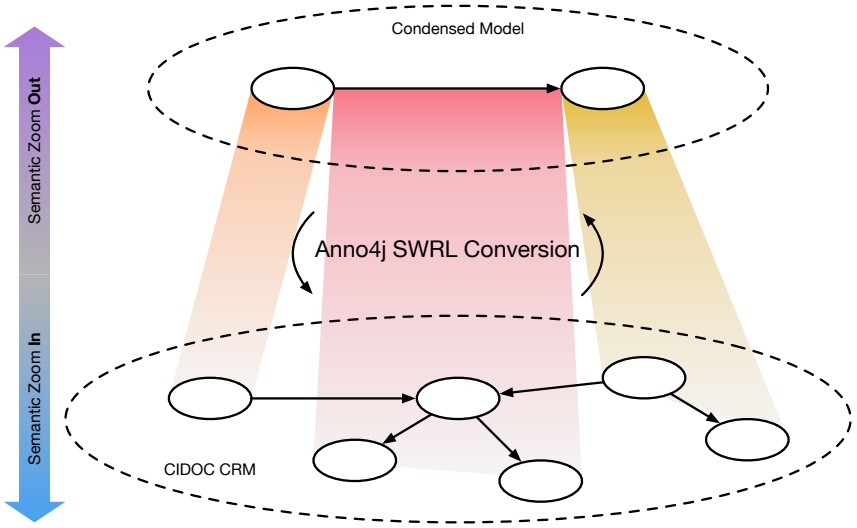


Figure 58: Illustration of the Semantic Zoom Feature Possible with Anno4j.

There are two different metadata domains visualised in Figure 58: on the bottom half there are instance nodes depicted that originate from the CIDOC CRM, while the top half represents two instance nodes of a generic model that implements the same semantics as the CIDOC CRM. With the envisioned Semantic Zoom mechanic by applying defined SWRL rules via Anno4j, the automated conversion translates instance data of the one domain model to the other, and vice versa, depending on which data is created initially. This is depicted by the transitions between the two domain layers. Therefore, a Semantic Zoom **Out** happens when one examines data in the form of the top layer, while a possible Semantic Zoom **In** happens when more detailed data in the form of the CIDOC CRM is to be regarded.

The benefits of this implementation are as follows. Firstly, the SWRL rules implemented in the Anno4j library can be run automatically. This enables the database to hold two graphs that are semantically

equal but in two different models. Therefore, access to the underlying semantics can be accessed from both ways. CIDOC CRM experts can still write CIDOC triples as they are familiar with, while non-CIDOC CRM experts can write and utilise the easier model and still require (roughly) the same semantical information. For some applications it might also be easier to apply the easier model, for example if the overall semantical relationships between the core information carriers are to be presented on a web page or similar forms.

Secondly, with the process of data being created in the two-fold way described above, the resulting data always keeps the claim of being directly conform to the CIDOC CRM. With an application like this one and in the world of the Semantic Web, this is always a positive claim as aligning your results to a well-known ontology or vocabulary is useful, as it invites others to use the respective data much more easily and conveniently.

However, as this envisioned idea still represents a conceptual construct, there are still shortcomings or factors that need further planning and effort in order to produce a thorough feature. The SWRL feature of the Anno4j library has been described in the outlook section of Chapter 4, but a first working version has been implemented in the most current version of Anno4j. However, it has not yet been tested in a full stack use case. Furthermore, this feature falls into the same drawback as they exist with the generation of the Anno4j Classes out of schemata when it comes to actually defining the SWRL rules, especially in the case of a complex metadata model. Just as with implementing Anno4j Classes by hand, the definition of the SWRL rules by hand can become cumbersome and error-prone. A solution to streamline this process should be devised in order to follow the overall claim of the Anno4j library and therefore support a more convenient approach.

A slight "inconvenience" exists in terms of possible loss of information via the implemented conversion feature. When mapping the CIDOC CRM to a less complex data model, it is clear that not all information can be transformed, as there would not be a simplification possible otherwise. Nevertheless, as described above, this is not only a drawback but also a beneficial factor, as the easier domain model can serve other purposes much more conveniently and non-CIDOC CRM experts can use the data much more easily. Additionally, the CIDOC CRM data is never lost.

CONCLUSION OF THE EVALUATION CHAPTER In summary this chapter evaluates and confirms the applicability of the technical contribution of this work, namely the Anno4j library. It is started by a related work section that enlists related approaches that evaluate the overall ORdfM concept from a general point of view. These results show, that the concept is deemed very useful in Semantic Web appli-

cations and that there are no technical shortcomings that arise from the incorporation of the respective abstraction layer, but rather many advantageous factors. All of these implications can be adopted for the Anno4j library and therefore its core functionality is verified qualitatively.

This overall evaluation is then followed by own implemented evaluations and benchmarks in order to extend the findings of the related approaches. The benchmark applies combined functionality of the Anno4j library, as it takes a (mocked) metadata model schema as input for the Anno4j Generation Tool in order to generate a domain model and then persist respective information. Different parameters of the schema were identified that implement the complexity of the model on the one hand, and on the other hand these parameters do have different timely impacts on the creation runtimes of the persistence process, which in terms is evaluated by the benchmark. With a baseline setting for the parameters that have been inferred to an average value from publicly available LOD datasets, in the first evaluations single parameters have been varied in order to determine their impact. Among these six parameters, three were identified as non-impactful as their variation did not show any impact on the runtime of initial creations. In contrast, the other three, namely **superclasses**, **inheritance depth**, and the **amount of properties and relationships**, showed a polynomial growth in terms of runtimes for increasing values and therefore are considered as potentially problematic.

Next to the numbers indicating the runtimes, as a side effect, an assessment of the ontology structure has been gained that allows to evaluate the complexity of a given RDFS or OWL ontology in order to estimate their required runtimes. This is very useful, as another contribution of this work lies in a command line-based tool of the Anno4j library that is able to pre-generate the Proxy Classes that are required by the library in order to create instances of Anno4j Classes. With the approach of the first benchmark, the "classic" procedure of the library is to create those Proxy Classes, given an ontology, with the initial creation of a given Anno4j Class. The command line tool is able to pre-generate these Proxy Classes in order to support them to a given Java process. Doing so, the time intensive process of creating the Proxy Classes is skipped, reducing the overall runtime to an imperceptible minimum. This process has been evaluated in a second benchmark, showing that the initially problematic parameter impacts can be reduced to a mean value of only some microseconds.

A follow-up evaluation uses the domain knowledge gained from the first evaluation and computes paired parameter configurations, using the same ontology parameters as above. Most of the combinations had expected results and therefore, by being dependant or non-dependant on each other, have multiplicative or no added impact on application and instance creation runtimes. There were however

some unexpected results between several parameter combinations, which showed that the prior gained knowledge about the parameters has to be refined and adapted when the parameters are used in conjunction with each other. The **relationship and property count** itself was estimated to be a so-called **hard** parameter, but turned out to be non-dependant on each other parameter and therefore applies its runtime increases only in isolation. In contrast, the **Partial Classes** parameter, which was assumed to be a **soft** parameter, has heavy impact in the combined runtimes with other parameters and therefore needs particular attendance, especially as it is a newly added Anno4j feature to the ORdfM concept. Lastly, the combination of high **inheritance depth** with many **super- or parent classes** turned out to be non-computable, which in terms raises interesting topics to be researched in future work. This gained knowledge about the ontology parameters allows to gain good estimations whether a pre-generation step of the Anno4j library is reasonable, as it can tune down application and creation runtimes to a minimum, regardless of the ontology parameter setting.

Overall, after the concept and different possibilities of the Anno4j library have been highlighted throughout this work, this chapter confirmed the applicability and the robustness of the library. This claim is backed up by the application of the library in the real-world project MICO, which served as further technical embedding and supports a combination point with the MICO Metadata Model. To extend the real-world application, this chapter did also describe the ViSIT project as another use case for the concept and implementation of Anno4j.

6.4 RECAPITULATION OF POSED RESEARCH QUESTIONS

After all contributions and the work done for this thesis has been documented throughout prior chapters and sections, this section will recapitulate the research questions that have been formulated at the very beginning. Thereby, an assessment of the fulfilment of their task and therefore the anticipated goal of the research questions will be formulated.

1. How to express, create, and especially recombine the created results of various heterogenous metadata producers?

The envisioned Semantic Web use case, which was introduced in Section 1.1.1, demanded the recombination of various information pieces originating from different participating information peers. As these peers in general are not known nor interconnected, a common data model is not applied, which induces that respective information pieces need costly and sometimes complicated ad hoc interpretation. To support this scenario, a metadata model is needed that

supports the expression, creation, and especially recombination of various metadata inputs.

A baseline to do this has been found in the W3C Web Annotation Data Model WADM. This ontology proved to be well suited to represent so-called Web Annotations, information units that further describe another item.

The use case from above was then further enhanced by a MICO Multimedia use case, which can be seen as a specialisation of the general Semantic Web use case. The Web Annotations of the WADM could conveniently be transferred to the Multimedia scenario, however this interpretation posed further requirements to the metadata model, as provenance, better recombination possibilities, data quality and validity, as well as traceability are important to suffice the overall demand of the scenario and consecutively also the research question.

A solution to this is proposed with the MICO Metadata Model MMM, which is a modular, extensible, and expressive metadata ontology for the Multimedia domain, which is used and applied in the MICO use case. As this use case is a specialisation, the model can be converted to an overall Semantic Web case without further effort.

The MMM was evaluated with requirements checks from two different perspectives: some were inferred from related work and target overall Web-conformance, while own defined requirements were inferred from the two use case scenarios mentioned above.

For most of the requirements, the MMM is well suitable and supports a good solution to the issues, and it therefore poses an answer to the research question. New and potentially unknown metadata producers can easily be integrated into the MMM, as it supports well-defined interfaces that can be implemented in the model and then be shared with other participating peers. Nevertheless, some of the requirements arise further issues that are beyond the possibilities of a data model. The modelling of metadata treated in this research question is seen as a part of the overall Semantic Web Technologies, and the following research questions focus on the topic of enhancing these technologies and thereby proposing solutions to said issues.

2. Are there ways to lower the initial barrier that is posed by Semantic Web Technologies, which manifests mainly in the fields of creation, processing, and requesting of the respective semantic data?

The mentioned barrier in this question is expressed in three pillars: creation, processing, and requesting of metadata. By enhancing a single one of the pillars, the overall barrier gets reduced. Therefore it is reasonable to consider all three of them by itself. The answer to this question in this work is seen in the implemented Object-RDF-Mapping Anno4j, which allows users to access the data present in the Semantic Web by using Java rather than the actual Semantic Web Technologies.

In terms of the creation of RDF data, the process can fully be mapped to only using Java technologies. With only a little knowledge about Semantic Web internals like URIs and how they interplay in an RDF graph, a domain model can conveniently be instantiated by implementing basic Java classes, the so-called Anno4j Classes, which resemble building plans that are provided with getter and setter pairs that represent the relationships and properties which can be associated with the currently considered RDF Class. By plain Java instantiation of those Anno4j Classes, to which the user is assumed to be familiar with, the respectively created information is translated to RDF triples on-the-fly and persisted in the particular connected database.

The processing and working with the Semantic Web data does also get facilitated as well as improved. Both the results of creating and querying a Semantic Web database are present to the user as common Java objects, therefore allowing to apply any Java features to the data, for example code instructions or even testing capabilities. This opens a wide range of accessibility to the Semantic Web data without imposing an up-front initial barrier to the user. As mentioned above with the creation of the data, its modification is also persisted on-the-fly by the Anno4j library.

Lastly, querying capabilities are also enhanced by using the Anno4j library. The main querying tool for the Semantic Web is the SPARQL querying language, which is similar to SQL from the relational world. This especially poses another, sometimes complex, technology that users have to learn when they want to take part in the Semantic Web. To counteract this, Anno4j allows for basic querying, which directly matches for RDF Classes, enabling the user to apply Java features to the returned set of objects. Also supported is more fine-tuned path-based querying, which is assumed to be easier to understand than the pattern-based SPARQL queries. Additionally, the path queries can be created modularly, adding single query requirements step by step which are eventually recombined, rather than creating one large query at once. This range of querying capabilities gives users the choice between different approaches, eventually reducing efforts that have to be spent in order to retrieve desired information from a Semantic Web database.

In summary, these three pillars show that it is possible to reduce or lower the initial barrier that is posed by Semantic Web Technologies with an implementation like Anno4j. By altering the underlying technologies of the processes that users have to apply, the access is eventually made easier for non-Semantic Web experts. Nevertheless, the barrier can only be lowered, but most likely not removed completely. This work however developed further ideas and approaches that might further increase this circumstance. These are mentioned in Section 4.8 and later in Section 7.2.

3. With the Anno4j library applied as use case:

3.1 Can Object-RDF-Mappings deal with complex domain models?

As it has been pointed out by related work [132], a common agreement exists that ORdfM implementations have issues when it comes to more complex domain models or ontologies that are to be used, as the general representative does not support any automatic features of incorporating the ontology, or generating the domain model vice versa. The only way is to create the domain model by hand, which is error-prone as well as time consuming. With the Anno4j Generation Tool, this thesis supports a technical solution to solve this circumstance. An RDFS or OWL schema can be inserted, triggering a process that automatically generates the domain model that can then be applied in the Anno4j library for further utilisation of the ontology via the library's features.

3.2 Is the application of complex domain models possible in an efficient way?

First evaluations of the Anno4j Generation Tool with complex ontologies have shown problematic performance, as merely the creation of a single instance of an Anno4j Class took several minutes of processing time. A subsequent analysis of the involved processes has shown that the merging of different behaviours for a single Anno4j Class takes close to 90% of time of the overall process. As a solution, the Generation Tool has been extended in order to be able to produce so called Proxy Classes, which contain the originally created behaviour. These can be supported to the overall Anno4j project up-front, which reduces subsequent instantiation processes to an imperceptible minimum. With only a considerable timing constraint that can be done before an application is deployed, this feature enables Anno4j to efficiently deal with complex domain models. The evaluations conducted throughout Section 6.2 as well as the ViSIT project as a real-world use case, which utilises the CIDOC CRM ontology, give quantitative proof of this claim.

3.3 What can be done with Object-RDF-Mappings in order to (better) support data consistency?

Throughout this work it has been mentioned that issues in terms of contained data exist with Semantic Web databases, and that these need definite attention to make the respective database useable in a way that was envisioned by the initial Semantic Web idea. These issues mostly manifest in terms of data quality, as many databases contain faulty or

partly missing data and oftentimes lack descriptive information about its intended usage.

This circumstance is alleviated at some occasions by the utilisation of the Anno4j library. Simply the application of the structure of Java code and their classes for the corresponding domain model already induces a basic way of consistency, as the data that is to be created can only follow the structure that is dictated by these Java and Anno4j Classes. For example, assigned IRIs as well as RDF Class allocations are fixed, and relationships or properties can only be assigned to other specifically defined node types or restricted basic datatypes respectively.

Next to this, several validity requirements can be introduced in RDFS and especially OWL ontologies. These are also defined as RDF triples and are persisted in the same database. However, oftentimes these rules are not followed, as the input data is not necessarily tested for these requirements prior to its injection in every database. The Anno4j library can solve this circumstance, as the process of persisting data can be preceded by a validation check, which examines if the to be persisted data is conform to defined validity requirements that are present in the database. If these are not met, no data is persisted at all and the user is given feedback about the reason. The definition of the requirements can also be done conveniently in Anno4j by adding so-called Schema Annotations, which are added directly to the Anno4j Classes.

These enhancements to the ORdfM concept can therefore increase data quality and thereby increase the overall data consistency.

3.4 Can the application of Object-RDF-Mappings be enhanced by allowing different access technologies?

While the introduction of more access possibilities to the database correlates with the same disadvantages as seen with complex domain models - error-proneness as well as a time consuming process - it can open the database for users that are not familiar with Semantic Web Technologies via other standards and protocols.

For Anno4j, the expansion of access technologies has been done using the HTTP protocol, aligning to a RESTful API structure oriented towards the Spring framework. Its structured approach can be combined with the Anno4j Classes, therefore not imposing many changes to the basic data entities. The necessary remaining additions to the domain model can either be written by hand if they need to be

fine-tuned, while a basic HTTP behaviour can be automatically generated via the Anno4j Generation Tool, which is improved by such a functionality.

Altogether, this HTTP functionality allows to lift Anno4j and the ORdfM concept to another level of accessibility, as a generated RESTful API can be used in other scenarios as the "basic" application of the mapping. Additional access possibilities are thereby opened for users, as they can then choose between classic SPARQL, standard Anno4j access, or HTTP requests to interact with a Semantic Web database.

Part V

SUMMARY AND CONCLUSION

7

RÉSUMÉ

7.1 CONCLUSION

The setting for this thesis is given by the **Semantic Web**, an "extension" to the universal Web with the focus on lifting a "Web of Documents" to a Web that elevates its contained data and information in a way that computers are then able to "understand" it and hence have increased capabilities of working with it. The classic Web of Documents was mainly designed for humans to interact, so therefore the information is represented in a way for humans to process visually, while computers can only use this information to a small degree, if at all.

To counteract this circumstance, the vision of the Semantic Web coined by Tim Berners-Lee in 1998 supports guidelines, best practices, and concepts that should be met and applied by developers and applications that want to take part in the Semantic Web and its corresponding **Web of Data**. Since its first days, the Web of Data is constantly growing and exhibits exponential growth, constituting 1139 distinct datasets and knowledge bases that are openly accessible.

It has been seen that the uptake of this idea is mainly met in the scientific sector, while the economic world or non-scientific institutions rarely apply Semantic Web technologies in order to participate in the Web of Data and therefore they can not profit from its benefits. However, particularly this low uptake in other sectors does contradict the original idea of the Semantic Web. With every contribution to the overall knowledge base, every participating peer could benefit. From simple things like having sheerly more data to more complex features like peer reviews of inserted data or the interlinking of the given data which allows for the discovery of more continuative data, the beneficial factors are numerous.

As a motivational standpoint for this work, several blocking factors that impede the concept of the Semantic Web or rather the uptake of the Semantic Web technologies have been assumed. Oftentimes, *insufficient knowledge* about those respective technologies hinder developers to lift their application to be conform to the Semantic Web, as they are in many cases familiar with other concepts and do not want to invest the timely effort to familiarise themselves with the respective languages that would be required. Secondly, the Semantic Web technologies have *formal underpinnings* and therefore incorporate formal semantics as well as logics that need to be applied in order to make full use of given Semantic Web data. Lastly, existing knowledge bases

are *hard to access technologically* or manifest *inconsistent data*, eventually hampering or preventing actual use of the contained data at all. In the end, individual factors or a combination of them might lead to the same result: developers that are discouraged to use the Semantic Web.

The bits and pieces of information that represent the basic building blocks in the Web of Data are called **metadata**, descriptive information about data that characterises it in a semantical and high-level way. A very prominent domain of metadata is the **multimedia** domain, which is also very dominant in the Web world of today, and therefore one can see a potentially close relation between multimedia and the Semantic Web. The main items of this domain are constituted by videos, audio, pictures, and text. For those the metadata plays an important role as the describing information is of essential importance when it comes to making the multimedia item available and searchable online.

This work showed an approach to tackle the issues of the Semantic Web described above in a multimedia-oriented use case. Therefore, Chapter 2 introduced the concept of the Semantic Web as a whole by defining its vision and requirements, also delimiting it against other concepts that are similar or closely related. Then, the **basics of Semantic Web technologies** have been explained which are constituted by the *Resource Description Framework RDF* and its de-facto standard querying language *SPARQL*, the *SPARQL Protocol and RDF Query Language*. These have then been accompanied by the definition of *RDF ontologies and vocabularies*, accumulations of defining schema descriptions that instruct a structural format on RDF metadata allowing for the specification of a certain domain. Aligning to these schemata when producing metadata allows direct understanding and interpretation on the one hand. On the other hand querying gets facilitated, as queries can be aligned structurally towards the defined schemata in order to request desired information. The *Resource Description Framework Schema RDFS* as well as the more detailed *Web Ontology Language OWL* stand for the commonly known representatives of ontology modelling languages.

With the use case of describing multimedia metadata in mind, after an enlistment of related work in the fields of both multimedia metadata and efforts to lift metadata description to the Semantic Web, which yielded fundamental insights, Chapter 3 describes the contribution comprised of the design and implementation of a comprehensive metadata model for the multimedia domain. Therefore the W3C's *Web Annotation Data Model WADM* with its Web Annotations is implemented as a baseline, constituting a modular structure for the information pieces of the describing metadata. Applying this structure, a syntactical distinction is made between the actual semantical content of a piece of information, and its target, which represents the mul-

timedia item or subpart of it that is actually referenced by the Web Annotation. This allows a fine-grained modelling of metadata that enables rich and comprehensive querying capabilities. This annotation structure has been adapted and extended by the **MICO Metadata Model MMM**, lifting the concept of the Web Annotation to a multimedia use case. This is done by aligning the described distinction of content and target to multimedia requirements and the addition of modules for composition and provenance information between multiple annotation results. This eventually allows to recombine various metadata sources for one given multimedia input, which are also comprised by a provenance chain that enables traceability between finer-grained metadata processes. In the end, a thorough, modularly structured, and rich metadata background can be produced, allowing for comprehensive querying capabilities which thus enables an enhanced applicability and usability of the initial multimedia item.

