UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik
Fachgruppe Didaktik der Informatik

ANALYSIS AND INTEGRATION OF WEB 2.0 DATA SOURCES INTO A SYSTEM FOR ANALYSIS AND STORAGE OF ARTEFACT-ACTOR-NETWORKS

Bachelorarbeit

vorgelegt von

Adrian Wilke

Paderborn, 27. Juli 2010

Betreuer_____
Dipl.-Inform. Wolfgang Reinhardt

Gutachter_____
Prof. Dr. Johannes Magenheim
Prof. Dr. Marco Platzner

NACHTRÄGLICHE ÄNDERUNGEN ZUR VERÖFFENTLICHUNG

Entfernung persönlicher Angaben, Hinweis Copyright, Layout Titelblatt, Kürzung Danksagung, keine inhaltlichen Änderungen.

URN: urn:nbn:de:hbz:466:2-28332
DOI: 10.17619/UNIPB/1-87

Adrian Wilke: *Analysis and integration of Web 2.0 data sources into a system for analysis and storage of Artefact-Actor-Networks*

DANKSAGUNG

Bei allen, die mich während der Anfertigung meiner Bachelorarbeit unterstützt haben, möchte ich mich herzlich bedanken. Herrn Prof. Dr. Johannes Magenheim danke ich für die Möglichkeit, die Arbeit in der Fachgruppe Didaktik der Informatik anfertigen zu können. Bei meinem Betreuer Dipl.-Inform. Wolfgang Reinhardt bedanke ich mich für den Themenvorschlag und seine Betreuung. Vielen Dank an Tobias Varlemann und Matthias Moi, die mir gerade anfänglich sehr geholfen haben. Meiner Familie möchte ich für ihre Unterstützung aller Art danken. Vielen Dank auch an Selwyn Nurse, für seine spontane Bereitschaft, einige Texte nach sprachlichen Fehlern durchzusehen.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AA	Artefact-Actor-Relation
AAN	Artefact-Actor-Networks
ACT^2	Actor-Actor-Relation
API	Application Programming Interface
ART^2	Artefact-Artefact-Relation
DCMI	Dublin Core Metadata Initiative
FOAF	Friend of a Friend
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
OSGi	Open Services Gateway initiative
OWL	Web Ontology Language
PDF	Portable Document Format
PLE	Personal Learning Environment
RDF	Resource Description Framework
REST	Representational State Transfer
RDFS	RDF Schema
RSS	Really Simple Syndication
SIOC	Semantically-Interlinked Online Communities
SPARQL	SPARQL Protocol and RDF Query Language
TEL	Technology Enhanced Learning
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language

1

INTRODUCTION

The evolution of the World Wide Web (WWW) is based on new technologies and practices. Rising possibilities for publishing and sharing data increased processes of collaboration. Published artifacts, such as documents, are described via comments and social tagging. Artifacts can also contain references to other artifacts. Networks of artifacts are generated in this manner.

Furthermore, the cooperating persons are in contact. Social networks are formed through the communication and interaction of the participating persons. Social networks exist within organizations and personal surroundings. They can be widened by a consolidation of different sub-networks. Moreover, social networks are spanned by communities formed on Web 2.0 platforms.

Both of these network types can be combined. The concept of Artefact-Actor-Networks (AAN) is based on the consolidation of artefact networks and social networks. It provides additional opportunities for retrieving information. A platform for new requests is generated based on the data from several source networks. For instance, all artifacts of a persons personal surrounding, extracted from different sources, but related to a specific topic can be requested.

Appropriate data must exist and is a precondition before information can be received for further processing. This data can be extracted from different sources. Appropriate sources are open and available for free, such as services provided by the Web 2.0. Data of the microblogging service Twitter [Twi10], and data based on MediaWikis [Wik10] have been integrated into the AAN system. The integration of the social bookmarking service Delicious [Yah10a], and the document-sharing services SlideShare [Sli10b] and Scribd [Scr10b] are treated in this thesis.

1.1 OBJECTIVE AND TASKS

Objective of this thesis is the extension of the existing AAN system with additional data sources. The basis and data sources of this extension is formed by the social networks Delicious, SlideShare, and Scribd. During the embedding of the networks, additional components for access and storage have to be developed. Furthermore, corresponding ontologies have to be defined.

The development and implementation of the system components *Crawler* and *Parser* of the AAN framework is a part of the thesis. If necessary, the same applies to an additional *CrawlerManager*. Therefore the respective OSGi bundles and services have to be created. Data of the networks should be stored as RDF data and, if necessary, as full text.

1.2 STRUCTURE OF THE THESIS

The structure of this thesis depends on the gradual development of the components. In Chapter 1 the work is introduced. An initial overview of the topic is given, the objective of the thesis is clarified, and the structure of the document is explained.

The theoretical framework of this work is described in Chapter 2. The classification of the thesis is done in the context of the field of Science 2.0. It contains the adaption of Web 2.0 opportunities for scientific requirements.

The problem description in Chapter 3 comprises two points. Firstly, the concept of AAN is explained. This mainly comprehends the consolidation of artefact networks and social networks, the introduction of different semantic relation types, and a description of the ontology. Secondly, the social networks Delicious, SlideShare, and Scribd are presented.

Foundations, which are necessary for comprehension of the technical part of this thesis, are part of Chapter 4. At first, the OSGi Service Platform is introduced. It forms the basis of the AAN framework, which is described afterwards. The next section introduces formats, which are used to describe and store the data of AAN. Finally, further used technologies are described.

The most extensive section of the thesis is the solution approach in Chapter 5. Firstly, the data sources are analyzed. This is done by determining relevant data for AAN and the choice of interfaces. Secondly, a solution design is presented. It contains an integration of data into the ontology and a draft for the software components.

The implementation of the developed components is described in Chapter 6. Details of the implementation are described by the workflow of *CrawlerManager* and *Parser* components. At least, a practical application of the implemented components is presented.

Chapter 7 provides a conclusion of the work. Finally, an outlook is given by possibilities of future works.

2

THEORETICAL FRAMEWORK

The young field of Science 2.0 is closely related with several aspects of this work. Further, the field is an origin of the fundamental concepts, on which this thesis is based.

2.1 SCIENCE 2.0

2.1.1 Web 2.0

The innovations of the so-called Web 2.0 have been implemented in the world of science. Main elements of the evolution of the web are new technologies and the resulting options for interaction. Several applications have derived from the development of these technologies: News are published by the usage of blogs, a plethora of users describe artifacts of every stripe by tagging, websites are shared via social bookmarking, documents are shared, and they are edited collaboratively in the form of wikis. The use of these applications results in a altered behavior, caused by the options of new activities. Users can cooperate as they work together. Users can also comment contents and communicate with each other, collaborate and share information. The web can be used worldwide and applications can also be used for organization. This wealth of options supports creativity and productivity. It has been adapted in fields such as journalism, industry, and politics. But how is it adapted in science?

2.1.2 Introduction of Science 2.0

Waldrop [Walo8]) explained the current situation in the context of science: "Science 2.0 generally refers to new practices of scientists who post raw experimental results, nascent theories, claims of discovery and draft papers on the Web for others to see and comment on." In his article, he gives examples such as OpenWetWare[1], a wiki of graduate students working in the laboratories of the Massachusetts Institute of Technology. It is used by students to make notes regarding contents of their studies and functions as a reference in addition to protocols. Another example is the social bookmarking service Connotea[2], which is used for research references. Such references can be scientific papers. Nowadays, free and public online repositories for papers also appear.

Another definition of Science 2.0 was suggested by Ullmann et al. [UWS+10]. After incorporating several viewpoints of Science 2.0., they described it as "the application of new practices that focus on opening up the research process to broaden participation and collaboration with the help of new technologies that are able to foster continuous engagement and further development."