The concept of metadata modelling as well as such comprehensive metadata models like the MICO Metadata Model are necessary in order to support domain-oriented guidelines to users for both creating and consuming metadata, eventually increasing both the metadata's quality as well as its applicability. As indicated above, in the current state of the Semantic Web, many different datasets originating from various domains exist and can freely be used. However, as also described in Chapter 1 and the introduction of this section, metadata models on their own seem not sufficient enough to attract many users, especially in the economical sector, regardless of its potential. In this thesis, it has been assumed that the technical barrier in order to adopt the Semantic Web technologies is to high and therefore potential developers are discouraged. As a solution to these problems, a contribution of this thesis exists in the implementation and description of the **Anno4j** library. It follows the *ORdfM concept* which is similar to the ORM concept of the relational world, with the difference that it deals with RDF data rather than relational data. By using the library it is possible to convert RDF data into Java objects, a concept that the assumed developer is familiar with, and vice versa. Therefore it can be abstracted from the interactions and communication processes to deal with the RDF data, enabling the user to interact with metadata while only needing low pre-knowledge of the Semantic Web technologies. These interactions are further diversified by the library by allowing different querying mechanisms which differ in the access technology as well as their "difficulty" to use. Additional features like the **code generation** and the **validation features** further increase the overall usability of the library, decrease development efforts, and enable database requirements like the well-known ACID criteria on the level of the abstraction layer.

To bring all the above together and at the same time show a practical application of the concepts and contributions, a third contribution

exists in the incorporation of the MMM and the Anno4j library in the **MICO Platform**. This platform is a multimedia analysis framework that anticipates the registration of so called (multimedia) extractors, autonomously working processes that can register themselves at the platform, which are then put together in workflow chains in order to analyse input multimedia step by step, eventually creating metadata results. With the propagation of produced results back to the platform, a combined metadata background is generated for the analysed multimedia items. All of this aligns with the multimedia metadata domain described above, so the MMM poses the metadata structure, while the Anno4j library is utilised as an abstraction layer in order to facilitate metadata communication, production, and consumption for the extractor developers. These developers are assumed as users who are not familiar with the Semantic Web technologies and who cannot invest effort into learning these technologies as well, as they have a complicated task at hand already. In addition to the technical implementation, a metadata workflow, called the **MICO Metadata Lifecycle**, was designed and also introduced to the Platform implementation. This lifecycle, and metadata lifecycles in general, allow a more detailed view on the core steps that metadata undertakes from production over adjustment up to the point of metadata consumption, enabling an even better approach for the overall metadata background of the initial multimedia item.

Last but not least, Chapter 6 enlists various experiments together with evaluations in order to support technical backing for the claims made for the ORdfM concept and hence Anno4j throughout the work. This is started by a related work section that highlights evaluations of external research, which verifies the applicability of the ORdfM concept from a general standpoint. Runtime evaluations show that the incorporation of the abstraction layer does not induce unbearable timing constraint, while other results confirm a definitive beneficial influence in development effectivity when using the ORdfM concept. As the Anno4j library therefore can be seen as backed up from an overall standpoint, these related work experiments are then complemented by own evaluations which target the new features that are introduced to the Anno4j library, which are not present in related approaches. Conclusions of evaluations are drawn in terms of which structural features of a given RDF/OWL schema are runtime-critical for the Anno4j Generation Tool in order to be able to make decisions if a generation step is reasonable or not. Initial runtime issues were detected when working with the tool and complex ontologies, but at the same time a solution is introduced to the library that allows convenient work even with these complex ontologies. Therefore, with only an initial and timely fixed effort for the generation of domain models, the Anno4j library can be used with any size of schemata without any runtime difficulties. With the ViSIT Project, another real

world use case is presented that poses another application of another domain, the cultural heritage, that benefits from the application of the Anno4j library.

The contributions in this work show an approach to bring Semantic Web technologies closer to people who are not familiar with them, eventually increasing the range of the Semantic Web. With this, a large community could not just utilise but also contribute to a common knowledge base which directly illustrates the original vision of the Semantic Web. The assumed issues that build up the initial barrier can be solved with the proposed approaches. Insufficient knowledge is reduced, as developers interact with familiar concepts rather than Semantic Web technologies. This argument does also hold true for the formal underpinnings, from which our approach can abstract from. Last but not least, inconsistent data is prohibited by validation features of Anno4j, while the technological access is diversified via different access technologies from which the user can choose from. Eventually, this approach was embedded into the multimedia domain, implementing an own metadata model to represent jointly produced metadata backgrounds for input multimedia items at the MICO Platform. Every contribution was confined with and at the same time learned from related work and approaches. In addition, some of the chapters did also enlist possible extensions that form an outlook and further improvements that exceed the limitations of this work. Added with some further ideas, these were the following.

7.2 FUTURE WORK AND OUTLOOK

The MMM already features a modular and extensible structure to incorporate new kinds of metadata, although a direct extension for existing metadata standards, like the MPEG-7 structure, would be very useful in order to allow users an out of the box utilisation of the model. Nevertheless, especially with the combination of the MMM and its application in the Anno4j library, respective implementations could enhance the applicability of already existing metadata at the point of time when the metadata is consumed in order to play out corresponding multimedia items. At the moment, the user is prompted to apply own strategies to consume the metadata. With a more (semi-) automated procedure to utilise the information of the MMM, the consumption step could be lifted to even better applicability for multimedia files.

Envisioned additions to the Anno4j library have been described in more detail in Section 4.8. These included the so-called *Object Queries*, a way to query RDF data via Anno4j by issuing Java objects rather than queries, the *SWRL Extension*, a way to decouple domain modelling from general development in Anno4j, and a *recommendation* feature with corresponding MMM implementation, allowing to find

similar entities in an RDF graph, which would be especially useful in a domain like the MICO Project.

Another feature for Anno4j that originated from the envisioned MICO+ idea and the extended schema functionality, is the reversed direction that the Generation Tool fulfils: generate the RDFS/OWL schema given an implemented domain model in Anno4j. This would close the circle between RDF data and Java objects from an abstract standpoint and would further increase the convenience of implementing new applications and especially improve the possibility of updating already existing applications. Next to this feature, an improved MICO Platform, which is called the MICO+ Platform, would greatly benefit from the additional features of Anno4j in version 2.4, namely the schema generation, validation features, as well as the RESTful implementation of Anno4j.

The experiments and evaluations that have been shown throughout Section 6.2 have shown that there is further effort needed in terms of some of the identified ontology parameters. Especially the combination between inheritance depth and the associated superclasses of a given Anno4j Class turned out to be problematic to compute when they are set to a value of four or higher. Although this is rarely met in common ontologies, this opens interesting questions as to why this combination leads to timely problematic computations of the library. Furthermore, the new introduced feature of Partial Classes and therefore also the respective ontology parameter proved to be non-harmful in isolation, but assumes more costly factors when combined with other ontology parameters. This also opens possibilities for further future work, as it is a novel feature to the ORdfM concept introduced by Anno4j.

Part VI

APPENDIX

A

APPENDIX

A.1 THE MICO PROJECT

The following sections enlist information about the overall MICO project in Section A.1.1 and one of its core components, the MICO Broker, in Section A.1.2.

A.1.1 *Background of the MICO Project*

Most of the contributions of this thesis are established in the **Media in Context MICO** Project[1] [2] [2], which is an FP7[3] research and innovation project that lasted from 2013 to 2016. Its main objective is to support federated analysis capabilities for multimedia objects, eventually creating metadata that greatly increases the object's usability. Figure 59 visualises the core workflow.

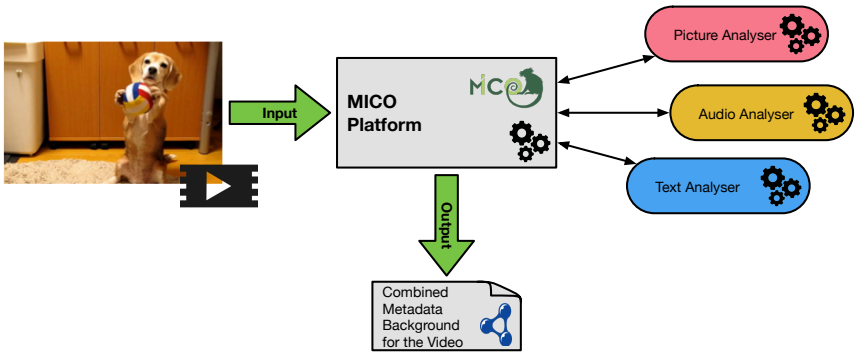


Figure 59: The Core Workflow of the MICO Platform.

The MICO Platform constitutes the centrepiece of the whole implementation of the project and is developed as a standalone software. Autonomously working multimedia extractors can register at the platform instance and will be called by the platform with respective input to analyse. As the context is given by multimedia, the input is mainly constituted by various multimedia types like video, audio, images, and text, but own potential high-level input can also be processed as long as the extractors are designed to work with it. As a result, the MICO Platform will combine all the generated metadata that is delivered by the single extractors in order to combine it into a metadata

1 https://www.salzburgresearch.at/projekt/mico/ (last visited 03/12/2018)
2 http://mico-project.eu/ (last visited 03/12/2018)
3 https://ec.europa.eu/research/fp7/index_en.cfm (last visited 03/12/2018)

background for the multimedia file. Extractors can also be combined and work in so called pipelines, propagating intermediary results between extractors to benefit from results that have been produced by prior extractors. With the combined metadata it is envisioned to enhance the applicability of the initial multimedia file in thorough and rich fashion, eventually even discovering hidden semantics, applications, and usage scenarios that had not been discovered yet.

The intention of the example use case shown in Figure 59 is to determine if a given video file contains the concept of a dog in it and therefore features a video eventually showing a dog as input for the MICO Platform. Three different (very generic and high-level) extractors analyse different parts of the video. The *Picture Analyser* processes single images or frames to determine if a dog-like shape occurs during the playtime, the *Audio Analyser* processes the audio stream of the video file in order to potentially detect barking noises, while the *Text Analyser* can filter through textual information like the title, potential comments, or descriptions, in order to find the dog concept or something similar in the textual representations. Every extractor will then provide a confidence about its result. While every result by itself might only give a vague estimation about the video containing a dog, the combined result of all three extractors contains way more information. Also, all produced results are presented uniformly and combined at the same place, allowing for more comprehensive interpretation and application.

The project altogether featured several technical work packages that were interconnected and based on one another. These packages dealt with the following topics:

MULTIMEDIA EXTRACTORS AND ORCHESTRATION This work package dealt with the development and implementation of the core workers in the MICO workflow: the (multimedia) extractors. The various multimedia formats required different implementation efforts and there were also new additions to the extractor domain in the form of animal detectors. Additionally, the difficult task of orchestrating these autonomous extractors in pipelines was also part of this work package.

CROSS-MULTIMEDIA METADATA MODELING All the implemented extractors produced a plethora of different varying metadata outputs. The main goal of this work package was the development of a dynamic multimedia metadata model that is able to comprise all the extractors' final and intermediary results in modular and structured fashion to enable rich and comprehensive querying capabilities. Next to this fact, provenance information and the recombination of produced results were envisioned. After the metadata model was devised it also turned out that ef-

forts were necessary to support both the steps of creating and consuming said metadata model.

CROSS-MEDIA QUERYING With the extractors and all their diverse results combined in the metadata, this work package aimed for the implementation of comprehensive querying mechanisms and possibilities to maximise the value and information that can be gained from the produced metadata backgrounds in order to increase the applicability of the input multimedia data.

CROSS-MEDIA RECOMMENDATION Last but not least, with all the technical foundations of the former work packages, this work package devised recommendation strategies that apply the information gained from prior steps in order to compare and eventually recommend the analysed multimedia files between one another. In addition, the recommendation is not only restricted to single media types, but can also produce cross-media recommendations. Doing so, the analysis of a picture can lead to the recommendation of a video with the same semantic content.

From the descriptions of the work packages it can be seen that the contributions of this thesis mainly target the Cross-Multimedia Metadata Modeling work package. However the described results do have impact on the other work packages as well, which is apparent throughout this thesis.

A.1.2 *Development Cycle of the MICO Broker*

As with other technical components like Anno4j and the MMM, the MICO Broker [4] started in a functionally rudimentary version, but has been extended and improved over the course of the MICO Project. The explanations of the extractors and their interplay with the MICO Platform were based on the simpler and earlier implementation, called *"Broker v1"*, as it is sufficient and more to the point for the content of this work. However, for a well rounded explanation of the MICO Platform, *"Broker v2"* and partly *"Broker v3"* will also be recapped here.

Although already robust and stable, the initial implementation of the broker had several shortcomings that required the implementation of the more thorough following version. This was induced especially by the fact that "manual" creation of workflows or pipelines, in the sense described above in which simple Strings for input and output specification were matched, was not sufficient enough for the workflow-oriented character of the MICO Platform. In addition, using this kind of matching, often times unintended or useless pipelines were created, or other counterproductive shapes like loops could occur. This circumstance needs to be backed up by richer extractor

information in order to support more thorough pipeline matching. Next to this, things like missing test potential, the support of extractor modes (for more differentiated application of the same extractor type, already discussed in Section 3.3.5), better workflow functionality in terms of error handling, workflow status, and progress tracking or logging, were necessary for a full-stack usable implementation of the MICO Broker.

The "simple" extractor definition was extended to have, next to still incorporating a textual representation of its mime types for input and output files, a *syntactic type* and a *semantic type* for input and output. The former gives a more detailed description about the file type that is either produced or required, and refers to an RDF class of the MMMterms ontology [20], which contains predefined result metadata constructs of the MICO use case for metadata input and output, and to a Dublin Core type [34] for binary data. The semantic type description gives more subjective semantical information about the extractor in combination with its use case scenario and is generally assigned by hand. This is mostly done in tag form and can be updated frequently, being especially useful for workflow administrators. While the mime and syntactic type should always be supported at deployment of the extractor, the semantic type can be addressed later on. All of this configuration information is also persisted in the triplestore of the respective MICO Platform instance, making it accessible for every component of the whole implementation if necessary.

To replace the manual matching of extractors in pipelines or workflows, a semi-automatic implementation of the orchestration unit supports a more thorough fashion to pipeline extractors. Various components implemented in v2 support this. The *"registration service"* accepts registering extractors and collects I/O configuration data that is delivered by respective extractors, which then can be applied by the *"workflow planner"* unit. The result is the semi-automatic structure which suggests matching extractors depending on their information. The user can then fine-tune these recommended pipelines or use them as they are. The *"item injector"* then analyses injected media files and addresses them to appropriate selected and loaded pipelines. Finally, the *"workflow executor"* supports correct forwarding of intermediate and final results to analysis processes with the remaining extractors.

A.2 FURTHER INFORMATIONS AND APPENDED DATA

The following list enumerates rules that are to be applied when RDF data is drawn as a graph in this work. They imply which RDF feature is illustrated at what place in the graph.

1. **rdf:Class** objects will be illustrated as rectangles with solid lines. Their respective IRIs will be printed inside using namespace prefixing.
2. **Instance nodes** as well as **blank nodes** are drawn as ellipses with solid lines. For the normal instances, IRIs are printed inside with namespace prefixing. If necessary for the respective example, blank nodes will receive a local IRI, otherwise the ellipses will be blank.
3. **Literals** are illustrated as rounded rectangles with solid lines. Their values are printed inside, most times their datatype will be omitted.
4. **rdf:Property** objects are drawn as solid arrows from subject to object. Their IRIs are displayed on top of the arrow with the application of namespace prefixing.
5. **Subgraphs** or **named graphs** (as described after this disclaimer) are displayed as rounded rectangles with dashes lines that are spanned around those nodes and edges that are included in the given named graph. The IRI of the subgraph is written somewhere inside of this rectangle, also using namespace prefixing.
6. Whenever a node or edge is drawn with **dashed lines** rather than normal solid lines, the circumstance of omitting further graph elements is intended. This is done for reasons of higher understandability and straightforwardness.
7. **Colours** may be used in different occasions of the graph, but they do not have any semantic meaning. They are used to increase the readability of the given graph by for example grouping several elements of the same colour or confining elements from one another.

List 1: Set of Applied Rules for RDF Graph Illustrations.

Table 3: RDF Ontologies and Vocabularies with Their Respective Namespace IRI and its RDF Prefix Used in This Work

Specification Name	Prefix	Namespace
Web Annotation Data Model [143]	oa	http://www.w3.org/ns/oa#
MICO Metadata Model [23]	mmm	http://www.mico-project.eu/ns/mmm/2.0/schema#
MICO Metadata Model Terms [20]	mmm-terms	http://www.mico-project.eu/ns/mmmterms/2.0/schema#
Representing Content in RDF [100]	cnt	http://www.w3.org/2011/content#
Dublin Core Elements [34]	dc	http://purl.org/dc/elements/1.1/
Dublin Core Terms	dcterms	http://purl.org/dc/terms/
Dublin Core Types	dctypes	http://purl.org/dc/dcmitype/
Friend-of-a-Friend Vocabulary [38]	foaf	http://xmlns.com/foaf/0.1/
Provenance Ontology [140]	prov	http://www.w3.org/ns/prov#
Resource Description Framework [151]	rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
RDF Schema [72]	rdfs	http://www.w3.org/2000/01/rdf-schema#
RDF Review Vocabulary http://vocab.org/review/terms.html	ref	http://purl.org/stuff/rev#

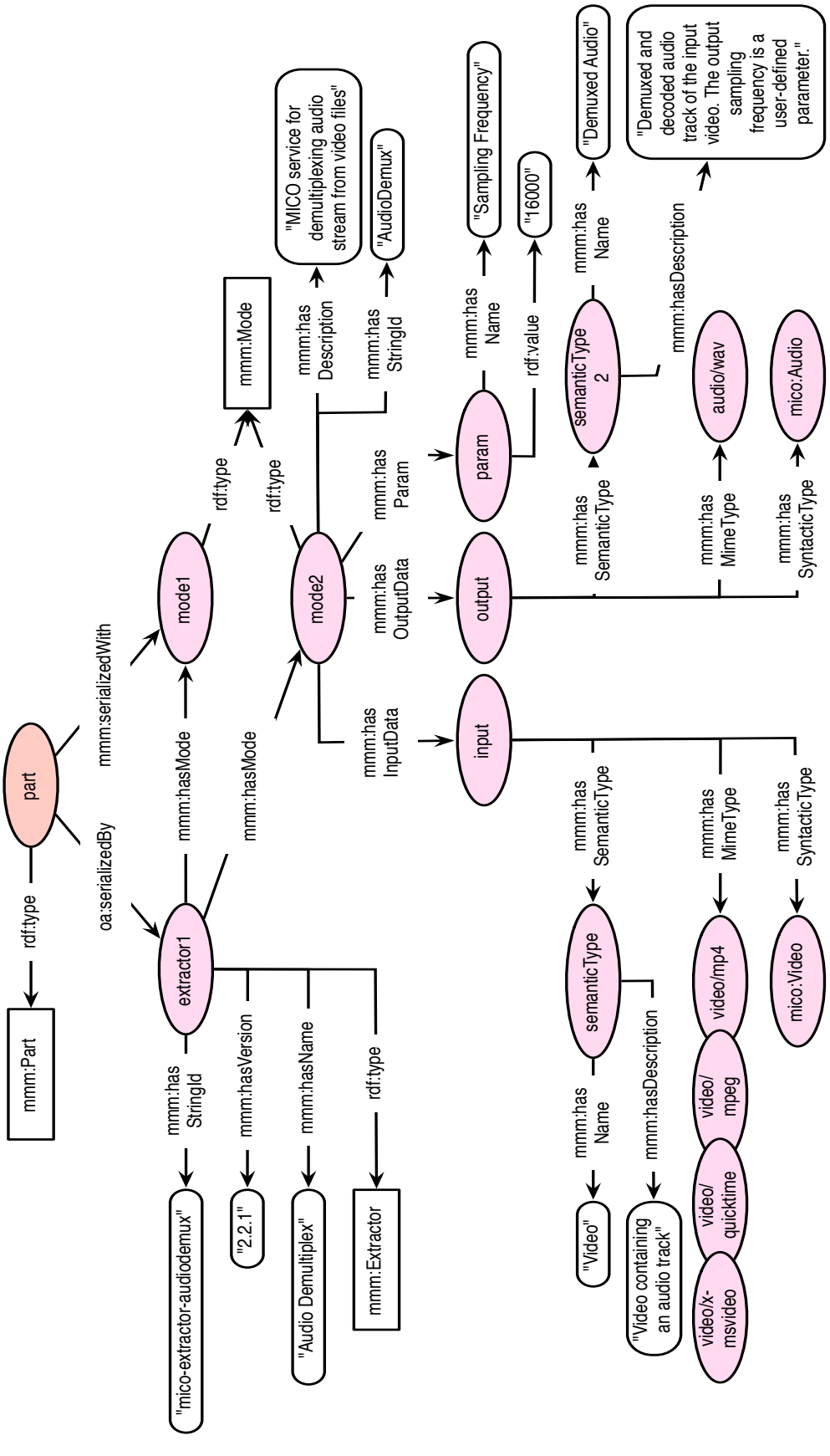


Figure 60: Exemplary Extractor Model for a MICO Audio-Demux Extractor, Leaving Out Some Side Information for the Sake of Clarity.

```
Anno4j anno4j = new Anno4j();

ItemMMM item = anno4j.createObject(ItemMMM.class);

AssetMMM asset = anno4j.createObject(AssetMMM.class);
asset.setFormat("image/jpg");
asset.setLocation("http://example.org/assets/pandaPicture.jpg");
item.setAsset(asset);

// Create Colorlayout Part
PartMMM colorlayoutPart = anno4j.createObject(PartMMM.class);
colorlayoutPart.addInput(item);

ColorLayoutBody colorLayoutBody = anno4j.createObject(ColorLayoutBody.class);
colorLayoutBody.setYDCCoeff("38");
colorLayoutBody.setCbDCCoeff("22");
colorLayoutBody.setCrDCCoeff("34");
colorLayoutBody.setCbACCoeff("15 24");
colorLayoutBody.setCrACCoeff("18 9");
colorLayoutBody.setYACCoeff("11 28 13 18 9");

SpecificResource colorlayoutTarget = anno4j.createObject(SpecificResource.class);
colorlayoutTarget.setSource(item);

colorlayoutPart.addBody(colorLayoutBody);
colorlayoutPart.addTarget(colorlayoutTarget);

// Create Animaldetection Part
PartMMM animaldetectionPart = anno4j.createObject(PartMMM.class);
animaldetectionPart.addInput(item);

AnimalDetectionBody animaldetectionBody = anno4j.createObject(AnimalDetectionBody.
    class);
animaldetectionBody.setValue("Panda");
animaldetectionBody.setConfidence(0.9);

SpecificResourceMMM animaldetectionTarget = anno4j.createObject(
    SpecificResourceMMM.class);
FragmentSelector selector = anno4j.createObject(FragmentSelector.class);
selector.setSpatialFragment(300, 150, 50, 70);
selector.setConformsTo(SelectorFactory.getMediaFragmentsSpecification(anno4j));

animaldetectionTarget.setSelector(selector);
animaldetectionTarget.setSource(item);

animaldetectionPart.addBody(animaldetectionBody);
animaldetectionPart.addTarget(animaldetectionTarget);
```

Listing 30: Exemplary Creation of an MMM Item with Two Part Annotations Representing a ColorLayout and Animaldetection Result.