2.1.3 Awareness in Technology Enhanced Learning

A considerable amount of data is published every day owing to the application of Web 2.0 technologies. A part of this data, for instance resources created by experts, could be used for further education in numerous fields. In some cases, the data is already contained in a Personal Learning Environment (PLE). For example, if data was published on websites such as Wikipedia[3], which is already a part of a persons learning resources. It is conceivable, that a lot of additional experts are publishing resources, which are related to a current resource of interest. Accessing this data could be a problem as the user of this data has no information about the expert or the platform where this resource was published.

1 OpenWetWare, http://openwetware.org/
2 Connotea, http://www.connotea.org/
3 Wikipedia, http://wikipedia.org/

This is a general challenge in Technology Enhanced Learning (TEL). A similar scenario is the creation of content of researchers, which takes place beyond the of official scientific publications. Persons having interest in the contents but are unaware of its existence are disadvantaged. This example was described by Ullmann et al. [UWS+10]. They explain the importance of awareness: "However, in many community and group work situations the awareness of others is essential for effective and efficient work." Approaches to handle such challenges are one aspect of the field of Science 2.0

2.1.4 Investigation of collaboration

Like the use of Web 2.0 technologies, applications, and opportunities, the processes of the collaborative working itself can be reflected. Shneiderman [Shn08] supports practical approaches to examine the problem area of Science 2.0: "Science 2.0 challenges cannot be studied adequately in laboratory conditions because controlled experiments do not capture the rich context of Web 2.0 collaboration [...] Moreover, in Science 2.0 the mix of people and technology means that data must be collected in real settings". In relation to this, a set of preconditions for a study is needed. A basis of public data is necessary to examine processes within the field of Science 2.0 and Web 2.0 in general. Furthermore, there is a demand for software, which can handle data processing. Studies in this area should cover a broad field of given Science 2.0 opportunities, such as the use of special webservices. In an effort to use special web-services effectively, resources have to be analyzed, and data giving explanation of collaboration, has to be extracted. The collaboration data should be extracted from several channels of communication. Communication channels are usually used by numerous participants. Hence, knowledge of the relationship between these participants is desirable. An overview of states at different points in time would be desirable when examining the development of collaboration.

2.2 SUMMARY: NEW PRACTICES OF COLLABORATION

New technologies of the Web 2.0 provide new opportunities of collaboration and sharing information. These innovations are adopted by the field of science, whereby practices for research and learning come into being. For instance, some Web 2.0 services provide options for communication, which can be used to increase the awareness of others. Furthermore, processes of collaboration can be observed for research. This has to be done in real settings and requires special software for analysis.

3

PROBLEM DESCRIPTION

This chapter provides a description of the problem area. Firstly, the concept of Artefact-Actor-Networks (AAN) is introduced by a gradual construction of different types of networks. Secondly, an overview of the data sources is provided. Furthermore, aspects of the sources necessary for the integration into the AAN system are explained.

3.1 ARTEFACT-ACTOR-NETWORKS

This thesis is based on the special network type of Artefact-Actor-Networks (AAN), which was introduced by Reinhardt, Moi, and Varlemann in [RMV09].

AAN are the result derived as artefact networks and actor networks have been amalgamated. This consolidation is described in Section 3.1.1.

In AAN, distinction between the different classes is specified. Several types of connections can be determined through relations between the specified classes. Section 3.1.2 deals with those semantic relations, which are used later in the ontology.

Finally, an ontology, which has been defined in [RMV09], is introduced. This scheme of classes, relations, and properties is shown in Section 3.1.3.

3.1.1 Consolidation of network types

This section is structured as follows: Firstly, some terms are determined. Secondly, the consolidation of different artefact net-

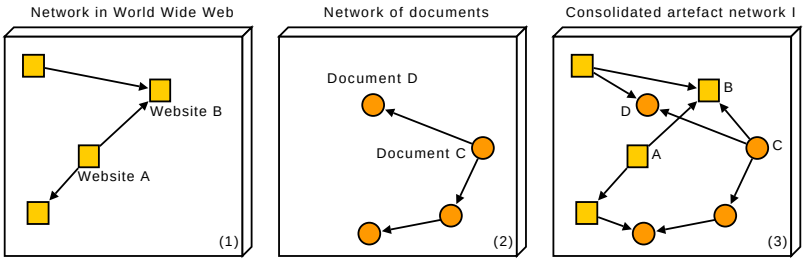

Figure 1: Consolidation of artefact networks I

works is described. In the following, actor networks are described in the same manner. The last step is the consolidation of both network types.