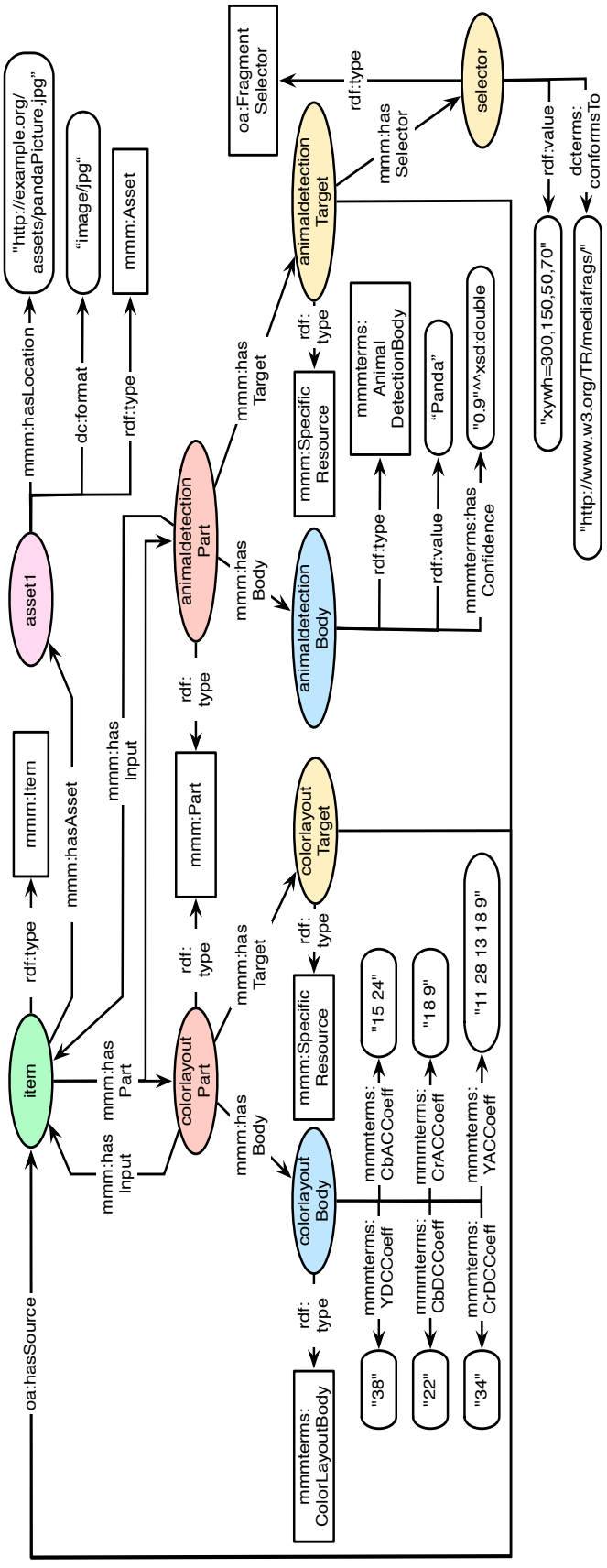


Figure 61: Resulting RDF Graph from the Code Enlisted in Listing 30, Creating an MMM Item with Two Part Annotations Representing a ColorLayout and Animaldetection Result.

```
// Necessary configuration
OntGenerationConfig config = new OntGenerationConfig();
// Language preference for identifiers and javadoc
config.setIdentifierLanguagePreference("en", OntGenerationConfig.UNTYPED_LITERAL);
config.setJavaDocLanguagePreference("en", "de", OntGenerationConfig.
    UNTYPED_LITERAL);

// Ambiguity checks turned off
config.setRDFSLabelAmbiguityChecking(false);
config.setFileOrFragmentAmbiguityChecking(false);

// Java base package where class files should be created
config.setBasePackage("com.someproject.pao.model");

JavaFileGenerator generator = new OWLJavaFileGenerator();
// Path to the respective schema file
generator.addRDF("/some/arbitrary/filepath/generationPAO2.rdfs.xml", "RDF/XML");
// Trigger the process by supporting the config and path to the destination
    directory
generator.generateJavaFiles(config, new File("/some/arbitrary/filepath/src/java/
    com/someproject/pao/model"));
```

Listing 31: Exemplary Use of the Anno4j Generation Tool.

```
<rdf:RDF xml:lang="en" xml:base="http://petsandowners.org/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/">

    <rdfs:Class rdf:about="Human">
        <rdfs:label xml:lang="en">Human</rdfs:label>
    </rdfs:Class>

    <rdfs:Class rdf:about="Animal">
        <rdfs:label xml:lang="en">Animal</rdfs:label>
    </rdfs:Class>

    <rdfs:Class rdf:about="Cat">
        <rdfs:subClassOf rdf:resource="Animal"/>
        <rdfs:label xml:lang="en">Cat</rdfs:label>
    </rdfs:Class>

    <rdf:Property rdf:about="has_Owner">
        <rdfs:label xml:lang="en">has owner</rdfs:label>
        <rdfs:domain rdf:resource="Cat"/>
        <rdfs:range rdf:resource="Human"/>
    </rdf:Property>

    <rdf:Property rdf:about="has_Cat_Friend">
        <rdfs:label xml:lang="en">has cat friend</rdfs:label>
        <rdfs:domain rdf:resource="Cat"/>
        <rdfs:range rdf:resource="Cat"/>
    </rdf:Property>

    <rdf:Property rdf:about="http://xmlns.com/foaf/0.1/age">
        <rdfs:label xml:lang="en">age</rdfs:label>
        <rdfs:domain rdf:resource="Cat"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </rdf:Property>

    <rdf:Property rdf:about="http://xmlns.com/foaf/0.1/name">
        <rdfs:label xml:lang="en">name</rdfs:label>
        <rdfs:domain rdf:resource="Cat"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </rdf:Property>
</rdf:RDF>
```

Listing 32: Example RDFS Schema for the Pets and Their Owners Ontology.

```
/**
 * Generated class for http://petsandowners.org/Human */
@Iri("http://petsandowners.org/Human")
public interface Human extends ResourceObject {
}

/**
 * Support class for {@link Human} */
@Partial
public abstract class HumanSupport extends SchemaSanitizingObjectSupport implements Human {
}

/**
 * Generated class for http://petsandowners.org/Animal */
@Iri("http://petsandowners.org/Animal")
public interface Animal extends ResourceObject {
}

/**
 * Support class for {@link Animal} */
@Partial
public abstract class AnimalSupport extends SchemaSanitizingObjectSupport implements Animal {
}

/**
 * Generated class for http://petsandowners.org/Cat */
@Iri("http://petsandowners.org/Cat")
public interface Cat extends Animal {
  @Iri("http://xmlns.com/foaf/0.1/name")
  Set<CharSequence> getNames();

  @Iri("http://petsandowners.org/has_Cat_Friend")
  Set<Cat> getHasCatFriends();

  @Iri("http://petsandowners.org/has_Owner")
  Set<Human> getHasOwners();

  @Iri("http://xmlns.com/foaf/0.1/age")
  Set<CharSequence> getAges();

  /**
   *
   * @param values The elements to set.
   * @throws IllegalArgumentException If the element(s) are not in the value space.
   * The value space is defined as:<ol>
   *    <li>The value is not null.</li>
   *
   * </ol> */
  void setHasOwners(Human... values);

  /**
   *
   * @param values The elements to set.
   * @throws IllegalArgumentException If the element(s) are not in the value space.
   * The value space is defined as:<ol>
   *    <li>The value is not null.</li>
   *
   * </ol> */
  void setAges(CharSequence... values);

  /**
   *
   * @param values The elements to set.
   * @throws IllegalArgumentException If the element(s) are not in the value space.
   * The value space is defined as:<ol>
   *    <li>The value is not null.</li>
   *
   * </ol> */
  void setHasCatFriends(Cat... values);

  /**
   *
   * @param values The elements to set.
   * @throws IllegalArgumentException If the element(s) are not in the value space.
   * The value space is defined as:<ol>
   *    <li>The value is not null.</li>
   *
   * </ol> */
  void setNames(CharSequence... values);

  /**
   *
   * @param values The elements to set.
   * @throws IllegalArgumentException If the element(s) are not in the value space.
   * The value space is defined as:<ol>
   *    <li>The value is not null.</li>
   *
   * </ol> */
  @Iri("http://xmlns.com/foaf/0.1/name")
```

```
90  void setNames(Set<? extends CharSequence> values);
91
92  /**
93   *
94   * @param values The elements to set.
95   * @throws IllegalArgumentException If the element(s) are not in the value space.
96   * The value space is defined as:<ol>
97   *    <li>The value is not null.</li>
98   *
99   * </ol> */
100 @Iri("http://xmlns.com/foaf/0.1/age")
101 void setAges(Set<? extends CharSequence> values);
102
103 /**
104  *
105  * @param values The elements to set.
106  * @throws IllegalArgumentException If the element(s) are not in the value space.
107  * The value space is defined as:<ol>
108  *    <li>The value is not null.</li>
109  *
110  * </ol> */
111 @Iri("http://petsandowners.org/has_Cat_Friend")
112 void setHasCatFriends(Set<Cat> values);
113
114 /**
115  *
116  * @param values The elements to set.
117  * @throws IllegalArgumentException If the element(s) are not in the value space.
118  * The value space is defined as:<ol>
119  *    <li>The value is not null.</li>
120  *
121  * </ol> */
122 @Iri("http://petsandowners.org/has_Owner")
123 void setHasOwners(Set<Human> values);
124
125 /**
126  *
127  * @param value The element to be added.
128  * @throws IllegalArgumentException If the element(s) are not in the value space.
129  * The value space is defined as:<ol>
130  *    <li>The value is not null.</li>
131  *
132  * </ol> */
133 void addHasCatFriend(Cat value);
134
135 /**
136  *
137  * @param value The element to be added.
138  * @throws IllegalArgumentException If the element(s) are not in the value space.
139  * The value space is defined as:<ol>
140  *    <li>The value is not null.</li>
141  *
142  * </ol> */
143 void addHasOwner(Human value);
144
145 /**
146  *
147  * @param value The element to be added.
148  * @throws IllegalArgumentException If the element(s) are not in the value space.
149  * The value space is defined as:<ol>
150  *    <li>The value is not null.</li>
151  *
152  * </ol> */
153 void addAge(CharSequence value);
154
155 /**
156  *
157  * @param value The element to be added.
158  * @throws IllegalArgumentException If the element(s) are not in the value space.
159  * The value space is defined as:<ol>
160  *    <li>The value is not null.</li>
161  *
162  * </ol> */
163 void addName(CharSequence value);
164
165 /**
166  *
167  * @param values The elements to be added.
168  * @throws IllegalArgumentException If the element(s) are not in the value space.
169  * The value space is defined as:<ol>
170  *    <li>The value is not null.</li>
171  *
172  * </ol> */
173 void addAllHasOwners(Set<? extends Human> values);
174
175 /**
176  *
177  * @param values The elements to be added.
178  * @throws IllegalArgumentException If the element(s) are not in the value space.
179  * The value space is defined as:<ol>
```

```
180     *    <li>The value is not null.</li>
181     *
182     * </ol> */
183    void addAllHasCatFriends(Set<? extends Cat> values);
184
185    /**
186     *
187     * @param values The elements to be added.
188     * @throws IllegalArgumentException If the element(s) are not in the value space.
189     * The value space is defined as:<ol>
190     *    <li>The value is not null.</li>
191     *
192     * </ol> */
193    void addAllNames(Set<? extends CharSequence> values);
194
195    /**
196     *
197     * @param values The elements to be added.
198     * @throws IllegalArgumentException If the element(s) are not in the value space.
199     * The value space is defined as:<ol>
200     *    <li>The value is not null.</li>
201     *
202     * </ol> */
203    void addAllAges(Set<? extends CharSequence> values);
204
205    /**
206     *
207     * @param value The element to be removed.
208     * @return Returns true if the element was present. Returns false otherwise. */
209    boolean removeHasCatFriend(Cat value);
210
211    /**
212     *
213     * @param value The element to be removed.
214     * @return Returns true if the element was present. Returns false otherwise. */
215    boolean removeAge(CharSequence value);
216
217    /**
218     *
219     * @param value The element to be removed.
220     * @return Returns true if the element was present. Returns false otherwise. */
221    boolean removeHasOwner(Human value);
222
223    /**
224     *
225     * @param value The element to be removed.
226     * @return Returns true if the element was present. Returns false otherwise. */
227    boolean removeName(CharSequence value);
228
229    /**
230     *
231     * @param values The elements to be removed.
232     * @return Returns true if any value was removed. */
233    boolean removeAllAges(Set<? extends CharSequence> values);
234
235    /**
236     *
237     * @param values The elements to be removed.
238     * @return Returns true if any value was removed. */
239    boolean removeAllNames(Set<? extends CharSequence> values);
240
241    /**
242     *
243     * @param values The elements to be removed.
244     * @return Returns true if any value was removed. */
245    boolean removeAllHasOwners(Set<? extends Human> values);
246
247    /**
248     *
249     * @param values The elements to be removed.
250     * @return Returns true if any value was removed. */
251    boolean removeAllHasCatFriends(Set<? extends Cat> values);
252   }
253
254   /**
255    * Support class for {@link Cat} */
256   @Partial
257   public abstract class CatSupport extends SchemaSanitizingObjectSupport implements Cat {
258    /**
259     *
260     * @param values The elements to set.
261     * @throws IllegalArgumentException If the element(s) are not in the value space.
262     * The value space is defined as:<ol>
263     *    <li>The value is not null.</li>
264     *
265     * </ol> */
266    @Override
267    public void setHasCatFriends(Cat... values) {
268     for(Cat current : values) {
269       if(current == null) {
```

```
270       throw new IllegalArgumentException("Value must not be null");
271     }
272   }
273   Set<Cat> _newValues = new HashSet<Cat>();
274   _newValues.addAll(Arrays.asList(values));
275   setHasCatFriends(_newValues);
276   sanitizeSchema("http://petsandowners.org/has_Cat_Friend");
277 }
278
279 /**
280  *
281  * @param values The elements to set.
282  * @throws IllegalArgumentException If the element(s) are not in the value space.
283  * The value space is defined as:<ol>
284  *    <li>The value is not null.</li>
285  *
286  * </ol> */
287 @Override
288 public void setHasOwners(Human... values) {
289   for(Human current : values) {
290     if(current == null) {
291       throw new IllegalArgumentException("Value must not be null");
292     }
293   }
294   Set<Human> _newValues = new HashSet<Human>();
295   _newValues.addAll(Arrays.asList(values));
296   setHasOwners(_newValues);
297   sanitizeSchema("http://petsandowners.org/has_Owner");
298 }
299
300 /**
301  *
302  * @param values The elements to set.
303  * @throws IllegalArgumentException If the element(s) are not in the value space.
304  * The value space is defined as:<ol>
305  *    <li>The value is not null.</li>
306  *
307  * </ol> */
308 @Override
309 public void setNames(CharSequence... values) {
310   for(CharSequence current : values) {
311     if(current == null) {
312       throw new IllegalArgumentException("Value must not be null");
313     }
314   }
315   Set<CharSequence> _newValues = new HashSet<CharSequence>();
316   _newValues.addAll(Arrays.asList(values));
317   setNames(_newValues);
318   sanitizeSchema("http://xmlns.com/foaf/0.1/name");
319 }
320
321 /**
322  *
323  * @param values The elements to set.
324  * @throws IllegalArgumentException If the element(s) are not in the value space.
325  * The value space is defined as:<ol>
326  *    <li>The value is not null.</li>
327  *
328  * </ol> */
329 @Override
330 public void setAges(CharSequence... values) {
331   for(CharSequence current : values) {
332     if(current == null) {
333       throw new IllegalArgumentException("Value must not be null");
334     }
335   }
336   Set<CharSequence> _newValues = new HashSet<CharSequence>();
337   _newValues.addAll(Arrays.asList(values));
338   setAges(_newValues);
339   sanitizeSchema("http://xmlns.com/foaf/0.1/age");
340 }
341
342 /**
343  *
344  * @param value The element to be added.
345  * @throws IllegalArgumentException If the element(s) are not in the value space.
346  * The value space is defined as:<ol>
347  *    <li>The value is not null.</li>
348  *
349  * </ol> */
350 @Override
351 public void addAge(CharSequence value) {
352   if(value == null) {
353     throw new IllegalArgumentException("Value must not be null");
354   }
355   Set<CharSequence> _acc = new HashSet<CharSequence>();
356   _acc.addAll(getAges());
357   _acc.add(value);
358   setAges(_acc);
359   sanitizeSchema("http://xmlns.com/foaf/0.1/age");
```