Terminology

In many fields, especially in the computer science, relationships are visualized as *graphs*. In the following, the composition of the different network types are explained by the usage of graphs consisting of *nodes* and *edges*. *Nodes*, used in the examples, represent *objects* as *instances* of the respective *classes* and *edges* are representing *relations* between *classes*.

A *relation* between two objects has an originating point and a terminating point. The originating point is called the *domain* and the terminating point the *range*.

In the AAN context, several *classes* consist in special networks. In contrast to [RMV09], these networks are called *spaces* instead of layers. This naming has been chosen, because the spaces are not built upon each other. A space contains all classes, relations, and data, which are contained in a network. For example, a website is an instance of the class *OnlineArtefact*, which describes the space of artifacts, which can be accessed on the WWW.

In the AAN ontology (Section 3.1.3), *actors* are seen as accounts of *persons* in several networks. It must be mentioned, that overlapping occurs in this section of the terminology.

The point of view of social networks is described in the section actor networks (Section 3.1.1), to express the composition of this network.

Consequently, *actors* are seen similar to *persons*.

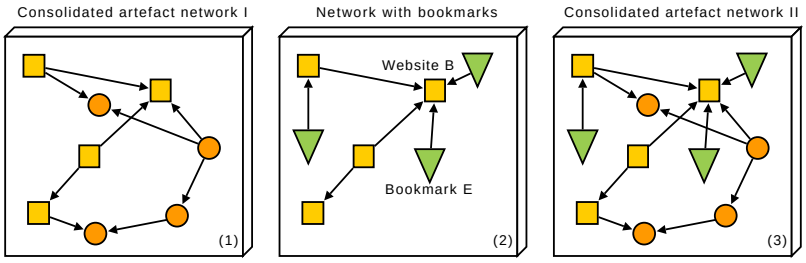

Figure 2: Consolidation of artefact networks II

Artefact networks

A simple artefact network can be imagined as follows: Different sites on the WWW are connected by hyperlinks. For instance, as shown in Figure 1-1, a *website A* is linked to another *website B*. Related sites form an artefact sub-network in the WWW due to a linkage between the websites.

Such networks exist in different spaces. One other example are scientific documents. These refer to each other via the reference part, where papers can be exactly determined by the given meta-data. A *document C*, which refers a *document D* is shown in Figure 1-2.

It is imaginable, that a *document C* contains an URL of a *website B*. A website also could contain reference data to a document. Through these combinations, the two given spaces are consolidated (see 1-3). The involved artefact networks form subspaces of the resulting network.

Webservices such as SlideShare [Sli10b] or Scribd [Scr10b] are parts of such a consolidation. They form interfaces between the space of documents and the space of the WWW. For instance, a document can be uploaded and published into SlideShare. A relation between the respective part of the website, is created during the publishing process, and the uploaded document is available. Consequently, the Uniform Resource Locator (URL) of the website can be used to reference the document.

Bookmarks form a special class in the context of networks. Domain and range of bookmarks are naturally different. Bookmarks link to web documents, the range is given by *OnlineArtefacts*, which is the space of artifacts in the web. By social bookmarking services such as Delicious [Yah10a], bookmarks can be

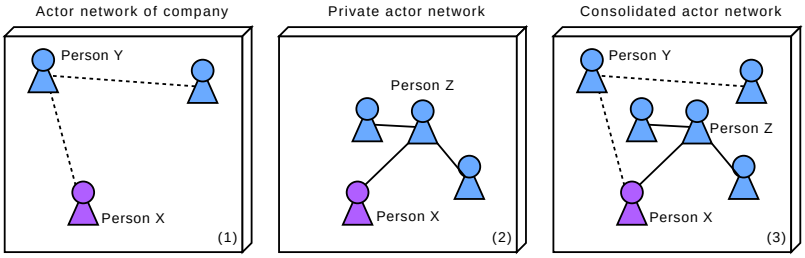

Figure 3: Consolidation of actor networks

stored and published online. Consequently, the web document has a domain designated by the respective bookmarking service.

Figure 2 shows the combination (2-3) of the consolidated artefact network (2-1) and a consolidated bookmarking network (2-2).

Actor networks

Actor networks, or social networks, are a result of persons interacting with each other. Mitchell [Mit69] defined social networks as follows:

"[...] a specific set of linkages among a defined set of persons with the additional property that the characteristics of these linkages as a whole may be used to interpret the social behavior of the persons involved"

In Figure 3-1, known colleagues of *person Y* in a company are visualized. Figure 3-2 shows the space of the private network of a *person Z*. Both persons, *Y* and *Z*, do not know each other personally. By interaction with *person X* a consolidation of both spaces takes place. As shown in Figure 3-3, *person Y* and *person Z* are connected by involving *person X* as an communication interface.

By interpretation of the figure is assumed, that a network is a personal environment. Such a network can also be a social network like Facebook[1] or studiVZ[2]

Consolidation to AAN

Finally, artefact networks and actor networks can be combined to a consolidated AAN. In such a network, there are numerous pos-

1 Facebook,http://www.facebook.com/
2 studiVZ,http://www.studivz.de/

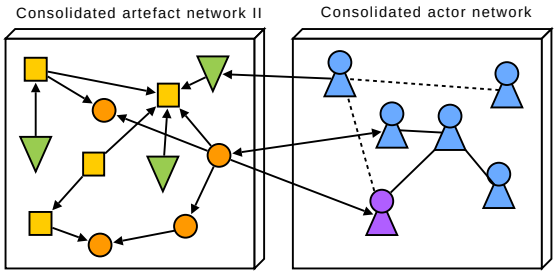

Figure 4: Consolidation of artefact- and actor networks

sibilities for information retrieval. It is logical that on account of the number of consolidations, the created network will consists of multiple branches. A consolidated AAN is shown in Figure 4 on page .

The combination of the different network types create new indirect relations. For instance, a connection between two non-connected actors could result, if the first actor has set a bookmark to a document of the other actor.

Further, a new class of relations between the different network types is created by the consolidation. Similar to the classes of the networks, relations can also be classified. Relations with a special meaning are called semantic relations. The next section deals with the different types of such semantic relations.

3.1.2 Semantic relations

If relations represent a special meaning, they are called semantic relations. Semantic relations are used in ontologies to describe relationships between classes. The structure of AAN, described in Part 3.1.1, exhibits three basic types of semantic relations: Artefact-Artefact-Relations, Actor-Actor-Relations, and Artefact-Actor-Relations.

Artefact-Artefact-Relations

An Artefact-Artefact-Relation (ART^2) is a relation with the class *Artefact* at domain and range. This type of relation can be found in artefact networks and AAN. The meaning of semantic relations of this type is added by inheritance.

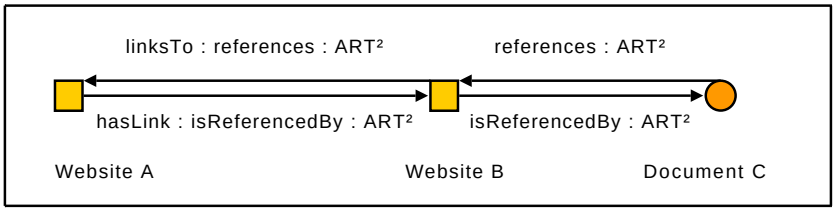

Figure 5: Artefact-Artefact-Relations

Figure 5 shows three objects related by four different semantic relations. In this instance two properties can be found:

- Relations can have inverses. The semantic relation *linksTo*, which is spanned from *website B* to *website A*, has the inverse relation *hasLink*. For the other relations, inverses can also be determined.
- Relations can be defined in an hierarchical order. The relation *linksTo* is a sub-class of the relation *references*, which is a sub-class of *ART²*.