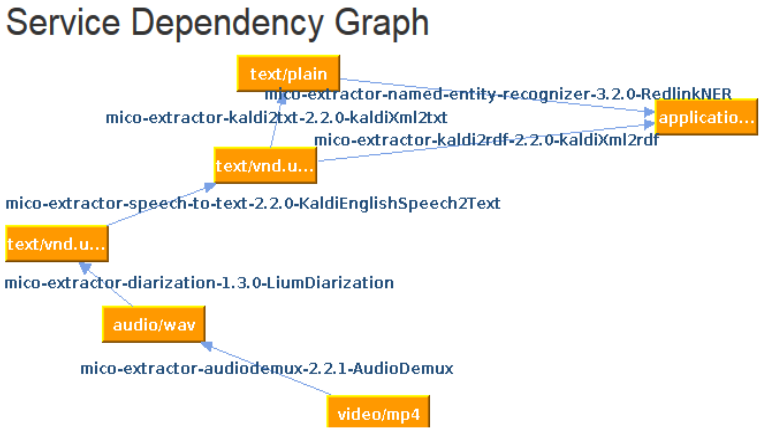
```
360   }
361
362   /**
363    *
364    * @param value The element to be added.
365    * @throws IllegalArgumentException If the element(s) are not in the value space.
366    * The value space is defined as:<ol>
367    *    <li>The value is not null.</li>
368    *
369    * </ol> */
370   @Override
371   public void addHasCatFriend(Cat value) {
372     if(value == null) {
373       throw new IllegalArgumentException("Value must not be null");
374     }
375     Set<Cat> _acc = new HashSet<Cat>();
376     _acc.addAll(getHasCatFriends());
377     _acc.add(value);
378     setHasCatFriends(_acc);
379     sanitizeSchema("http://petsandowners.org/has_Cat_Friend");
380   }
381
382   /**
383    *
384    * @param value The element to be added.
385    * @throws IllegalArgumentException If the element(s) are not in the value space.
386    * The value space is defined as:<ol>
387    *    <li>The value is not null.</li>
388    *
389    * </ol> */
390   @Override
391   public void addHasOwner(Human value) {
392     if(value == null) {
393       throw new IllegalArgumentException("Value must not be null");
394     }
395     Set<Human> _acc = new HashSet<Human>();
396     _acc.addAll(getHasOwners());
397     _acc.add(value);
398     setHasOwners(_acc);
399     sanitizeSchema("http://petsandowners.org/has_Owner");
400   }
401
402   /**
403    *
404    * @param value The element to be added.
405    * @throws IllegalArgumentException If the element(s) are not in the value space.
406    * The value space is defined as:<ol>
407    *    <li>The value is not null.</li>
408    *
409    * </ol> */
410   @Override
411   public void addName(CharSequence value) {
412     if(value == null) {
413       throw new IllegalArgumentException("Value must not be null");
414     }
415     Set<CharSequence> _acc = new HashSet<CharSequence>();
416     _acc.addAll(getNames());
417     _acc.add(value);
418     setNames(_acc);
419     sanitizeSchema("http://xmlns.com/foaf/0.1/name");
420   }
421
422   /**
423    *
424    * @param values The elements to be added.
425    * @throws IllegalArgumentException If the element(s) are not in the value space.
426    * The value space is defined as:<ol>
427    *    <li>The value is not null.</li>
428    *
429    * </ol> */
430   @Override
431   public void addAllHasCatFriends(Set<? extends Cat> values) {
432     for(Cat current : values) {
433       if(current == null) {
434         throw new IllegalArgumentException("Value must not be null");
435       }
436     }
437     Set<Cat> _acc = new HashSet<Cat>();
438     _acc.addAll(getHasCatFriends());
439     _acc.addAll(values);
440     setHasCatFriends(_acc);
441     sanitizeSchema("http://petsandowners.org/has_Cat_Friend");
442   }
443
444   /**
445    *
446    * @param values The elements to be added.
447    * @throws IllegalArgumentException If the element(s) are not in the value space.
448    * The value space is defined as:<ol>
449    *    <li>The value is not null.</li>
```

```
450     *
451     * </ol> */
452     @Override
453     public void addAllAges(Set<? extends CharSequence> values) {
454       for(CharSequence current : values) {
455         if(current == null) {
456           throw new IllegalArgumentException("Value must not be null");
457         }
458       }
459       Set<CharSequence> _acc = new HashSet<CharSequence>();
460       _acc.addAll(getAges());
461       _acc.addAll(values);
462       setAges(_acc);
463       sanitizeSchema("http://xmlns.com/foaf/0.1/age");
464     }
465
466     /**
467      *
468      * @param values The elements to be added.
469      * @throws IllegalArgumentException If the element(s) are not in the value space.
470      * The value space is defined as:<ol>
471      *    <li>The value is not null.</li>
472      *
473      * </ol> */
474     @Override
475     public void addAllHasOwners(Set<? extends Human> values) {
476       for(Human current : values) {
477         if(current == null) {
478           throw new IllegalArgumentException("Value must not be null");
479         }
480       }
481       Set<Human> _acc = new HashSet<Human>();
482       _acc.addAll(getHasOwners());
483       _acc.addAll(values);
484       setHasOwners(_acc);
485       sanitizeSchema("http://petsandowners.org/has_Owner");
486     }
487
488     /**
489      *
490      * @param values The elements to be added.
491      * @throws IllegalArgumentException If the element(s) are not in the value space.
492      * The value space is defined as:<ol>
493      *    <li>The value is not null.</li>
494      *
495      * </ol> */
496     @Override
497     public void addAllNames(Set<? extends CharSequence> values) {
498       for(CharSequence current : values) {
499         if(current == null) {
500           throw new IllegalArgumentException("Value must not be null");
501         }
502       }
503       Set<CharSequence> _acc = new HashSet<CharSequence>();
504       _acc.addAll(getNames());
505       _acc.addAll(values);
506       setNames(_acc);
507       sanitizeSchema("http://xmlns.com/foaf/0.1/name");
508     }
509
510     /**
511      *
512      * @param value The element to be removed.
513      * @return Returns true if the element was present. Returns false otherwise. */
514     @Override
515     public boolean removeHasCatFriend(Cat value) {
516       Set<Cat> _oldValues = getHasCatFriends();
517       if(_oldValues.contains(value)) {
518         removeValue("http://petsandowners.org/has_Cat_Friend", value);
519         sanitizeSchema("http://petsandowners.org/has_Cat_Friend");
520         // Refresh values:
521         if(getHasCatFriends() instanceof PropertySet) {
522           ((PropertySet) getHasCatFriends()).refresh();
523         }
524         return true;
525       }
526       return false;
527     }
528
529     /**
530      *
531      * @param value The element to be removed.
532      * @return Returns true if the element was present. Returns false otherwise. */
533     @Override
534     public boolean removeName(CharSequence value) {
535       Set<CharSequence> _oldValues = getNames();
536       if(_oldValues.contains(value)) {
537         removeValue("http://xmlns.com/foaf/0.1/name", value);
538         sanitizeSchema("http://xmlns.com/foaf/0.1/name");
539         // Refresh values:
```

```
540       if(getNames() instanceof PropertySet) {
541         ((PropertySet) getNames()).refresh();
542       }
543       return true;
544     }
545     return false;
546   }
547
548   /**
549    *
550    * @param value The element to be removed.
551    * @return Returns true if the element was present. Returns false otherwise. */
552   @Override
553   public boolean removeHasOwner(Human value) {
554     Set<Human> _oldValues = getHasOwners();
555     if(_oldValues.contains(value)) {
556       removeValue("http://petsandowners.org/has_Owner", value);
557       sanitizeSchema("http://petsandowners.org/has_Owner");
558       // Refresh values:
559       if(getHasOwners() instanceof PropertySet) {
560         ((PropertySet) getHasOwners()).refresh();
561       }
562       return true;
563     }
564     return false;
565   }
566
567   /**
568    *
569    * @param value The element to be removed.
570    * @return Returns true if the element was present. Returns false otherwise. */
571   @Override
572   public boolean removeAge(CharSequence value) {
573     Set<CharSequence> _oldValues = getAges();
574     if(_oldValues.contains(value)) {
575       removeValue("http://xmlns.com/foaf/0.1/age", value);
576       sanitizeSchema("http://xmlns.com/foaf/0.1/age");
577       // Refresh values:
578       if(getAges() instanceof PropertySet) {
579         ((PropertySet) getAges()).refresh();
580       }
581       return true;
582     }
583     return false;
584   }
585
586   /**
587    *
588    * @param values The elements to be removed.
589    * @return Returns true if any value was removed. */
590   @Override
591   public boolean removeAllHasOwners(Set<? extends Human> values) {
592     boolean changed = false;
593     Set<Human> _oldValues = getHasOwners();
594     Set<Human> _containedValues = new HashSet<Human>();
595     for(Human current : values) {
596       if(_oldValues.contains(current)) {
597         changed |= true;
598         _containedValues.add(current);
599       }
600     }
601     if(!_containedValues.isEmpty()) {
602       for(Human _current : _containedValues) {
603         removeValue("http://petsandowners.org/has_Owner", _current);
604       }
605     }
606     sanitizeSchema("http://petsandowners.org/has_Owner");
607     // Refresh values:
608     if(getHasOwners() instanceof PropertySet) {
609       ((PropertySet) getHasOwners()).refresh();
610     }
611     return changed;
612   }
613
614   /**
615    *
616    * @param values The elements to be removed.
617    * @return Returns true if any value was removed. */
618   @Override
619   public boolean removeAllAges(Set<? extends CharSequence> values) {
620     boolean changed = false;
621     Set<CharSequence> _oldValues = getAges();
622     Set<CharSequence> _containedValues = new HashSet<CharSequence>();
623     for(CharSequence current : values) {
624       if(_oldValues.contains(current)) {
625         changed |= true;
626         _containedValues.add(current);
627       }
628     }
629     if(!_containedValues.isEmpty()) {
```

```
630       for(CharSequence _current : _containedValues) {
631         removeValue("http://xmlns.com/foaf/0.1/age", _current);
632       }
633     }
634     sanitizeSchema("http://xmlns.com/foaf/0.1/age");
635     // Refresh values:
636     if(getAges() instanceof PropertySet) {
637       ((PropertySet) getAges()).refresh();
638     }
639     return changed;
640   }
641
642   /**
643    *
644    * @param values The elements to be removed.
645    * @return Returns true if any value was removed. */
646   @Override
647   public boolean removeAllNames(Set<? extends CharSequence> values) {
648     boolean changed = false;
649     Set<CharSequence> _oldValues = getNames();
650     Set<CharSequence> _containedValues = new HashSet<CharSequence>();
651     for(CharSequence current : values) {
652       if(_oldValues.contains(current)) {
653         changed |= true;
654         _containedValues.add(current);
655       }
656     }
657     if(!_containedValues.isEmpty()) {
658       for(CharSequence _current : _containedValues) {
659         removeValue("http://xmlns.com/foaf/0.1/name", _current);
660       }
661     }
662     sanitizeSchema("http://xmlns.com/foaf/0.1/name");
663     // Refresh values:
664     if(getNames() instanceof PropertySet) {
665       ((PropertySet) getNames()).refresh();
666     }
667     return changed;
668   }
669
670   /**
671    *
672    * @param values The elements to be removed.
673    * @return Returns true if any value was removed. */
674   @Override
675   public boolean removeAllHasCatFriends(Set<? extends Cat> values) {
676     boolean changed = false;
677     Set<Cat> _oldValues = getHasCatFriends();
678     Set<Cat> _containedValues = new HashSet<Cat>();
679     for(Cat current : values) {
680       if(_oldValues.contains(current)) {
681         changed |= true;
682         _containedValues.add(current);
683       }
684     }
685     if(!_containedValues.isEmpty()) {
686       for(Cat _current : _containedValues) {
687         removeValue("http://petsandowners.org/has_Cat_Friend", _current);
688       }
689     }
690     sanitizeSchema("http://petsandowners.org/has_Cat_Friend");
691     // Refresh values:
692     if(getHasCatFriends() instanceof PropertySet) {
693       ((PropertySet) getHasCatFriends()).refresh();
694     }
695     return changed;
696   }
697 }
```

Listing 33: Exemplary Output of the Code Shown in Listing 31 with the Input Schema Shown in Listing 32.



Items

URI	Actions	Registration	State
http://mico-platform:8080/marmotta/aa6a9f95-defd-4ef0-86eb-4931d81a35ba	Inspect - Metadata	2016-11-21T14:55:19.209Z	In Progress
http://mico-platform:8080/marmotta/01b889ea-d401-4bc0-a660-18d07bb6d704	Inspect - Metadata	2016-11-21T13:59:21.109Z	Finished
http://mico-platform:8080/marmotta/01b889ea-d401-4bc0-a660-18d07bb6d704	Inspect - Metadata	2016-11-21T11:23:51.348Z	In Progress
http://mico-platform:8080/marmotta/49d95720-b391-42b7-97e3-cfdb39143ca2	Inspect - Metadata	2016-11-21T10:48:58.230Z	Failed
http://mico-platform:8080/marmotta/2c519ee5-4470-48c0-a5e2-05f54b43674d	Inspect - Metadata	2016-11-21T08:33:59.639Z	Finished
http://mico-platform:8080/marmotta/6270eaf0-75e5-44ff-a181-0fab80ae3c05	Inspect - Metadata	2016-11-21T08:28:41.212Z	Finished
http://mico-platform:8080/marmotta/cd6f19b0-e2dc-47a5-a0ab-8100c7527704	Inspect - Metadata	2016-11-21T08:02:01.674Z	Finished
http://mico-platform:8080/marmotta/5df4c10c-d214-40ff-b90f-177d5fc0f044	Inspect - Metadata	2016-11-21T07:53:34.717Z	Finished
http://mico-platform:8080/marmotta/a17d3a38-01ad-4a29-b610-881cbac6941e	Inspect - Metadata	2016-11-21T07:46:42.816Z	Finished

Figure 62: Screenshot of the MICO Platform Item Overview.

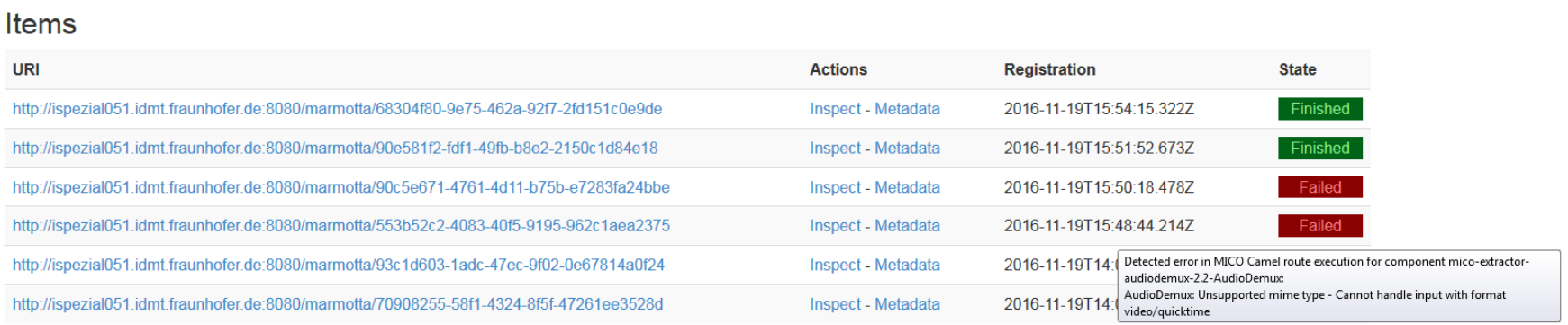


Figure 63: Screenshot of the MICO Platform Item Overview, Hovering an as “Failed” Marked Item Progress.

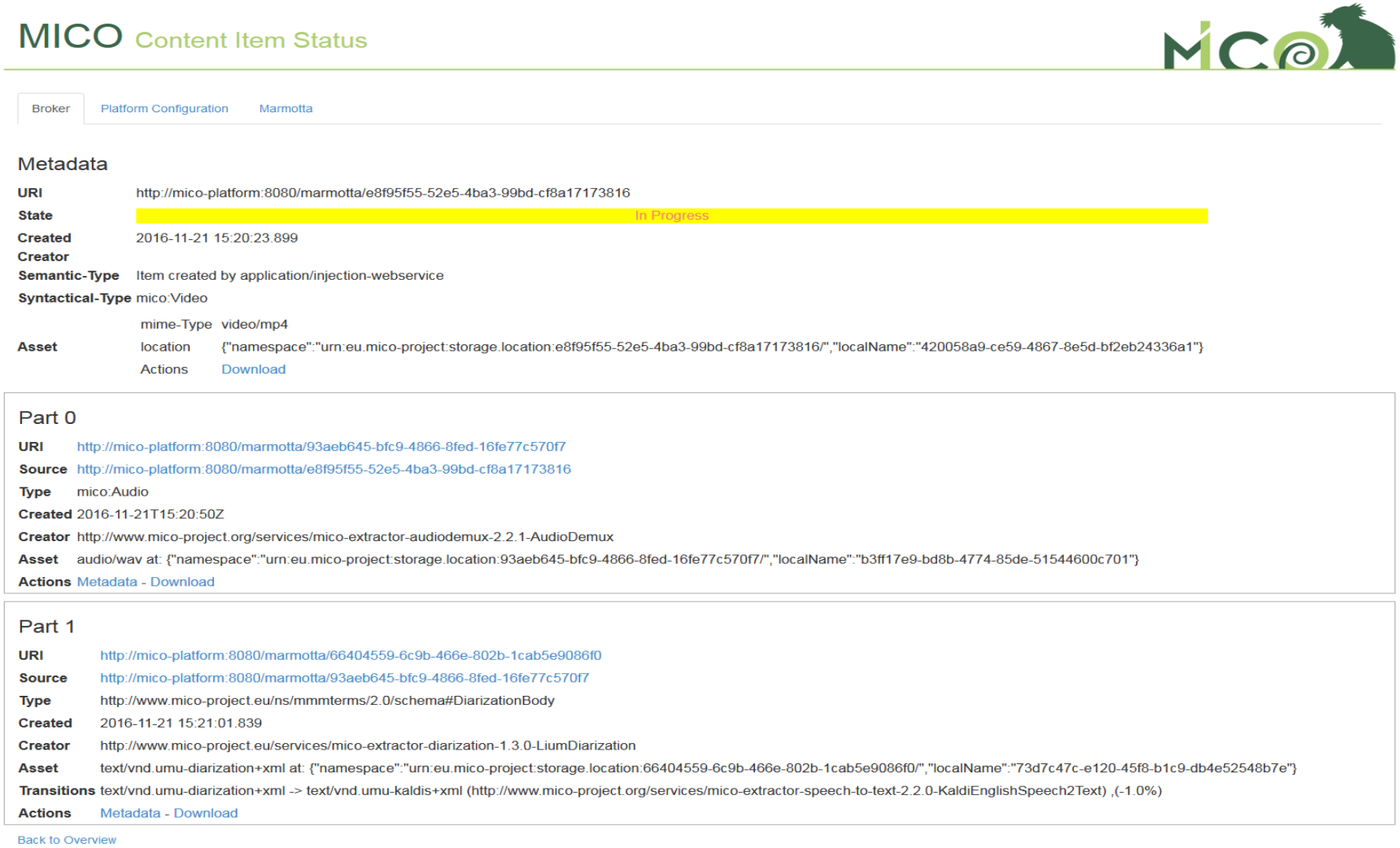
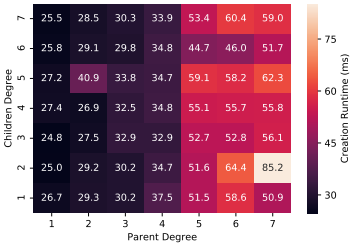
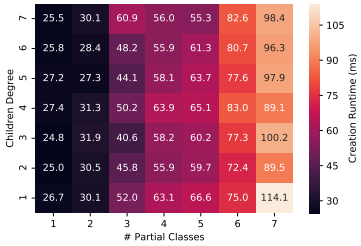


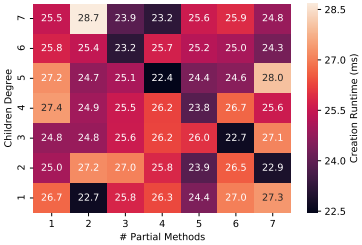
Figure 64: Screenshot of the MICO Platform View Inspecting an Item in Detail.



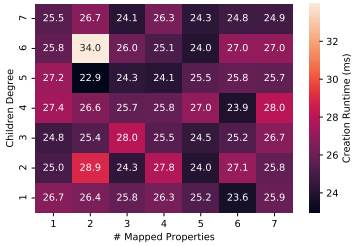
(a) Heatmap for the Children Degree and Parent Degree Parameters.



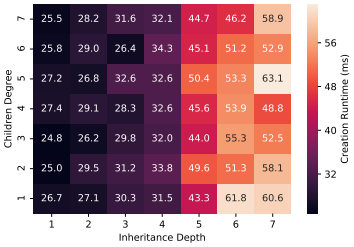
(b) Heatmap for the Children Degree and Partial Classes Parameters.



(c) Heatmap for the Children Degree and Partial Methods Parameters.



(d) Heatmap for the Children Degree and Properties/Relationships Parameters.



(e) Heatmap for the Children Degree and Inheritance Depth Parameters.

Figure 65: Resulting Heatmaps of the Paired Ontology Parameter Evaluation (1).

General Name	OWL / RDFS	Schema Annotation
Symmetry	owl:SymmetricProperty	@Symmetric
A relationship with this Schema Annotation directed from RDF entity "x" to "y" is only valid, if the same relationship is also present directing from "y" to "x".		
Transitivity	owl:TransitiveProperty	@Transitive
This Schema Annotation is only validated to true for two relationships from RDF entity "x" to "y" and "y" to "z" respectively, if there is also an edge from "x" to "z".		
Functional Property	owl:FunctionalProperty	@Functional
A relationship/property marked with this Schema Annotation can only be successfully validated, if for every source "x" there is only exactly one sink "y".		
Inverse Functional Property	owl:InverseFunctionalProperty	@InverseFunctional
The counterpart to "`owl:FunctionalProperty`" is only valid, if every sink "y" does only have exactly one source "x" for the relationship or property.		
Bijective Property	-	@Bijective
This Schema Annotation combines "`owl:FunctionalProperty`" and "`InverseFunctionalProperty`".		
Inverse Property	owl:inverseOf	@InverseOf
For a relationship "r" there can be an inverse relationship (with another IRI) "s" defined, inducing validity only if for every "r" from "x" to "y", there must also exist an edge of "s" from "y" to "x".		
Subproperty	rdfs:subPropertyOf	@SubPropertyOf
Marks a relationship or property to inherit all the values of the defined superproperty.		
(Min/Max) Cardinality	owl:cardinality owl:minCardinality owl:maxCardinality	@Cardinality @MinCardinality @MaxCardinality
A cardinality restriction can be defined for a relationship or property, being only validated when the multiplicity of the respective edge is inside the defined boundaries.		
All Values From	owl:allValuesFrom	@AllValuesFrom
Restricts a given relationship or property to a defined set of classes and types for its sink, so if there is an edge of the respective relationship or property from "x" to "y", then "y" must always be conform to the defined set. Corresponds to the *for all* quantifier of Predicate logic.		
Some Values From	owl:someValuesFrom	@SomeValuesFrom
Restricts a given relationship or property to a defined set of classes or types for its sink, so if there are several representatives of the respective relationship or property, at least one of their sinks must correspond to the defined set. Corresponds to the *existential* quantifier of Predicate logic.		

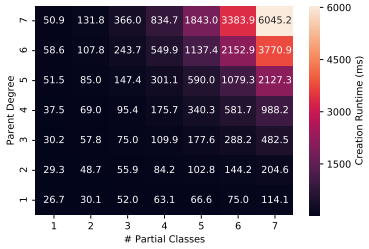
Table 4: Available Schema Annotations in Anno4j. The Table Shows a General Name, the Respective Pendant in the OWL or RDFS Schema, and the Corresponding (Java) Schema Annotation, in Combination with a Description of the Implemented Concept.

CPU	Intel®Core™ i5-2400 (3,10 GHz)
RAM	8GB DDR3
OS	Gentoo 2.3 (Linux Kernel 4.12.12)
Java-Version	1.7
JVM	Oracle®Java™ SE Runtime Environment (build 1.8.0_144-b01)
Anno4j	Version 2.4

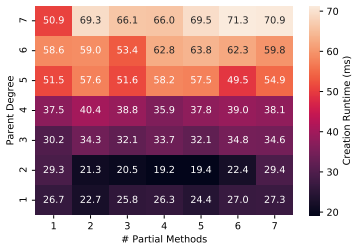
Table 5: Properties of the Test System Used in the Benchmarks Enlisted in Section 6.2.1.

CPU	AMD Opteron™ Processor 6220, with 16 cores and 2 threads per core (3 GHz)
RAM	264GB
OS	Ubuntu 18.04
Java-Version	1.8
JVM	Oracle®Java™ SE Runtime Environment (build 1.8.0_171-b11)
Anno4j	Version 2.4

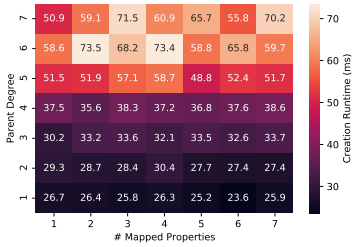
Table 6: Properties of the Test System Used in the Benchmarks Enlisted in Section 6.2.3.



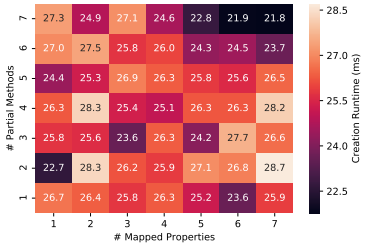
(a) Heatmap for the Parent Degree and Partial Class Parameters.



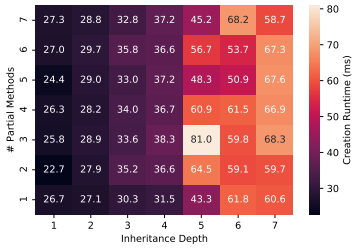
(b) Heatmap for the Parent Degree and Partial Methods Parameters.



(c) Heatmap for the Parent Degree and Properties/Relationships Parameters.

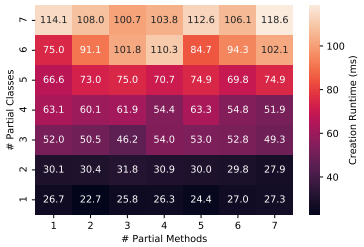


(d) Heatmap for the Partial Methods and Properties/Relationships Parameters.

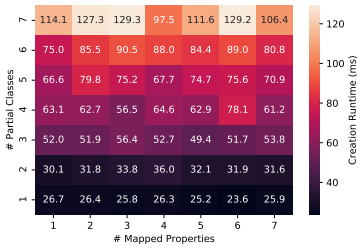


(e) Heatmap for the Partial Methods and Inheritance Depth Parameters.

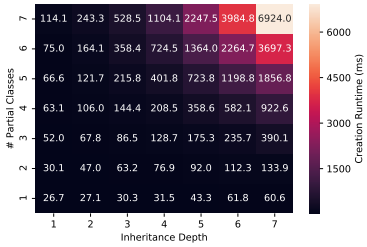
Figure 66: Resulting Heatmaps of the Paired Ontology Parameter Evaluation (2).



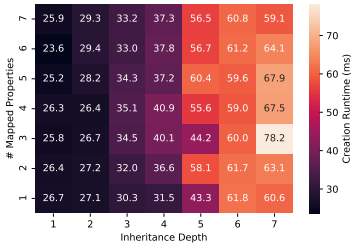
(a) Heatmap for the Partial Classes and Partial Methods Parameters.



(b) Heatmap for the Partial Classes and Properties/Relationships Parameters.



(c) Heatmap for the Partial Classes and Inheritance Depth Parameters.



(d) Heatmap for the Properties/Relationships and Inheritance Depth Parameters.

Figure 67: Resulting Heatmaps of the Paired Ontology Parameter Evaluation (3).

cd	pd	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
7	5	53.417	34.606	95.752	9.575	18.767
2	7	85.209	45.738	203.242	20.324	39.835
6	2	29.124	22.613	15.297	1.53	2.998
5	6	58.202	38.509	97.244	9.724	19.06
7	7	58.975	45.292	60.864	6.086	11.929
2	5	51.554	33.492	112.663	11.266	22.082
4	3	32.476	28.749	16.328	1.633	3.2
3	2	27.456	23.839	13.902	1.39	2.725
5	4	34.665	30.66	13.973	1.397	2.739
6	6	45.993	38.256	21.52	2.152	4.218
3	4	32.943	29.301	13.323	1.332	2.611
5	2	40.922	23.967	121.896	12.19	23.892
2	3	30.154	25.551	15.784	1.578	3.094
4	5	55.137	30.997	132.185	13.218	25.908
6	4	34.754	32.24	14.141	1.414	2.772
3	6	52.793	33.738	109.001	10.9	21.364
7	3	30.259	24.583	17.169	1.717	3.365
4	7	55.816	41.76	66.597	6.66	13.053
6	7	51.743	42.549	47.602	4.76	9.33
3	5	52.701	32.484	99.16	9.916	19.435
5	3	33.766	25.66	17.087	1.709	3.349
2	2	29.173	21.285	19.234	1.923	3.77
4	4	34.779	30.574	14.646	1.465	2.871
6	5	44.71	33.337	63.371	6.337	12.421
3	7	56.106	41.111	66.811	6.681	13.095
7	2	28.496	23.861	16.267	1.627	3.188
4	6	55.65	35.613	104.633	10.463	20.508
7	4	33.928	31.159	13.902	1.39	2.725
2	6	64.367	36.859	157.026	15.703	30.777
6	3	29.814	25.591	12.628	1.263	2.475
5	7	62.333	42.5	102.958	10.296	20.18
7	6	60.4	37.419	123.918	12.392	24.288
2	4	34.657	31.539	13.929	1.393	2.73
4	2	26.914	22.776	12.105	1.211	2.373
3	3	32.887	27.968	15.903	1.59	3.117
5	5	59.064	33.052	141.094	14.109	27.654
4	1	27.387	19.699	18.335	1.833	3.594
7	1	25.472	19.678	15.931	1.593	3.122
2	1	25.017	19.888	13.056	1.306	2.559
5	1	27.23	21.333	14.808	1.481	2.902
3	1	24.774	20.309	13.953	1.395	2.735
6	1	25.754	21.662	13.037	1.304	2.555
1	3	30.155	26.558	14.643	1.464	2.87
1	7	50.863	40.608	58.155	5.816	11.398
1	5	51.489	32.733	95.36	9.536	18.69
1	6	58.575	39.758	102.669	10.267	20.123
1	4	37.489	30.448	19.729	1.973	3.867
1	2	29.267	23.728	14.493	1.449	2.841
1	1	26.693	21.201	16.266	1.627	3.188

Table 7: Ontology Parameter Pair Evaluation for Children Degree and Parent Degree.

cd	pc	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
2	7	89.452	67.054	102.405	10.241	20.071
7	5	55.331	43.075	79.794	7.979	15.64
7	4	56.032	35.819	104.171	10.417	20.417
2	6	72.425	57.072	79.485	7.948	15.579
4	3	50.248	30.201	102.847	10.285	20.158
2	5	59.665	44.159	83.076	8.308	16.283
7	7	98.423	72.551	139.962	13.996	27.433
7	6	82.624	55.213	171.015	17.101	33.519
2	4	55.862	37.827	102.166	10.217	20.025
4	2	31.256	24.235	18.886	1.889	3.702
4	4	63.9	39.317	138.335	13.834	27.114
2	2	30.524	24.562	17.681	1.768	3.465
2	3	45.795	28.964	84.664	8.466	16.594
4	5	65.109	45.554	115.067	11.507	22.553
4	6	83.018	55.081	164.894	16.489	32.319
7	2	30.118	25.62	14.779	1.478	2.897
7	3	60.865	29.615	165.964	16.596	32.529
4	7	89.144	71.279	92.574	9.257	18.144
3	5	60.157	45.19	74.045	7.405	14.513
6	7	96.348	70.77	139.919	13.992	27.424
5	3	44.143	29.745	70.955	7.095	13.907
5	2	27.347	23.395	10.992	1.099	2.154
6	6	80.661	56.212	147.621	14.762	28.934
3	4	58.174	38.422	127.925	12.793	25.073
3	7	100.184	69.179	189.878	18.988	37.216
6	5	61.349	44.712	91.645	9.164	17.962
6	4	55.925	36.234	98.755	9.875	19.356
3	6	77.287	57.333	108.298	10.83	21.226
6	2	28.393	24.227	13.162	1.316	2.58
5	6	77.604	53.185	135.189	13.519	26.497
5	7	97.938	73.698	121.577	12.158	23.829
6	3	48.192	30.44	93.502	9.35	18.326
3	2	31.884	26.804	16.164	1.616	3.168
5	4	58.085	38.235	101.508	10.151	19.896
5	5	63.673	44.719	117.812	11.781	23.091
3	3	40.551	30.14	62.77	6.277	12.303
4	1	27.387	19.699	18.335	1.833	3.594
7	1	25.472	19.678	15.931	1.593	3.122
2	1	25.017	19.888	13.056	1.306	2.559
5	1	27.23	21.333	14.808	1.481	2.902
3	1	24.774	20.309	13.953	1.395	2.735
6	1	25.754	21.662	13.037	1.304	2.555
1	3	51.959	29.475	133.372	13.337	26.141
1	2	30.122	24.189	14.702	1.47	2.882
1	6	75.045	54.983	110.475	11.047	21.653
1	7	114.146	70.607	212.429	21.243	41.636
1	4	63.127	37.829	138.158	13.816	27.079
1	5	66.557	48.654	93.846	9.385	18.394
1	1	26.693	21.201	16.266	1.627	3.188

Table 8: Ontology Parameter Pair Evaluation for Children Degree and Partial Classes.

cd	pm	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
7	7	24.825	19.824	14.776	1.478	2.896
5	5	24.375	20.265	11.446	1.145	2.243
3	4	26.247	21.057	14.568	1.457	2.855
6	3	23.183	18.998	10.855	1.086	2.128
2	7	22.903	18.689	12.056	1.206	2.363
4	6	26.658	20.972	15.004	1.5	2.941
3	3	25.585	20.261	14.221	1.422	2.787
5	2	24.715	17.936	14.895	1.49	2.919
6	4	25.702	20.83	13.7	1.37	2.685
5	3	25.068	20.412	13.594	1.359	2.664
3	2	24.818	19.743	12.765	1.276	2.502
6	5	25.248	19.849	13.519	1.352	2.65
4	7	25.575	20.738	13.058	1.306	2.559
2	6	26.547	19.824	18.049	1.805	3.538
3	5	25.981	21.053	14.717	1.472	2.884
5	4	22.448	19.169	9.818	0.982	1.924
6	2	25.42	21.081	13.058	1.306	2.559
7	6	25.896	21.763	13.71	1.371	2.687
5	6	24.591	20.216	11.451	1.145	2.244
3	7	27.091	21.161	16.333	1.633	3.201
4	2	24.942	19.908	14.754	1.475	2.892
2	3	27.037	22.861	15.293	1.529	2.998
7	4	23.207	20.076	10.056	1.006	1.971
6	7	24.339	19.598	14.347	1.435	2.812
2	4	25.808	21.373	12.557	1.256	2.461
4	5	23.835	20.122	12.784	1.278	2.506
7	3	23.903	18.896	12.719	1.272	2.493
4	4	26.242	19.727	15.212	1.521	2.981
2	5	23.859	19.501	12.345	1.235	2.42
7	2	28.675	21.757	16.916	1.692	3.316
6	6	25.036	18.947	15.966	1.597	3.129
2	2	27.189	20.663	18.874	1.887	3.699
4	3	25.544	19.677	16.136	1.614	3.163
7	5	25.562	22.07	11.293	1.129	2.213
3	6	22.674	19.328	11.446	1.145	2.243
5	7	27.987	21.878	14.709	1.471	2.883
4	1	27.387	19.699	18.335	1.833	3.594
7	1	25.472	19.678	15.931	1.593	3.122
2	1	25.017	19.888	13.056	1.306	2.559
5	1	27.23	21.333	14.808	1.481	2.902
3	1	24.774	20.309	13.953	1.395	2.735
6	1	25.754	21.662	13.037	1.304	2.555
1	6	27.018	20.487	15.352	1.535	3.009
1	7	27.344	22.133	17.468	1.747	3.424
1	5	24.433	20.863	12.414	1.241	2.433
1	2	22.668	19.844	10.447	1.045	2.048
1	3	25.761	22.225	11.917	1.192	2.336
1	4	26.292	19.741	18.548	1.855	3.635
1	1	26.693	21.201	16.266	1.627	3.188

Table 9: Ontology Parameter Pair Evaluation for Children Degree and Partial Methods.

cd	prop	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
4	7	28.003	23.184	16.631	1.663	3.26
6	6	26.979	22.098	13.899	1.39	2.724
2	4	27.784	22.023	15.153	1.515	2.97
2	2	28.94	21.374	20.054	2.005	3.931
3	6	25.24	21.037	13.005	1.301	2.549
7	2	26.711	20.611	15.23	1.523	2.985
5	5	25.496	21.377	12.674	1.267	2.484
5	3	24.291	18.692	13.801	1.38	2.705
7	4	26.305	21.135	15.121	1.512	2.964
3	5	24.468	19.548	13.32	1.332	2.611
3	3	28.01	19.79	18.822	1.882	3.689
5	6	25.837	22.163	13.114	1.311	2.57
7	7	24.882	19.69	14.688	1.469	2.879
4	4	25.806	20.43	13.08	1.308	2.564
6	3	26.034	21.175	13.343	1.334	2.615
6	5	23.986	19.445	12.717	1.272	2.493
4	2	26.556	21.727	13.276	1.328	2.602
2	7	25.754	21.614	13.689	1.369	2.683
2	6	27.104	20.584	18.109	1.811	3.549
6	2	33.991	19.503	117.826	11.783	23.094
4	5	26.969	20.471	17.288	1.729	3.388
4	3	25.719	20.785	14.546	1.455	2.851
6	4	25.11	19.755	13.368	1.337	2.62
5	7	25.745	22.0	12.416	1.242	2.434
7	6	24.834	20.83	12.569	1.257	2.464
3	4	25.5	19.883	16.15	1.615	3.165
3	2	25.387	20.645	13.824	1.382	2.709
5	4	24.116	20.629	11.335	1.133	2.222
7	3	24.113	21.287	10.288	1.029	2.016
7	5	24.272	20.248	12.528	1.253	2.455
5	2	22.941	18.371	12.005	1.2	2.353
3	7	26.703	19.965	16.181	1.618	3.171
2	5	23.982	18.673	13.974	1.397	2.739
2	3	24.343	19.965	13.73	1.373	2.691
4	6	23.861	19.665	13.076	1.308	2.563
6	7	26.984	20.547	19.175	1.918	3.758
4	1	27.387	19.699	18.335	1.833	3.594
7	1	25.472	19.678	15.931	1.593	3.122
2	1	25.017	19.888	13.056	1.306	2.559
5	1	27.23	21.333	14.808	1.481	2.902
3	1	24.774	20.309	13.953	1.395	2.735
6	1	25.754	21.662	13.037	1.304	2.555
1	7	25.905	21.319	12.591	1.259	2.468
1	2	26.378	20.754	15.755	1.576	3.088
1	4	26.304	23.486	12.127	1.213	2.377
1	3	25.802	22.086	11.821	1.182	2.317
1	5	25.248	20.017	16.569	1.657	3.248
1	6	23.648	20.627	10.668	1.067	2.091
1	1	26.693	21.201	16.266	1.627	3.188

Table 10: Ontology Parameter Pair Evaluation for Children Degree and Properties and Relationships.

cd	d	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
2	5	49.581	29.839	99.575	9.957	19.517
7	6	46.176	33.558	70.571	7.057	13.832
7	7	58.895	40.547	106.664	10.666	20.906
2	4	33.825	27.741	17.721	1.772	3.473
2	6	51.339	34.384	82.608	8.261	16.191
4	2	29.063	24.505	13.212	1.321	2.589
7	5	44.67	33.986	62.453	6.245	12.241
7	4	32.105	27.691	13.432	1.343	2.633
4	3	28.252	24.233	14.068	1.407	2.757
2	7	58.114	40.054	97.224	9.722	19.056
2	3	31.231	25.518	14.911	1.491	2.923
4	7	48.781	36.423	61.63	6.163	12.079
4	6	53.942	36.202	95.638	9.564	18.745
2	2	29.492	24.949	13.026	1.303	2.553
4	4	32.59	27.491	13.071	1.307	2.562
7	3	31.581	24.861	16.799	1.68	3.293
7	2	28.205	22.929	13.365	1.336	2.619
4	5	45.584	33.184	70.616	7.062	13.841
3	6	55.332	37.916	99.901	9.99	19.581
5	2	26.782	21.888	15.482	1.548	3.034
6	5	45.145	31.534	70.552	7.055	13.828
6	4	34.253	29.187	17.502	1.75	3.43
5	3	32.588	25.194	19.046	1.905	3.733
3	7	52.512	38.495	72.074	7.207	14.127
3	5	44.006	29.612	81.515	8.152	15.977
6	6	51.188	33.259	99.394	9.939	19.481
6	7	52.936	38.915	72.201	7.22	14.151
3	4	32.019	26.7	12.855	1.286	2.52
5	4	32.597	27.683	15.86	1.586	3.108
6	3	26.369	23.551	9.846	0.985	1.93
6	2	28.974	22.172	15.195	1.52	2.978
5	5	50.391	31.721	101.206	10.121	19.836
3	3	29.832	24.272	13.692	1.369	2.684
5	7	63.059	38.519	166.095	16.609	32.555
5	6	53.337	36.673	91.464	9.146	17.927
3	2	26.233	21.638	13.348	1.335	2.616
4	1	27.387	19.699	18.335	1.833	3.594
7	1	25.472	19.678	15.931	1.593	3.122
2	1	25.017	19.888	13.056	1.306	2.559
5	1	27.23	21.333	14.808	1.481	2.902
3	1	24.774	20.309	13.953	1.395	2.735
6	1	25.754	21.662	13.037	1.304	2.555
1	2	27.13	22.629	12.824	1.282	2.513
1	3	30.331	26.109	12.726	1.273	2.494
1	4	31.451	27.463	12.169	1.217	2.385
1	5	43.329	31.582	73.906	7.391	14.486
1	7	60.637	41.891	94.306	9.431	18.484
1	6	61.827	34.954	142.661	14.266	27.962
1	1	26.693	21.201	16.266	1.627	3.188

Table 11: Ontology Parameter Pair Evaluation for Children Degree and Inheritance Depth.