If an ART² relation is spanned between two artifacts, it can be determined, which type of relationship between the artifacts is given. In the example, *document C* references to the *website B*. As the relation *linksTo* from *website B* to *website A* is more special, the given information is more exact.

Basic default types for ART² relations like *source* or *relation* are published by the Dublin Core conventions [Dub10]. More specific and useful definitions, like *links_to* with the subclass *references*, are provided by the SIOC project [SIO10]. These relations can be used for social bookmarking. The three networks, still to be analyzed, consist of simple relations between artifacts, standard vocabularies should provide sufficient relations types to describe the relationships.

Actor-Actor-Relations

Types of an Actor-Actor-Relation (ACT²) are part of AAN, as well as artefact networks. At the domain and range a class of the type *Actor* is defined.

In figure 6 semantic relations *hasInBuddyList* and *relates* are shown. The relation *relates* describes a simple directed relationship between two actors. A more specific statement is given by the

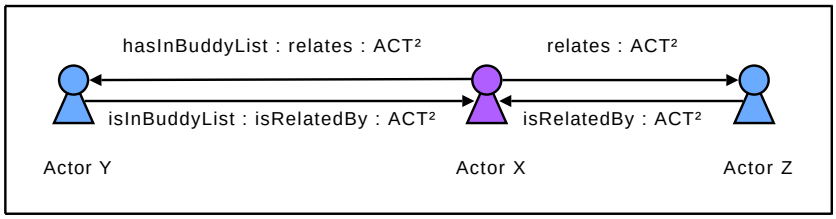

Figure 6: Actor-Actor-Relations

relation *hasInBuddyList*. It gives evidence about the type of the relationship, *actor X* has a list of buddies, where *actor Y* is listed.

The Friend of a Friend (FOAF) project [FOA10] provides the basic ACT² relation *knows*. This is a very general statement and can be used as a base to extend the vocabulary. An extensive approach is provided by RELATIONSHIP [Dav10]. It provides very special ART² relations like *influencedBy* or *closeFriendOf*.

The networks Delicious, SlideShare and Scribd mainly provide relationships between artifacts. Offering the possibility to add other network users to a personal list of known users, semantic relations like *knows* or *relates* could be added. Which relations will be created to describe the relationships is determined in Chapter 5.

Artefact-Actor-Relations

The third type of semantic relations in AAN describe relationships between artifacts and actors. An Artefact-Actor-Relation (AA) is part of a set, which can be further subdivided into two types:

- Actor-to-Artefact relations with domain *Actor* and range *Artefact*, and
- Artefact-to-Actor relations with domain *Artefact* and range *Actor*.

This type of relation is a main part of an consolidated AAN. Figure 7 shows two semantic relations, which are the inverse of each other. For example, in this construct, for example, all bookmarks of an actor or recent bookmarks of an website by several actors can be requested.

The vocabularies mentioned above provide numerous possibilities to express relationships. For the integration of social book-

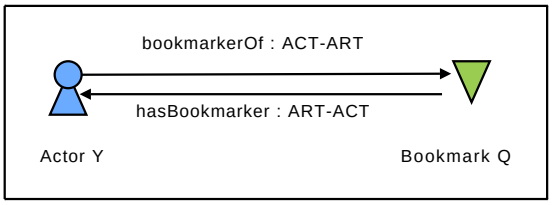

Figure 7: Artefact-Actor-Relations

marks, a semantic relation *isBookmarkerOf* could describe the fact, that an actor has created a bookmark, sufficiently. In the case of publishing documents on SlideShare or Scribd, it has to be determined, if an actor is the creator or publisher of an artefact.

The consolidation of the network types and the resulting types of semantic relations have to be described technically. This has been done by the AAN ontology.

3.1.3 Ontology

In the context of the Semantic Web, ontologies need to specify descriptions for the following concepts [W3C10b]:

- *Classes* (general things) in the many domains of interest
- The *relationships* that can exist among things
- The *properties* (or attributes) those things may have

In this thesis it is differenced between the terms *classes*, *semantic relations* and *data properties*. *Classes* are given by different network spaces, as described in Section 3.1.1. The main *semantic relations* have been introduced in Section 3.1.2. *Properties* consist of metadata, describing the respective instances of classes.

Technically, the AAN ontology is based on Web Ontology Language, which will be introduced in Section 4.3.3.

Figure 8 visualizes a prime version of the AAN ontology, based on [RMV09] and [Var10]. The overview is subdivided into five blocks: *AANBase*, *Web*, *Blog*, *Microblog*, and *Personmanagement*. The main part is the block *AANBase*, where the basic classes of *Actor*, *Artefact* and *Keyword* are defined. These are related via semantic relations *knows* (ACT^2), *hasArtefact* (ART^2), and *hasKey-*

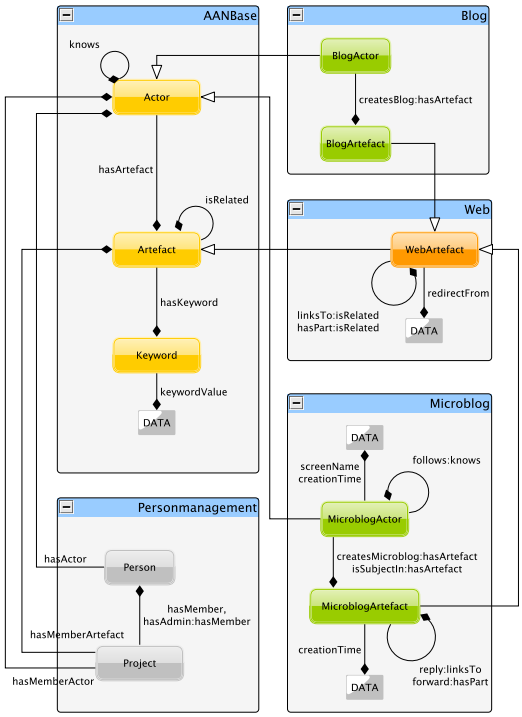

Figure 8: Prime version of the AAN ontology

word (AA). This set of fundamental classes and relations can be used for extension of the AAN network.

In the visualized part, two types of inheritance are shown. Firstly, classes are defined in a hierarchy. For instance, the class *WebArtefact* a specialization of the class *Artefact*. Classes of special network spaces are united in blocks. An example are the classes *MicroblogActor* and *MicroblogArtefact*, which belong to the space of *Microblogs*. The second type of inheritance is the specialization of semantic relations. The relation *isRelated* is a generalized form of *linksTo*.

3.1.4 Practical application

Fields of application of AAN arise by the use and the extension of the given structures and data.

Moi [Moi10] has extended the AAN system by the possibility of integrating MediaWiki sources [Wik10]. In his diploma thesis, he integrated a computation and visualization of semantical sim-

ilarity for MediaWiki artifacts, which is based on artifacts stored within the AAN system.

Information Retrieval offers the greatest ambiance for this application. Objects of the classes *Artefact*, *Actor*, and *Keyword* can provide valuable information by simply applying special queries. If one requests artifacts by insertion of keywords of interest, then related artifacts of different networks and their actors will be offered. Actors may be contacts with expertise. Hence, AAN are interesting for the area of expert finding. Another example is to request related data to an artefact. The given tags for the artefact provide relations to other artifacts. The data of these artifacts should receive relevant information to the topic of the initial artefact. Consequently, the use of AAN is a tool for investigation and can be used as a part of a PLE. Several types of requests and returned data are conceivable

Valuable data can only be accessed if it stored within the AAN system. This is done by accessing data of several sources. Three source networks are introduced in the next section.