pd	pc	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
5	7	2127.316	1971.024	581.012	58.101	113.878
7	5	1842.983	1698.622	479.46	47.946	93.974
5	6	1079.287	964.206	286.77	28.677	56.207
7	4	834.731	743.209	314.467	31.447	61.636
7	7	6045.152	5645.861	1364.732	136.473	267.487
5	5	590.018	518.636	266.156	26.616	52.167
3	3	75.031	60.71	80.693	8.069	15.816
7	6	3383.872	3298.029	677.773	67.777	132.844
3	2	57.769	38.504	112.719	11.272	22.093
5	4	301.07	270.102	109.803	10.98	21.521
5	2	85.02	62.552	115.008	11.501	22.542
3	4	109.862	95.768	81.589	8.159	15.992
3	5	177.631	155.438	90.151	9.015	17.67
5	3	147.397	129.427	98.868	9.887	19.378
3	6	288.207	249.646	144.036	14.404	28.231
7	2	131.777	113.436	103.439	10.344	20.274
3	7	482.514	430.065	212.92	21.292	41.732
7	3	366.032	292.324	277.807	27.781	54.45
4	2	68.997	50.615	108.057	10.806	21.179
2	4	84.155	58.421	148.555	14.856	29.117
2	5	102.793	83.312	101.56	10.156	19.906
4	3	95.375	88.965	24.206	2.421	4.744
2	6	144.218	123.502	98.537	9.854	19.313
6	2	107.812	81.691	121.442	12.144	23.803
2	7	204.585	171.08	156.089	15.609	30.593
6	3	243.711	208.524	144.456	14.446	28.313
4	7	988.225	889.384	299.657	29.966	58.733
6	5	1137.417	999.718	428.568	42.857	83.999
4	6	581.705	510.781	265.846	26.585	52.106
6	4	549.928	461.695	258.969	25.897	50.758
6	7	3770.878	3566.996	793.109	79.311	155.449
4	5	340.333	308.52	153.988	15.399	30.182
2	3	55.94	40.739	80.894	8.089	15.855
6	6	2152.933	2043.929	535.53	53.553	104.964
2	2	48.736	31.988	82.165	8.217	16.104
4	4	175.67	157.529	83.174	8.317	16.302
3	1	30.155	26.558	14.643	1.464	2.87
7	1	50.863	40.608	58.155	5.816	11.398
5	1	51.489	32.733	95.36	9.536	18.69
6	1	58.575	39.758	102.669	10.267	20.123
4	1	37.489	30.448	19.729	1.973	3.867
2	1	29.267	23.728	14.493	1.449	2.841
1	3	51.959	29.475	133.372	13.337	26.141
1	2	30.122	24.189	14.702	1.47	2.882
1	6	75.045	54.983	110.475	11.047	21.653
1	7	114.146	70.607	212.429	21.243	41.636
1	4	63.127	37.829	138.158	13.816	27.079
1	5	66.557	48.654	93.846	9.385	18.394
1	1	26.693	21.201	16.266	1.627	3.188

Table 12: Ontology Parameter Pair Evaluation for Parent Degree and Partial Classes.

pd	pm	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
7	7	70.936	55.931	119.815	11.982	23.484
2	4	19.231	17.418	7.694	0.769	1.508
4	5	37.782	37.95	14.236	1.424	2.79
6	2	59.006	41.882	99.799	9.98	19.561
5	7	54.892	40.946	106.457	10.646	20.866
3	6	34.847	35.714	12.346	1.235	2.42
6	5	63.838	44.808	122.291	12.229	23.969
4	2	40.434	37.577	18.34	1.834	3.595
2	3	20.467	18.335	8.424	0.842	1.651
3	7	34.591	34.67	11.563	1.156	2.266
5	6	49.531	42.152	73.432	7.343	14.393
2	2	21.296	17.864	12.231	1.223	2.397
4	3	38.843	38.774	15.23	1.523	2.985
6	4	62.848	45.854	104.378	10.438	20.458
7	6	71.323	50.783	111.709	11.171	21.895
6	3	53.397	46.937	70.99	7.099	13.914
4	4	35.928	32.069	15.637	1.564	3.065
2	5	19.357	17.253	7.307	0.731	1.432
2	7	29.428	27.182	15.321	1.532	3.003
4	6	38.955	37.366	14.801	1.48	2.901
3	2	34.252	32.724	14.078	1.408	2.759
7	4	66.018	52.818	91.463	9.146	17.927
5	3	51.559	40.145	82.497	8.25	16.169
6	6	62.292	49.666	84.343	8.434	16.531
5	4	58.164	42.208	97.032	9.703	19.018
7	3	66.083	59.507	76.509	7.651	14.996
3	5	32.145	29.042	12.362	1.236	2.423
6	7	59.808	45.077	91.978	9.198	18.028
3	4	33.7	30.572	15.527	1.553	3.043
7	2	69.268	55.28	97.354	9.735	19.081
5	5	57.549	42.052	99.774	9.977	19.556
4	7	38.13	34.281	22.198	2.22	4.351
2	6	22.412	18.823	9.266	0.927	1.816
5	2	57.573	41.784	113.759	11.376	22.297
7	5	69.461	49.462	132.592	13.259	25.988
3	3	32.101	27.884	14.463	1.446	2.835
3	1	30.155	26.558	14.643	1.464	2.87
7	1	50.863	40.608	58.155	5.816	11.398
5	1	51.489	32.733	95.36	9.536	18.69
6	1	58.575	39.758	102.669	10.267	20.123
4	1	37.489	30.448	19.729	1.973	3.867
2	1	29.267	23.728	14.493	1.449	2.841
1	6	27.018	20.487	15.352	1.535	3.009
1	7	27.344	22.133	17.468	1.747	3.424
1	5	24.433	20.863	12.414	1.241	2.433
1	2	22.668	19.844	10.447	1.045	2.048
1	3	25.761	22.225	11.917	1.192	2.336
1	4	26.292	19.741	18.548	1.855	3.635
1	1	26.693	21.201	16.266	1.627	3.188

Table 13: Ontology Parameter Pair Evaluation for Parent Degree and Partial Methods.

pd	prop	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
6	3	68.228	50.401	146.016	14.602	28.619
6	5	58.791	42.489	99.818	9.982	19.564
3	7	33.684	31.494	14.059	1.406	2.756
5	2	51.925	39.859	83.542	8.354	16.374
5	4	58.689	44.198	96.799	9.68	18.973
4	3	38.299	36.844	16.539	1.654	3.242
4	5	36.847	35.18	14.424	1.442	2.827
7	2	59.068	56.685	36.559	3.656	7.166
2	6	27.364	26.599	10.903	1.09	2.137
7	4	60.937	54.821	56.003	5.6	10.977
4	6	37.581	36.634	13.504	1.35	2.647
2	3	28.357	25.171	14.654	1.465	2.872
7	7	70.219	49.433	121.617	12.162	23.837
2	5	27.674	28.251	9.216	0.922	1.806
3	2	33.212	31.366	12.59	1.259	2.468
3	4	32.059	30.575	12.696	1.27	2.488
6	6	65.759	48.268	111.414	11.141	21.837
5	7	51.713	44.492	62.414	6.241	12.233
5	6	52.419	40.762	79.452	7.945	15.573
3	3	33.573	33.088	13.607	1.361	2.667
6	7	59.699	42.614	102.338	10.234	20.058
3	5	33.545	31.626	12.255	1.226	2.402
2	2	28.712	25.89	13.264	1.326	2.6
2	4	30.414	30.004	13.075	1.308	2.563
7	6	55.811	57.324	22.147	2.215	4.341
4	7	38.568	35.489	15.839	1.584	3.104
7	3	71.541	59.86	94.393	9.439	18.501
7	5	65.654	49.05	96.622	9.662	18.938
2	7	27.411	26.409	9.913	0.991	1.943
4	2	35.618	33.483	12.856	1.286	2.52
4	4	37.229	35.619	15.414	1.541	3.021
5	3	57.128	41.83	104.371	10.437	20.457
5	5	48.822	39.151	58.671	5.867	11.499
6	2	73.545	46.026	194.934	19.493	38.207
3	6	32.596	30.684	13.305	1.33	2.608
6	4	73.422	44.39	154.568	15.457	30.295
3	1	30.155	26.558	14.643	1.464	2.87
7	1	50.863	40.608	58.155	5.816	11.398
5	1	51.489	32.733	95.36	9.536	18.69
6	1	58.575	39.758	102.669	10.267	20.123
4	1	37.489	30.448	19.729	1.973	3.867
2	1	29.267	23.728	14.493	1.449	2.841
1	7	25.905	21.319	12.591	1.259	2.468
1	2	26.378	20.754	15.755	1.576	3.088
1	4	26.304	23.486	12.127	1.213	2.377
1	3	25.802	22.086	11.821	1.182	2.317
1	5	25.248	20.017	16.569	1.657	3.248
1	6	23.648	20.627	10.668	1.067	2.091
1	1	26.693	21.201	16.266	1.627	3.188

Table 14: Ontology Parameter Pair Evaluation for Parent Degree and Properties and Relationships.

pc	pm	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
7	7	118.577	104.518	125.189	12.519	24.537
6	7	102.134	73.349	183.228	18.323	35.913
3	6	52.801	38.541	108.991	10.899	21.362
5	7	74.886	58.401	109.06	10.906	21.376
2	6	29.781	29.754	9.105	0.911	1.785
4	7	51.857	48.215	54.436	5.444	10.669
4	6	54.788	44.507	61.222	6.122	11.999
2	7	27.922	26.364	10.008	1.001	1.961
5	6	69.756	63.373	83.011	8.301	16.27
3	7	49.274	36.523	83.156	8.316	16.299
6	6	94.261	77.652	129.948	12.995	25.47
7	6	106.067	98.472	102.415	10.241	20.073
2	2	30.356	28.544	12.763	1.276	2.502
4	3	61.945	46.887	100.769	10.077	19.751
6	4	110.254	79.604	201.056	20.106	39.407
3	2	50.495	34.289	101.854	10.185	19.963
5	3	75.036	57.531	110.15	11.015	21.589
7	4	103.834	82.289	99.296	9.93	19.462
4	4	54.377	47.801	72.734	7.273	14.256
2	5	30.034	29.577	13.514	1.351	2.649
6	3	101.817	77.842	163.827	16.383	32.11
5	4	70.655	54.514	90.114	9.011	17.662
3	5	53.041	39.153	100.495	10.05	19.697
7	3	100.736	90.103	91.367	9.137	17.908
3	4	53.955	33.275	133.078	13.308	26.083
5	5	74.865	59.359	108.508	10.851	21.268
7	2	108.046	86.484	117.183	11.718	22.968
2	4	30.881	29.721	13.393	1.339	2.625
4	5	63.293	48.243	101.668	10.167	19.927
6	2	91.115	73.778	107.532	10.753	21.076
5	2	73.041	62.664	86.292	8.629	16.913
3	3	46.162	36.245	71.229	7.123	13.961
7	5	112.583	93.667	146.432	14.643	28.701
4	2	60.056	43.891	107.686	10.769	21.106
2	3	31.787	29.492	16.943	1.694	3.321
6	5	84.745	66.278	92.541	9.254	18.138
3	1	51.959	29.475	133.372	13.337	26.141
2	1	30.122	24.189	14.702	1.47	2.882
6	1	75.045	54.983	110.475	11.047	21.653
7	1	114.146	70.607	212.429	21.243	41.636
4	1	63.127	37.829	138.158	13.816	27.079
5	1	66.557	48.654	93.846	9.385	18.394
1	6	27.018	20.487	15.352	1.535	3.009
1	7	27.344	22.133	17.468	1.747	3.424
1	5	24.433	20.863	12.414	1.241	2.433
1	2	22.668	19.844	10.447	1.045	2.048
1	3	25.761	22.225	11.917	1.192	2.336
1	4	26.292	19.741	18.548	1.855	3.635
1	1	26.693	21.201	16.266	1.627	3.188

Table 15: Ontology Parameter Pair Evaluation for Partial Classes and Partial Methods.

pc	prop	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
4	4	64.604	53.628	89.613	8.961	17.564
3	7	53.816	39.11	94.234	9.423	18.47
4	2	62.736	48.358	95.952	9.595	18.806
2	7	31.565	30.154	14.725	1.473	2.886
5	4	67.746	58.823	73.431	7.343	14.392
5	2	79.82	57.447	153.418	15.342	30.07
6	2	85.489	74.303	83.209	8.321	16.309
6	4	87.984	68.411	117.031	11.703	22.938
7	2	127.325	85.154	226.787	22.679	44.45
7	4	97.544	93.606	67.021	6.702	13.136
6	7	80.759	76.192	68.695	6.869	13.464
7	7	106.447	101.146	96.004	9.6	18.817
3	4	52.667	40.071	85.563	8.556	16.77
4	7	61.214	48.89	81.875	8.188	16.048
3	2	51.906	36.956	101.325	10.132	19.86
5	7	70.902	59.611	81.074	8.107	15.891
2	4	35.963	34.91	16.57	1.657	3.248
2	2	31.794	31.357	13.576	1.358	2.661
2	5	32.078	32.274	12.746	1.275	2.498
5	6	75.645	62.845	101.335	10.134	19.862
2	3	33.807	30.132	17.732	1.773	3.475
4	6	78.05	52.919	166.06	16.606	32.548
3	5	49.401	36.251	79.566	7.957	15.595
3	3	56.354	42.506	101.841	10.184	19.961
7	6	129.187	97.101	206.118	20.612	40.399
6	6	88.955	74.676	105.607	10.561	20.699
7	3	129.251	97.046	225.203	22.52	44.14
7	5	111.586	99.447	120.557	12.056	23.629
6	3	90.485	80.353	97.832	9.783	19.175
6	5	84.426	68.114	93.11	9.311	18.25
5	5	74.743	65.407	98.497	9.85	19.305
2	6	31.914	31.709	10.903	1.09	2.137
5	3	75.226	57.992	113.4	11.34	22.226
3	6	51.689	39.07	84.538	8.454	16.569
4	5	62.859	50.989	91.759	9.176	17.985
4	3	56.5	53.332	68.926	6.893	13.51
3	1	51.959	29.475	133.372	13.337	26.141
2	1	30.122	24.189	14.702	1.47	2.882
6	1	75.045	54.983	110.475	11.047	21.653
7	1	114.146	70.607	212.429	21.243	41.636
4	1	63.127	37.829	138.158	13.816	27.079
5	1	66.557	48.654	93.846	9.385	18.394
1	7	25.905	21.319	12.591	1.259	2.468
1	2	26.378	20.754	15.755	1.576	3.088
1	4	26.304	23.486	12.127	1.213	2.377
1	3	25.802	22.086	11.821	1.182	2.317
1	5	25.248	20.017	16.569	1.657	3.248
1	6	23.648	20.627	10.668	1.067	2.091
1	1	26.693	21.201	16.266	1.627	3.188

Table 16: Ontology Parameter Pair Evaluation for Partial Classes and Properties and Relationships.

pc	d	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
5	5	723.776	684.833	270.678	27.068	53.053
6	7	3697.346	3711.853	685.251	68.525	134.309
4	4	208.515	198.297	96.49	9.649	18.912
7	6	3984.776	3962.702	748.631	74.863	146.732
7	7	6924.042	6780.342	978.16	97.816	191.719
4	5	358.557	340.453	131.841	13.184	25.841
6	6	2264.669	2176.415	571.82	57.182	112.077
5	4	401.835	392.618	161.238	16.124	31.603
2	3	63.199	51.848	90.477	9.048	17.734
5	6	1198.823	1160.685	355.452	35.545	69.668
6	4	724.546	692.515	261.062	26.106	51.168
4	7	922.594	882.259	351.328	35.133	68.86
3	2	67.849	57.748	92.692	9.269	18.168
7	5	2247.46	2179.538	530.212	53.021	103.922
7	4	1104.126	1050.621	385.307	38.531	75.52
3	3	86.46	77.455	74.731	7.473	14.647
4	6	582.12	577.809	204.357	20.436	40.054
6	5	1364.031	1317.485	376.416	37.642	73.778
5	7	1856.813	1801.828	444.379	44.438	87.098
2	2	47.041	37.275	71.117	7.112	13.939
5	3	215.847	217.958	88.16	8.816	17.279
2	6	112.256	96.037	119.195	11.919	23.362
3	7	390.09	363.759	242.064	24.206	47.444
4	2	105.965	75.792	203.741	20.374	39.933
4	3	144.37	133.774	117.442	11.744	23.019
3	6	235.707	214.685	95.972	9.597	18.811
2	7	133.909	137.394	58.779	5.878	11.521
5	2	121.731	107.751	109.843	10.984	21.529
2	5	92.014	74.38	120.877	12.088	23.692
6	2	164.123	150.769	78.692	7.869	15.424
3	4	128.697	119.47	122.62	12.262	24.033
7	3	528.464	456.508	253.462	25.346	49.679
7	2	243.318	250.942	117.287	11.729	22.988
3	5	175.292	177.88	70.087	7.009	13.737
6	3	358.39	339.103	189.506	18.951	37.143
2	4	76.864	60.076	100.767	10.077	19.75
3	1	51.959	29.475	133.372	13.337	26.141
2	1	30.122	24.189	14.702	1.47	2.882
6	1	75.045	54.983	110.475	11.047	21.653
7	1	114.146	70.607	212.429	21.243	41.636
4	1	63.127	37.829	138.158	13.816	27.079
5	1	66.557	48.654	93.846	9.385	18.394
1	2	27.13	22.629	12.824	1.282	2.513
1	3	30.331	26.109	12.726	1.273	2.494
1	4	31.451	27.463	12.169	1.217	2.385
1	5	43.329	31.582	73.906	7.391	14.486
1	7	60.637	41.891	94.306	9.431	18.484
1	6	61.827	34.954	142.661	14.266	27.962
1	1	26.693	21.201	16.266	1.627	3.188

Table 17: Ontology Parameter Pair Evaluation for Partial Classes and Inheritance Depth.

pm	prop	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
5	5	25.846	24.808	10.98	1.098	2.152
5	3	26.855	25.889	12.345	1.235	2.42
6	7	23.697	21.253	10.598	1.06	2.077
7	2	24.875	23.867	9.716	0.972	1.904
4	6	26.26	23.96	14.771	1.477	2.895
7	4	24.65	23.661	12.536	1.254	2.457
2	5	27.079	25.119	15.731	1.573	3.083
2	3	26.198	25.809	11.382	1.138	2.231
3	6	27.662	25.363	13.775	1.378	2.7
2	6	26.843	25.404	13.598	1.36	2.665
3	5	24.182	22.081	9.569	0.957	1.876
3	3	23.624	22.609	9.102	0.91	1.784
5	6	25.574	23.862	12.178	1.218	2.387
6	2	27.455	26.395	14.836	1.484	2.908
6	4	26.002	24.725	13.041	1.304	2.556
4	5	26.261	24.981	11.52	1.152	2.258
7	7	21.773	21.174	9.964	0.996	1.953
4	3	25.363	25.277	10.876	1.088	2.132
4	4	25.101	21.691	14.57	1.457	2.856
4	2	28.252	26.868	13.414	1.341	2.629
7	6	21.853	21.294	8.624	0.862	1.69
6	3	25.775	25.54	11.159	1.116	2.187
5	7	26.468	26.703	11.97	1.197	2.346
6	5	24.329	22.428	10.3	1.03	2.019
3	4	26.339	27.25	7.793	0.779	1.527
3	2	25.626	25.039	10.949	1.095	2.146
2	7	28.711	26.276	13.768	1.377	2.698
3	7	26.595	26.315	10.835	1.083	2.124
2	4	25.888	26.389	9.32	0.932	1.827
2	2	28.334	25.911	15.835	1.583	3.104
4	7	28.194	25.282	13.356	1.336	2.618
7	3	27.097	23.94	14.326	1.433	2.808
7	5	22.778	21.825	10.639	1.064	2.085
5	4	26.341	25.832	10.985	1.099	2.153
6	6	24.531	22.695	12.637	1.264	2.477
5	2	25.334	24.714	13.097	1.31	2.567
6	1	27.018	20.487	15.352	1.535	3.009
7	1	27.344	22.133	17.468	1.747	3.424
5	1	24.433	20.863	12.414	1.241	2.433
2	1	22.668	19.844	10.447	1.045	2.048
3	1	25.761	22.225	11.917	1.192	2.336
4	1	26.292	19.741	18.548	1.855	3.635
1	7	25.905	21.319	12.591	1.259	2.468
1	2	26.378	20.754	15.755	1.576	3.088
1	4	26.304	23.486	12.127	1.213	2.377
1	3	25.802	22.086	11.821	1.182	2.317
1	5	25.248	20.017	16.569	1.657	3.248
1	6	23.648	20.627	10.668	1.067	2.091
1	1	26.693	21.201	16.266	1.627	3.188

Table 18: Ontology Parameter Pair Evaluation for Partial Methods and Properties and Relationships.

pm	d	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
7	5	45.225	39.054	44.713	4.471	8.764
7	4	37.232	36.08	14.626	1.463	2.867
6	2	29.672	29.585	11.01	1.101	2.158
7	6	68.239	45.014	169.693	16.969	33.26
7	7	58.688	49.065	87.439	8.744	17.138
6	3	35.764	34.453	15.336	1.534	3.006
6	7	67.273	51.037	108.012	10.801	21.17
7	3	32.76	30.512	13.926	1.393	2.73
7	2	28.754	29.917	10.674	1.067	2.092
6	6	53.653	44.608	75.446	7.545	14.787
6	4	36.585	34.045	16.0	1.6	3.136
6	5	56.668	38.982	114.176	11.418	22.379
2	3	35.17	35.318	12.532	1.253	2.456
3	7	68.257	52.575	110.642	11.064	21.686
4	4	36.708	36.201	13.737	1.374	2.693
4	5	60.921	43.102	123.304	12.33	24.168
3	6	59.791	44.954	107.46	10.746	21.062
2	2	27.926	25.477	11.295	1.129	2.214
5	3	33.039	33.298	11.206	1.121	2.196
3	4	38.297	35.421	18.245	1.824	3.576
4	7	66.876	48.31	114.415	11.441	22.425
4	6	61.506	43.924	107.178	10.718	21.007
3	5	80.978	39.397	213.301	21.33	41.807
5	2	29.034	26.244	15.449	1.545	3.028
2	5	64.467	41.553	165.221	16.522	32.383
5	6	50.946	44.482	55.925	5.593	10.961
4	2	28.24	25.762	10.889	1.089	2.134
4	3	33.956	31.852	14.747	1.475	2.89
5	7	67.595	52.443	107.304	10.73	21.032
2	4	36.567	34.773	14.5	1.45	2.842
2	6	59.094	44.296	97.932	9.793	19.195
5	5	48.254	41.204	67.854	6.785	13.299
3	2	28.854	28.674	12.027	1.203	2.357
3	3	33.569	32.11	15.315	1.531	3.002
5	4	37.154	34.868	15.667	1.567	3.071
2	7	59.664	49.315	84.493	8.449	16.561
6	1	27.018	20.487	15.352	1.535	3.009
7	1	27.344	22.133	17.468	1.747	3.424
5	1	24.433	20.863	12.414	1.241	2.433
2	1	22.668	19.844	10.447	1.045	2.048
3	1	25.761	22.225	11.917	1.192	2.336
4	1	26.292	19.741	18.548	1.855	3.635
1	2	27.13	22.629	12.824	1.282	2.513
1	3	30.331	26.109	12.726	1.273	2.494
1	4	31.451	27.463	12.169	1.217	2.385
1	5	43.329	31.582	73.906	7.391	14.486
1	7	60.637	41.891	94.306	9.431	18.484
1	6	61.827	34.954	142.661	14.266	27.962
1	1	26.693	21.201	16.266	1.627	3.188

Table 19: Ontology Parameter Pair Evaluation for Partial Methods and Inheritance Depth.

prop	d	ms (mean)	ms (median)	ms (std. deviation)	Std. Error	95
2	7	63.146	52.3	82.43	8.243	16.156
7	2	29.309	27.386	16.941	1.694	3.32
4	7	67.495	51.686	99.718	9.972	19.545
4	6	58.978	46.352	77.706	7.771	15.23
7	3	33.232	30.313	18.079	1.808	3.543
2	6	61.679	53.07	93.041	9.304	18.236
2	4	36.597	36.475	15.192	1.519	2.978
4	4	40.883	38.4	21.243	2.124	4.164
4	5	55.609	45.432	89.331	8.933	17.509
2	5	58.128	41.17	95.563	9.556	18.73
7	4	37.262	35.65	14.518	1.452	2.846
7	5	56.45	44.583	102.746	10.275	20.138
2	2	27.181	27.931	8.777	0.878	1.72
7	7	59.071	51.214	68.786	6.879	13.482
4	2	26.415	25.542	8.589	0.859	1.683
4	3	35.057	33.285	15.394	1.539	3.017
7	6	60.787	47.746	93.184	9.318	18.264
2	3	31.967	30.088	12.995	1.3	2.547
6	6	61.231	49.544	105.751	10.575	20.727
5	3	34.329	33.952	15.701	1.57	3.077
3	3	34.524	35.903	14.306	1.431	2.804
3	2	26.689	26.295	9.37	0.937	1.837
5	2	28.179	27.031	10.047	1.005	1.969
6	7	64.103	43.717	108.144	10.814	21.196
6	5	56.721	41.697	105.055	10.505	20.591
6	4	37.774	37.828	17.949	1.795	3.518
5	5	60.382	43.148	108.771	10.877	21.319
3	5	44.204	38.782	54.618	5.462	10.705
3	4	40.084	37.317	21.303	2.13	4.175
5	4	37.18	33.058	19.242	1.924	3.771
6	3	32.992	32.359	14.417	1.442	2.826
5	6	59.552	45.964	91.98	9.198	18.028
3	6	59.952	46.417	94.945	9.494	18.609
3	7	78.209	52.939	151.93	15.193	29.778
5	7	67.851	56.419	81.511	8.151	15.976
6	2	29.438	27.407	13.845	1.384	2.714
7	1	25.905	21.319	12.591	1.259	2.468
2	1	26.378	20.754	15.755	1.576	3.088
4	1	26.304	23.486	12.127	1.213	2.377
3	1	25.802	22.086	11.821	1.182	2.317
5	1	25.248	20.017	16.569	1.657	3.248
6	1	23.648	20.627	10.668	1.067	2.091
1	2	27.13	22.629	12.824	1.282	2.513
1	3	30.331	26.109	12.726	1.273	2.494
1	4	31.451	27.463	12.169	1.217	2.385
1	5	43.329	31.582	73.906	7.391	14.486
1	7	60.637	41.891	94.306	9.431	18.484
1	6	61.827	34.954	142.661	14.266	27.962
1	1	26.693	21.201	16.266	1.627	3.188

Table 20: Ontology Parameter Pair Evaluation for Properties and Relationships and Inheritance Depth.

BIBLIOGRAPHY

[1] Ben Adida, Ivan Herman, Manu Sporny, and Mark Birbeck. *RDFa 1.1 Primer - Third Edition*. W3C Note. Mar. 2015. URL: http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/.

[2] Patrick Aichroth et al. "MICO - Media in Context." In: *2015 IEEE International Conference on Multimedia & Expo Workshops, ICME Workshops 2015, Turin, Italy, June 29 - July 3, 2015*. IEEE Computer Society, 2015, pp. 1–4.

[3] Patrick Aichroth, Henrik Björklund, Johanna Björklund, Kai Schlegel, Thomas Kurz, and Antonio Perez. *Volume 3: MICO Enabling Technology Modules: Version 1*. Deliverable, Technical Report. Media in Context - MICO, Nov. 2015.

[4] Patrick Aichroth, Marcel Sieland, Luca Cuccovillo, and Thomas Köllmer. "The MICO Broker: An Orchestration Framework for Linked Data Extractors." In: *Joint Proceedings of the 4th International Workshop on Linked Media and the 3rd Developers Hackshop co-located with the 13th Extended Semantic Web Conference ESWC 2016, Heraklion, Crete, Greece, May 30, 2016*. 2016.

[5] Patrick Aichroth, Johanna Björklund, Emanuel Berndl, Thomas Kurz, and Thomas Köllmer. *Volume 5: MICO Enabling Technology Modules - Final Version*. Deliverable, Technical Report. Media in Context - MICO, Apr. 2016.

[6] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. "Wireless Sensor Networks: A Survey." In: *Computer Networks* 38.4 (2002), pp. 393–422.

[7] Richard Arndt, Raphaël Troncy, Steffen Staab, Lynda Hardman, and Miroslav Vacura. "COMM: Designing a Well-Founded Multimedia Ontology for the Web." In: *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*. Ed. by Karl Aberer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 30–43.

[8] Richard Arndt, Raphaël Troncy, Steffen Staab, and Lynda Hardman. "Comm: A Core Ontology for Multimedia Annotation." In: *Handbook on Ontologies*. Springer, 2009.

[9] Ahmad Assaf, Aline Senart, and Raphaël Troncy. "Roomba: Automatic Validation, Correction and Generation of Dataset Metadata." In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15 Companion. Florence, Italy: ACM, 2015, pp. 159–162.

[10] Ahmad Assaf, Raphaël Troncy, and Aline Senart. "What's up LOD Cloud?" In: *The Semantic Web: ESWC 2015 Satellite Events*. Ed. by Fabien Gandon, Christophe Guéret, Serena Villata, John Breslin, Catherine Faron-Zucker, and Antoine Zimmermann. Cham: Springer International Publishing, 2015, pp. 247–254.

[11] Sören Auer. "Creating Knowledge out of Interlinked Data: Making the Web a Data Washing Machine." In: *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*. WIMS '11. Sogndal, Norway: ACM, 2011, 4:1–4:8.

[12] Sören Auer, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. "Introduction to Linked Data and Its Lifecycle on the Web." In: *Proceedings of the 7th International Conference on Reasoning Web: Semantic Technologies for the Web of Data*. RW'11. Galway, Ireland: Springer-Verlag, 2011, pp. 1–75.

[13] Sören Auer et al. "Managing the Life-Cycle of Linked Data with the LOD2 Stack." In: *The Semantic Web – ISWC 2012*. Ed. by Philippe Cudré-Mauroux et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–16.

[14] Franz Baader, Ian Horrocks, and Ulrike Sattler. "Description Logics as Ontology Languages for the Semantic Web." In: *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*. Ed. by Dieter Hutter and Werner Stephan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 228–248.

[15] Steve Battle, David Wood, James Leigh, and Luke Ruth. "The Callimachus Project: RDFa As a Web Template Language." In: *Proceedings of the Third International Conference on Consuming Linked Data - Volume 905*. COLD'12. Boston, MA, 2012, pp. 1–14.

[16] Christian Bauer and Gavin King. *Hibernate in Action*. Greenwich, CT: Manning, 2005.

[17] Sean Bechhofer and Alistair Miles. *SKOS Simple Knowledge Organization System Reference*. W3C Recommendation. Aug. 2009. URL: http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[18] Diana Benito-Osorio, Marta Peris-Ortiz, Carlos Rueda Armengot, and Alberto Colino. "Web 5.0: the future of emotional competences in higher education." In: *Global Business Perspectives* 1.3 (2013), pp. 274–287.

[19] Ansgar Bernardi, Harald Holz, Heiko Maus, and Ludger van Elst. "Komplexe Arbeitswelten in der Wissensgesellschaft." In: *Semantic Web: Wege zur vernetzten Wissensgesellschaft*. Ed. by Tassilo Pellegrini and Andreas Blumauer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 27–45.

[20] Emanuel Berndl, Kai Schlegel, and Andreas Eisenkolb. *MICO Metadata Model - Terms*. Model Specification Recommendation. Oct. 2016. URL: https://mico-project.bitbucket.io/vocabs/mmmterms/2.0/documentation/.

[21] Emanuel Berndl, Kai Schlegel, Andreas Eisenkolb, Thomas Weißgerber, and Harald Kosch. "Anno4j - Idiomatic Access to the W3C Web Annotation Data Model." In: *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*. Ed. by Harald Sack, Giuseppe Rizzo, Nadine Steinmetz, Dunja Mladenic, Sören Auer, and Christoph Lange. Vol. 9989. Lecture Notes in Computer Science. 2016, pp. 257–270.

[22] Emanuel Berndl, Kai Schlegel, Andreas Eisenkolb, and Harald Kosch. "Idiomatic Persistence and Querying for the W3C Web Annotation Data Model." In: *Joint Proceedings of the 4th International Workshop on Linked Media and the 3rd Developers Hackshop co-located with the 13th Extended Semantic Web Conference ESWC 2016, Heraklion, Crete, Greece, May 30, 2016*. Ed. by Raphaël Troncy, Ruben Verborgh, Lyndon J. B. Nixon, Thomas Kurz, Kai Schlegel, and Miel Vander Sande. Vol. 1615. CEUR Workshop Proceedings. CEUR-WS.org, 2016.

[23] Emanuel Berndl, Kai Schlegel, Andreas Eisenkolb, and Thomas Weißgerber. *MICO Metadata Model*. Model Specification Recommendation. Oct. 2016. URL: https://mico-project.bitbucket.io/vocabs/mmm/2.0/documentation/.

[24] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. RFC Editor, 2005, pp. 1–60. URL: https://tools.ietf.org/rfc/rfc3986.txt.

[25] T. Berners-Lee, L. Masinter, and M. McCahill. *Uniform Resource Locators (URL)*. RFC 1738. RFC Editor, 1994, pp. 1–24. URL: https://www.ietf.org/rfc/rfc1738.txt.

[26] Tim Berners-Lee. *Linked Data*. Website. Last visited 03/12/2018. 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html.

[27] Tim Berners-Lee, James Hendler, Ora Lassila, et al. "The Semantic Web." In: *Scientific American* 284.5 (2001), pp. 28–37.

[28] Tim Berners-Lee et al. *Semantic Web Road Map*. Website. Last visited 03/12/2018. 1998. URL: https://www.w3.org/DesignIssues/Semantic.html.

[29] Tim Berners-Lee et al. *What the Semantic Web can represent*. Website. Last visited 03/12/2018. 1998. URL: https://www.w3.org/DesignIssues/RDFnot.html.

[30] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked Data - The Story So Far." English. In: *International Journal on Semantic Web and Information Systems* 5.3 (2009), pp. 1–22.

[31] Christian Bizer and Andreas Schultz. "The Berlin SPARQL Benchmark." In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 5.2 (2009), pp. 1–24.

[32] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. "Linked Data on the Web (LDOW2008)." In: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. Beijing, China: ACM, 2008, pp. 1265–1266.

[33] Christian Bizer, Kai Eckert, Robert Meusel, Hannes Mühleisen, Michael Schuhmacher, and Johanna Völker. "Deployment of RDFa, Microdata, and Microformats on the Web - A Quantitative Analysis." In: *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*. Ed. by Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz. Vol. 8219. Lecture Notes in Computer Science. Springer, 2013, pp. 17–32.

[34] Dublin Core Usage Board. *DCMI Metadata Terms*. DCMI Recommendation. Dublin Core Metadata Initiative, June 2012. URL: http://dublincore.org/documents/dcmi-terms/.

[35] Barry W. Boehm. "Software Engineering: As it is." In: *Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, September 1979*. 1979, pp. 11–21.

[36] Harold Boley, Said Tabet, and Gerd Wagner. "Design Rationale of RuleML: A Markup Language for Semantic Web Rules." In: *Proceedings of the First International Conference on Semantic Web Working*. SWWS'01. California: CEUR-WS.org, 2001, pp. 381–401.

[37] Dan Brickley and Ramanathan Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[38] Dan Brickley and Libby Miller. *FOAF Vocabulary Specification*. Technical Report. xmlns, Jan. 2014. URL: http://xmlns.com/foaf/spec/20140114.html.

[39] Mark Butler. "Barriers to Real World Adoption of Semantic Web Technologies." In: *Proceedings of the 2002 International Conference on Semantic Web, RDF, and XML*. 2002, p. 7.

[40] Lorna M. Campbell and Sheila MacNeill. "The Semantic Web, Linked and Open Data." In: *A Briefing Paper of the Publications from the Centre for Educational Technology, Interoperability and Standards.* (2010).

[41] J. Cardoso. "The Semantic Web Vision: Where Are We?" In: *IEEE Intelligent Systems* 22.5 (2007), pp. 84–88.

[42] Gavin Carothers. *RDF 1.1 N-Quads*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-n-quads-20140225/.

[43] Gavin Carothers and Andy Seaborne. *RDF 1.1 N-Triples*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-n-triples-20140225/.

[44] Gavin Carothers and Andy Seaborne. *RDF 1.1 TriG*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-trig-20140225/.

[45] Scott Chacon and Ben Straub. *Pro Git*. 2nd Edition. Berkely, CA, USA: Apress, 2014.

[46] Shih-Fu Chang, T. Sikora, and A. Purl. "Overview of the MPEG-7 Standard." In: *IEEE Transactions on Circuits and Systems for Video Technology* 11.6 (2001), pp. 688–695.

[47] Peter Pin-Shan Chen. "The Entity-Relationship Model—Toward a Unified View of Data." In: *Readings in Artificial Intelligence and Databases*. Ed. by John Mylopolous and Michael Brodie. San Francisco (CA): Morgan Kaufmann, 1989, pp. 98 –111.

[48] L. Cheng, S. Kotoulas, T. Ward, and G. Theodoropoulos. "Runtime Characterization of Triple Stores." In: *2012 IEEE 15th International Conference on Computational Science and Engineering*. 2012, pp. 66–73.

[49] James Clark and Steve DeRose. *XML Path Language XPath*. W3C Recommendation. W3C, Nov. 1999. URL: https://www.w3.org/TR/1999/REC-xpath-19991116/.

[50] Daniel D Corkill. "Blackboard systems." In: *AI expert* 6.9 (1991).

[51] Carlos Coronel and Steven Morris. *Database Systems: Design, Implementation, & Management*. 12th ed. Cengage Learning, 2016.

[52] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[53] *DBpedia Facts and Figures*. Last visited 03/12/2018. URL: http://wiki.dbpedia.org/about/facts-figures.

[54] DCMI Usage Board. *DCMI Metadata Terms*. DCMI Recommendation. Dublin Core Metadata Initiative, 2006. URL: http://dublincore.org/documents/2006/12/18/dcmi-terms/.

[55] Neil F. Day. "MPEG-7 Applications." In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*. Vol. 4209. proc-spie. Mar. 2001, pp. 39–47.

[56] Ben De Meester, Ruben Verborgh, Pieter Pauwels, Wesley De Neve, Erik Mannens, and Rik Van de Walle. "Towards Robust and Reliable Multimedia Analysis Through Semantic Integration of Services." In: *Multimedia Tools and Applications* 75.22 (2016), pp. 14019–14038.

[57] Li Ding and Tim Finin. "Characterizing the Semantic Web on the Web." In: *The Semantic Web - ISWC 2006*. Ed. by Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora M. Aroyo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 242–257.

[58] Martin Doerr. "The CIDOC Conceptual Reference Module: An Ontological Approach to Semantic Interoperability of Metadata." In: *AI Magazine* 24.3 (2003), p. 75.

[59] Mario Döller and Harald Kosch. "MPEG-7 Multimedia Data Cartridge." In: *Multimedia Computing and Networking 2003*. Vol. 5019. International Society for Optics and Photonics. 2003, pp. 126–138.

[60] M. Duerst and M. Suignard. *Internationalized Resource Identifiers (IRIs)*. RFC 3987. RFC Editor, 2005, pp. 1–46. URL: https://tools.ietf.org/rfc/rfc3987.txt.

[61] Kévin Dunglas. *Persistence in PHP with the Doctrine ORM*. Birmingham: Packt Publishing Ltd., 2013.

[62] Daniel Eißing, Ansgar Scherp, and Carsten Saathoff. "Integration of Existing Multimedia Metadata Formats and Metadata Standards in the M3O." In: *Semantic Multimedia - 5th International Conference on Semantic and Digital Media Technologies, SAMT 2010. Saarbrücken, Germany, December 1-3, 2010. Revised Selected Papers*. Ed. by Thierry Declerck, Michael Granitzer, Marcin Grzegorzek, Massimo Romanelli, Stefan M. Rüger, and Michael Sintek. Vol. 6725. Lecture Notes in Computer Science. Springer, 2010, pp. 48–63.

[63] L. van Elst, F. R. Aschoff, A. Bernardi, and S. Schwarz. "Weakly-structured Workflows for Knowledge-intensive Tasks: An Experimental Evaluation." In: *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* Pp. 340–345.

[64] Wolfgang Ertel. "First-order Predicate Logic." In: *Introduction to Artificial Intelligence*. Cham: Springer International Publishing, 2017, pp. 39–64.

[65] Wolfgang Ertel. *Introduction to Artificial Intelligence, Second Edition*. Undergraduate Topics in Computer Science. Springer, 2017.

[66] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. "Object Detection with Discriminatively Trained Part-Based Models." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645.

[67] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Pearson Education, 1994.

[68] Mike Graves, Adam Constabaris, and Dan Brickley. "FOAF: Connecting People on the Semantic Web." In: *Cataloging & Classification Quarterly* 43.3-4 (2007), pp. 191–202.

[69] Jim Gray et al. "The Transaction Concept: Virtues and Limitations." In: *VLDB*. Vol. 81. 1981, pp. 144–154.

[70] W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. Technical Report. W3C, Dec. 2012. URL: http://www.w3.org/TR/2012/REC-owl2-overview-20121211/.

[71] W3C SPARQL Working Group. *SPARQL 1.1 Overview*. W3C Recommendation. W3C, Mar. 2013. URL: http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[72] Ramanathan Guha and Dan Brickley. *RDF Schema 1.1*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[73] Theo Haerder and Andreas Reuter. "Principles of Transaction-oriented Database Recovery." In: *ACM Comput. Surv.* 15.4 (Dec. 1983), pp. 287–317.

[74] Maurice Howard Halstead. *Elements of Software Science*. Vol. 7. Elsevier New York, 1977.

[75] Jonathon S. Hare, Paul H. Lewis, Peter G. B. Enser, and Christine J. Sandom. "Mind the Gap: Another Look at the Problem of the Semantic Gap in Image Retrieval." In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*. Vol. 6073. Jan. 2006.

[76] Frank van Harmelen and Deborah McGuinness. *OWL Web Ontology Language Overview*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[77] Olaf Hartig. "How Caching Improves Efficiency and Result Completeness for Querying Linked Data." In: *WWW2011 Workshop on Linked Data on the Web, Hyderabad, India, March 29, 2011*. 2011.

[78] Michael Hausenblas, Raphaël Troncy, Tobias Bürger, and Yves Raimond. "Interlinking Multimedia: How to Apply Linked Data Principles to Multimedia Fragments." In: *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009*.

[79] Patrick Hayes. *RDF Semantics*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

[80] Patrick Hayes and Peter Patel-Schneider. *RDF 1.1 Semantics*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[81] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[82] Sallie M. Henry and Dennis G. Kafura. "Software Structure Metrics Based on Information Flow." In: *IEEE Trans. Software Eng.* 7.5 (1981), pp. 510–518.

[83] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, and York Sure. *Semantic Web: Grundlagen*. Springer-Verlag, 2007.

[84] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. "Weaving the Pedantic Web." In: *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*. Ed. by Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas. Vol. 628. CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[85] Ian Horrocks. "DAML+OIL: A Description Logic for the Semantic Web." In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 25(1) (Apr. 2002), pp. 4–9.

[86] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." In: *W3C Member submission* 21 (2004), p. 79.

[87] C.-W Hsu, C.-C Chang, and C.-J Lin. *A Practical Guide to Support Vector Classification*. Tech. rep. Jan. 2003.

[88] Wei Hu, Cunxin Jia, Lei Wan, Liang He, Lixia Zhou, and Yuzhong Qu. "CAMO: Integration of Linked Open Data for Multimedia Metadata Enrichment." In: *The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. Ed. by Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble. Cham: Springer International Publishing, 2014, pp. 1–16.

[89] Jane Hunter. "Adding Multimedia to the Semantic Web: Building an MPEG-7 Ontology." In: *Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001*. Ed. by Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness. 2001, pp. 261–283.

[90] *ISO 10646:2014(E): Information Technology – Universal Coded Character Set (UCS)*. Standard. International Organization for Standardization, Mar. 2014.

[91] *ISO 21127:2006 - Information and Documentation – A Reference Ontology for the Interchange of Cultural Heritage Information*. Standard. International Organization for Standardization, Sept. 2006.

[92] *ISO 21127:2014 - Information and Documentation – A Reference Ontology for the Interchange of Cultural Heritage Information*. Standard. International Organization for Standardization, Sept. 2014.

[93] *ISO/IEC 17203:2017 (INCITS 469:2015) Information Technology – Open Virtualization Format (OVF) Specification*. Standard. International Organization for Standardization, Sept. 2017.

[94] Ian Jacobs and Norman Walsh. *Architecture of the World Wide Web, Volume One*. W3C Recommendation. W3C, Dec. 2004. URL: http://www.w3.org/TR/2004/REC-webarch-20041215/.

[95] Dennis G. Kafura and Geereddy R. Reddy. "The Use of Software Complexity Metrics in Software Maintenance." In: *IEEE Trans. Software Eng.* 13.3 (1987), pp. 335–343.

[96] E. Kasutani and A. Yamada. "The MPEG-7 Color Layout Descriptor: A Compact Image Feature Description for High-speed Image/Video Segment Retrieval." In: *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*. Vol. 1. 2001, 674–677 vol.1.

[97] Isaak Kavasidis, Simone Palazzo, Roberto Di Salvo, Daniela Giordano, and Concetto Spampinato. "An Innovative Web-based Collaborative Platform for Video Annotation." In: *Multimedia Tools and Applications* 70.1 (2014), pp. 413–432.

[98] Taghi M. Khoshgoftaar and John C. Munson. "Predicting Software Development Errors Using Software Complexity Metrics." In: *IEEE Journal on Selected Areas in Communications* 8.2 (1990), pp. 253–261.

[99] Graham Klyne and Jeremy Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[100] Johannes Koch, Philip Ackermann, and Carlos A. Velasco. *Representing Content in RDF 1.0*. W3C Note. W3C, Feb. 2017. URL: https://www.w3.org/TR/2017/NOTE-Content-in-RDF10-20170202/.

[101] Thomas Köllmer, Emanuel Berndl, Thomas Weißgerber, Patrick Aichroth, and Harald Kosch. "A Workflow for Cross Media Recommendations based on Linked Data Analysis." In: *Joint Proceedings of the 4th International Workshop on Linked Media and the 3rd Developers Hackshop co-located with the 13th Extended Semantic Web Conference ESWC 2016, Heraklion, Crete, Greece, May 30, 2016*. Ed. by Raphaël Troncy, Ruben Verborgh, Lyndon J. B. Nixon, Thomas Kurz, Kai Schlegel, and Miel Vander Sande. Vol. 1615. CEUR Workshop Proceedings. CEUR-WS.org, 2016.

[102] H. Kosch, L. Boszormenyi, M. Doller, M. Libsie, P. Schojer, and A. Kofler. "The Life Cycle of Multimedia Metadata." In: *IEEE MultiMedia* 12.1 (2005), pp. 80–86.

[103] Harald Kosch. *Distributed Multimedia Database Technologies Supported by MPEG-7 and MPEG-21*. CRC Press, 2003.

[104] Thomas Kurz, Kai Schlegel, and Harald Kosch. "Enabling Access to Linked Media with SPARQL-MM." In: *Proceedings of the 24nd international conference on World Wide Web (WWW2015) companion (LIME15)*. 2015.

[105] Markus Lanthaler, Gregg Kellogg, and Manu Sporny. *JSON-LD 1.0*. W3C Recommendation. W3C, Jan. 2014. URL: http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[106] O. Lassila. "Web Metadata: A Matter of Semantics." In: *IEEE Internet Computing* 2.4 (1998), pp. 30–37.

[107] Didier Le Gall. "MPEG: A Video Compression Standard for Multimedia Applications." In: *Commun. ACM* 34.4 (Apr. 1991), pp. 46–58.

[108] P. M. Lerman. "Fitting Segmented Regression Models by Grid Search." In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 29.1 (1980), pp. 77–84.

[109] Qin Li, Zhao Lu, Yunfei Yu, and Lu Liang. "Multimedia Ontology Modeling: An Approach Based on MPEG-7." In: *2011 3rd International Conference on Advanced Computer Control*. 2011, pp. 351–356.

[110] Sheng Liang. *Java Native Interface: Programmer's Guide and Reference*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[111] Eve Maler, Michael Sperberg-McQueen, Jean Paoli, François Yergeau, and Tim Bray. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. W3C, Nov. 2008. URL: http://www.w3.org/TR/2008/REC-xml-20081126/.

[112] Ashok Malhotra, David Peterson, Sandy Gao, Paul V. Biron, Michael Sperberg-McQueen, and Henry Thompson. *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. W3C Recommendation. W3C, Apr. 2012. URL: http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/.

[113] Frank Manola and Eric Miller. *RDF Primer*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[114] D. Marr. "Artificial Intelligence—A Personal View." In: *Artificial Intelligence* 9.1 (1977), pp. 37 –48.

[115] J. M. Martinez, R. Koenen, and F. Pereira. "MPEG-7: The Generic Multimedia Content Description Standard, Part 1." In: *IEEE MultiMedia* 9.2 (2002), pp. 78–87.

[116] Thomas J. McCabe. "A Complexity Measure." In: *IEEE Trans. Software Eng.* 2.4 (1976), pp. 308–320.

[117] Carma L. McClure. "A Model for Program Complexity Analysis." In: *Proceedings of the 3rd International Conference on Software Engineering, Atlanta, Georgia, USA, May 10-12*. 1978, pp. 149–157.

[118] Noah Mendelsohn. *The Self-Describing Web*. Technical Report. W3C, Feb. 2009. URL: https://www.w3.org/2001/tag/doc/selfDescribingDocuments-2009-02-07.html.

[119] Pablo N Mendes, Max Jakob, and Christian Bizer. "DBpedia: A Multilingual Cross-domain Knowledge Base." In: *LREC*. 2012, pp. 1813–1817.

[120] Alistair Miles, Brian Matthews, Michael D. Wilson, and Dan Brickley. "SKOS Core: Simple Knowledge Organisation for the Web." In: *Vocabularies in Practice: Proceedings of the 2005 International Conference on Dublin Core and Metadata Applications, DC 2005, Madrid, Spain, September 12-15, 2005*. Ed. by Thomas Baker and Eva Méndez. Dublin Core Metadata Initiative, 2005, pp. 3–10.

[121] Mark A. Musen. "The Protégé Project: A Look Back and a Look Forward." In: *AI Matters* 1.4 (June 2015), pp. 4–12.

[122] Axel-Cyrille Ngonga Ngomo, Sören Auer, Jens Lehmann, and Amrapali Zaveri. "Introduction to Linked Data and Its Lifecycle on the Web." In: *Reasoning Web. Reasoning on the Web in the Big Data Era: 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*. Cham: Springer International Publishing, 2014, pp. 1–99.

[123] Elizabeth J. O'Neil. "Object/Relational Mapping 2008: Hibernate and the Entity Data Model (Edm)." In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, pp. 1351–1356.

[124] Martin O'connor, Holger Knublauch, Samson Tu, Benjamin Grosof, Mike Dean, William Grosso, and Mark Musen. "Supporting Rule System Interoperability on the Semantic Web with SWRL." In: *The Semantic Web–ISWC 2005* (2005), pp. 974–986.

[125] Guy Paré. "Investigating Information Systems with Positivist Case Research." In: *The Communications of the Association for Information Systems* 13.1 (2004), pp. 233–364.

[126] Alan P. Parkes. "First Order Predicate Logic." In: *Introduction to Languages, Machines and Logic: Computable Languages, Abstract Machines and Formal Logic*. London: Springer London, 2002, pp. 291–306.

[127] Karan Patel. "Incremental Journey for World Wide Web: Introduced with Web 1.0 to Recent Web 5.0–A Survey Paper." In: *International Journal of Advanced Research in Computer Science and Software Engineering* 3.10 (2013).

[128] Pieter Pauwels and Rens Bod. "Including the Power of Interpretation Through a Simulation of Peirce's Process of Inquiry." In: *Literary and Linguistic Computing* 28.3 (2013), pp. 452–460.

[129] Christian Petersohn. "Temporal Video Segmentation." PhD thesis. Berlin Institute of Technology, 2010.

[130] Ajinkya Prabhune, Rainer Stotzka, Vaibhav Sakharkar, Jürgen Hesser, and Michael Gertz. "MetaStore: An Adaptive Metadata Management Framework for Heterogeneous Metadata Models." In: *Distributed and Parallel Databases* 36.1 (2018), pp. 153–194.

[131] Eric Prud'hommeaux and Gavin Carothers. *RDF 1.1 Turtle*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-turtle-20140225/.

[132] Matthias Quasthoff. "Effizientes Entwickeln von Semantic-Web-Software mit Object Triple Mapping." PhD thesis. University of Potsdam, 2011.

[133] Matthias Quasthoff and Christoph Meinel. "Semantic Web Admission Free–Obtaining RDF and OWL Data from Application Source Code." In: *Proc. of the 4th Int. Workshop on Semantic Web Enabled Software Engineering*. 2008, pp. 17–25.

[134] Matthias Quasthoff, Harald Sack, and Christoph Meinel. "How to Simplify Building Semantic Web Applications." In: *Proceedings of the 5th International Workshop on Semantic Web Enabled Software Engineering* (2009), pp. 45–57.

[135] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788.

[136] George Reese. *Database Programming with JDBC and JAVA*. O'Reilly Media, Inc., 2000.

[137] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)* Pearson Education, 2010.

[138] Carsten Saathoff and Ansgar Scherp. "M3O: The Multimedia Metadata Ontology." In: *Proceedings of the Workshop on Semantic Multimedia Database Technologies, 10th International Workshop of the Multimedia Metadata Community (SeMuDaTe 2009), Graz, Austria*. 2009.

[139] Carsten Saathoff and Ansgar Scherp. "Unlocking the Semantics of Multimedia Presentations in the Web with the Multimedia Metadata Ontology." In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 831–840.

[140] Satya Sahoo, Timothy Lebo, and Deborah McGuinness. *PROV-O: The PROV Ontology*. W3C Recommendation. W3C, Apr. 2013. URL: http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[141] Robert Sanderson. *Web Annotation Protocol*. W3C Recommendation. W3C, Feb. 2017. URL: https://www.w3.org/TR/2017/REC-annotation-protocol-20170223/.

[142] Robert Sanderson, Paolo Ciccarese, and Herbert Van de Sompel. *OADM Open Annotation Data Model*. Community Draft. W3C, Feb. 2013. URL: http://www.openannotation.org/spec/core/.

[143] Robert Sanderson, Paolo Ciccarese, and Benjamin Young. *WADM Web Annotation Data Model*. W3C Recommendation. W3C, Feb. 2017. URL: https://www.w3.org/TR/2017/REC-annotation-model-20170223/.

[144] Robert Sanderson, Benjamin Young, and Paolo Ciccarese. *Web Annotation Vocabulary*. W3C Recommendation. W3C, Feb. 2017. URL: https://www.w3.org/TR/2017/REC-annotation-vocab-20170223/.

[145] Sebastian Schaffert, Christoph Bauer, Thomas Kurz, Fabian Dorschel, Dietmar Glachs, and Manuel Fernandez. "The Linked Media Framework: Integrating and Interlinking Enterprise Media Content and Data." In: *Proceedings of the 8th International Conference on Semantic Systems*. I-SEMANTICS '12. Graz, Austria: ACM, 2012, pp. 25–32.

[146] Bernhard Schandl, Bernhard Haslhofer, Tobias Bürger, Andreas Langegger, and Wolfgang Halb. "Linked Data and Multimedia: The State of Affairs." In: *Multimedia Tools and Applications* 59.2 (2012), pp. 523–556.

[147] Ansgar Scherp, Daniel Eißing, and Carsten Saathoff. "A Method for Integrating Multimedia Metadata Standards and Metadata Formats with the Multimedia Metadata Ontology." In: *Int. J. Semantic Computing* 6.1 (2012), pp. 25–50.

[148] Manfred Schied, Anton Wolff, and Köstlbacher Christian. "Connecting Semantic MediaWiki to different Triple Stores Using RDF2Go." In: *Fifth Workshop on Semantic Wikis Linking Data and People 7th Extended Semantic Web Conference Hersonissos, Crete, Greece, June 2010*. 2010, pp. 159–163.

[149] Kai Schlegel, Emanuel Berndl, Michael Granitzer, Harald Kosch, and Thomas Kurz. "A Platform for Contextual Multimedia Data: Towards a Unified Metadata Model and Querying." In: *Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business, I-KNOW '15, Graz, Austria, October 21-23, 2015*. Ed. by Stefanie N. Lindstaedt, Tobias Ley, and Harald Sack. ACM, 2015, 1:1–1:8.

[150] Guus Schreiber and Fabien Gandon. *RDF 1.1 XML Syntax*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[151] Guus Schreiber and Yves Raimond. *RDF 1.1 Primer*. W3C Note. W3C, June 2014. URL: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[152] Andy Seaborne. *SPARQL 1.1 Property Paths*. W3C Working Draft. W3C, Jan. 2010. URL: http://www.w3.org/TR/2010/WD-sparql11-property-paths-20100126/.

[153] Julian Seidenberg and Alan L. Rector. "Web Ontology Segmentation: Analysis, Classification and Use." In: *Proceedings of the 15th International Conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*. 2006, pp. 13–22.

[154] N. Shadbolt, T. Berners-Lee, and W. Hall. "The Semantic Web Revisited." In: *IEEE Intelligent Systems* 21.3 (2006), pp. 96–101.

[155] Amit P. Sheth and Wolfgang Klas, eds. *Multimedia Data Management: Using Metadata to Integrate and Apply Digital Media*. McGraw-Hill, 1998.

[156] T. Sikora. "The MPEG-7 Visual Standard for Content Description - An Overview." In: *IEEE Transactions on Circuits and Systems for Video Technology* 11.6 (2001), pp. 696–702.

[157] Leslie F. Sikos. "The Semantic Gap." In: *Description Logics in Multimedia Reasoning*. Cham: Springer International Publishing, 2017, pp. 51–66.

[158] Evren Sirin and Bijan Parsia. "Pellet: An OWL DL Reasoner." In: *Proceedings of the 2004 International Workshop on Description Logics (DL2004), Whistler, British Columbia, Canada, June 6-8, 2004*. Ed. by Volker Haarslev and Ralf Möller. Vol. 104. CEUR Workshop Proceedings. CEUR-WS.org, 2004.

[159] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms." In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.* Ed. by Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger. 2012, pp. 2960–2968.

[160] Concetto Spampinato, Simone Palazzo, Bastian Boom, Jacco van Ossenbruggen, Isaak Kavasidis, Roberto Di Salvo, Fang-Pang Lin, Daniela Giordano, Lynda Hardman, and Robert B. Fisher. "Understanding Fish Behavior During Typhoon Events in Real-life Underwater Environments." In: *Multimedia Tools and Applications* 70.1 (2014), pp. 199–236.

[161] Lutz Suhrbier, Wolf-Henning Kusber, Okka Tschöpe, Anton Güntsch, and Walter G. Berendsohn. "AnnoSys - Implementation of a Generic Annotation System for Schema-based Data Using the Example of Biodiversity Collection Data." In: *Database* 2017 (2017), bax018.

[162] Robert Endre Tarjan. "Depth-First Search and Linear Graph Algorithms." In: *SIAM J. Comput.* 1.2 (1972), pp. 146–160.

[163] Jeni Tennison. *Best Practices for Fragment Identifiers and Media Type Definitions*. W3C Working Draft. W3C, Oct. 2012. URL: http://www.w3.org/TR/2012/WD-fragid-best-practices-20121025/.

[164] Ronald A. Thisted. *Elements of Statistical Computing: Numerical Computation*. London, UK: Chapman & Hall, Ltd., 1988.

[165] Raphaël Troncy, Silvia Pfeiffer, Davy Van Deursen, and Erik Mannens. *Media Fragments URI 1.0 (basic)*. W3C Recommendation. W3C, Sept. 2012. URL: http://www.w3.org/TR/2012/REC-media-frags-20120925/.

[166] J. Utecht and M. Brochhausen. "Measuring the Usability of Triple Stores for Knowledge Management on Trauma Care Organizations." In: *CEUR workshop proceedings*. Vol. 1546. 2015, pp. 241–242.

[167] M. Vacura, V. Svatek, C. Saathoff, T. Franz, and R. Troncy. "Describing Low-level Image Features Using the COMM Ontology." In: *2008 15th IEEE International Conference on Image Processing*. 2008, pp. 49–52.

[168] Ruben Verborgh, Davy Van Deursen, Erik Mannens, Chris Poppe, and Rik Van de Walle. "Enabling Context-aware Multimedia Annotation by a Novel Generic Semantic Problem-solving Platform." In: *Multimedia Tools Appl.* 61.1 (2012), pp. 105–129.

[169] Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Jos De Roo, Rik Van de Walle, and Joaquim Gabarro Valles. "Capturing the Functionality of Web Services with Functional Descriptions." In: *Multimedia Tools and Applications* 64.2 (2013), pp. 365–387.

[170] Ruben Verborgh, Thomas Steiner, Rik Van de Walle, and Joaquim Gabarro. "Linked Data and Linked APIs: Similarities, Differences, and Challenges." In: *The Semantic Web: ESWC 2012 Satellite Events*. Ed. by Elena Simperl, Barry Norton, Dunja Mladenic, Emanuele Della Valle, Irini Fundulaki, Alexandre Passant, and Raphaël Troncy. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 272–284.

[171] Max Völkel and York Sure. "RDFReactor - From Ontologies to Programmatic Data Access." In: *Poster Proceedings of the Fourth International Semantic Web Conference*. 2005, pp. 55–65.

[172] Taowei David Wang, Bijan Parsia, and James A. Hendler. "A Survey of the Web Ontology Landscape." In: *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*. Ed. by Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo. Vol. 4273. Lecture Notes in Computer Science. Springer, 2006, pp. 682–694.

[173] Taowei David Wang, Bijan Parsia, and James Hendler. "A Survey of the Web Ontology Landscape." In: *The Semantic Web - ISWC 2006*. Ed. by Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora M. Aroyo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 682–694.

[174] Roy Want. "Near Field Communication." In: *IEEE Pervasive Computing* 10.3 (2011), pp. 4–7.

[175] Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[176] Ken Wenzel. "Komma: An Application Framework for Ontology-based Software Systems." In: *Semantic Web–Interoperability, Usability, Applicability* (2010).

[177] Elaine J. Weyuker. "Evaluating Software Complexity Measures." In: *IEEE Trans. Software Eng.* 14.9 (1988), pp. 1357–1365.

[178] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. "The Internet of Things—A Survey of Topics and Trends." In: *Information Systems Frontiers* 17.2 (2015), pp. 261–274.

[179] Leland Wilkinson and Michael Friendly. "The History of the Cluster Heat Map." In: *The American Statistician* 63.2 (2009), pp. 179–184.

[180] Scott Norman Woodfield. "Enhanced Effort Estimation by Extending Basic Programming Models to Include Modularity Factors." PhD thesis. West Lafayette, IN, USA, 1980.

[181] Haining Yao, Anthony Mark Orme, and Letha Etzkorn. "Cohesion Metrics for Ontology Design and Application." In: *Journal of Computer Science* 1.1 (2005), pp. 107–113.

[182] Stephen S. Yau and James S. Collofello. "Some Stability Measures for Software Maintenance." In: *IEEE Trans. Software Eng.* 6.6 (1980), pp. 545–552.

[183] Deze Zeng, Song Guo, and Zixue Cheng. "The Web of Things: A Survey." In: *Journal of Communications* 6.6 (2011), pp. 424–438.

[184] Hongyu Zhang, Yuan-Fang Li, and Hee Beng Kuan Tan. "Measuring Design Complexity of Semantic Web Ontologies." In: *Journal of Systems and Software* 83.5 (2010), pp. 803–814.