3.2 EXTENSION BY DATA SOURCES

With the objective of analyzing stored data in the AAN system and to benefit from the system itself, data firstly has to be stored. At this time, data from the sources of websites, twitter entries, and MediaWikis can be accessed. Additional data of general interest and for the field of research is consisted in presentations, slides and similar documents. The social media websites SlideShare [Sli10b] and Scribd [Scr10b] provide such data, which include documents related to topical science subjects. Another relevant area are bookmarks. Bookmarks can be described with keywords and artifacts of every stripe can be referenced. The social bookmarking service Delicious [Yah10a] provides access to bookmarks of numerous fields of interest.

3.2.1 Delicious

The web-based social bookmarking service Delicious started in 2003. Five years later, 5.3 million users were registered and created 180 million unique URLs. [Yah10c]

The bookmarking service is used to store and share bookmarks online. One main advantage is the access to bookmarks worldwide, once a connection to the website is established. A backup of bookmarks is generated automatically when storage is done and a social bookmarking service is used. Registered users can create bookmarks and relate it with notices and keywords in the form of tags. Public bookmarks can be browsed on the website. This can be done for the bookmarks of a special user or a tag. It can also be combined by a user and a set of tags.

Delicious can be used for sharing bookmarks in projects. Therefore, users can add other users to their personal networks. Another option is to define a project-related tag. By the choice of a tag, a unique string is favorable. Tags such as "web20" are used frequently. Hence project bookmarks, exclusively listed by such a tag, are difficult to retrieve.

As Delicious is one of the first and most popular social bookmarking services, it has been used for research. The free availability of users, bookmarks, and tags provides options for evaluating concepts and algorithms in reality. For instance, Mika [Mik07] used it for a case study of an abstract model of social-semantic networks.

Delicious is useful for the application of AAN, because it provides a combination of both artefact networks and actor networks. Further, bookmarks are naturally related with artifacts of different network spaces. Referenced websites are a part of several networks in the WWW.

3.2.2 SlideShare

The slide hosting service SlideShare was launched in October 2006. In September 2007, 3 million unique visitors used the site. Today, there are 25 million visitors per month and 70 million monthly page views. [Sli07, Sli10a]

Users of the service can upload presentations (files of presentation programs such as Microsoft PowerPoint), general documents (files of document preparation systems such as Microsoft Word and other office application formats), and videos (video files of different formats and codecs). For each document, additional metadata can be entered: A title, a description, tags, a category, the language, details of privacy, and it can also be selected, if other users are allowed to download the document.

SlideShare is used for publishing data from different fields. As it is used for sharing presentations and documents in addition to scientific papers, it provides valuable data. Some examples of popular users are the founder of O'Reilly Media, Tim O'Reilly[3], the environmental organization WWF Germany[4], and Erik Duval[5], professor of the University of Leuven.

As uploaded documents could contain valuable data for several fields of interest, the integration of SlideShare into the AAN network provides usable data for Information Retrieval. Like in Delicious, additional actor networks can be extracted.

3.2.3 Scribd

Scribd, a website for sharing documents, was launched in March 2007. It has at least ten millions published documents and the same amount of readers monthly. [Scr10a]

On Scribd, users have the possibility to upload, share, and sell documents in several formats. Some of these formats are Portable Document Format (PDF), office documents (OpenDocument and Microsoft Word, PowerPoint, Exel), plain text, or the Tagged Image File Format. By uploading, details of title, categories, tags and descriptions can be set. Additional fields to describe the content are offered afterwards: Choices offered are, if a file is declared public or private, the language of the document can be set and comments can be allowed. Further, details of the view mode, download formats, printing, copying text, and the license can be set. Scribd users have the possibility to subscribe to other users. This action generates a new network of subscribers.

3 Tim O'Reilly at SlideShare, http://www.slideshare.net/timoreilly
4 World Wide Fund For Nature, Germany, at SlideShare, http://www.slideshare.net/wwfdeutschland
5 Erik Duval at SlideShare, http://www.slideshare.net/erik.duval

Scribd is used by several groups of people, such as authors, musicians and publishers. Publishers use the platform offering an insight in publications and to promote the probability of selling them. Some examples of publishers are the Springer Publishing Company[6], O'Reilly Media[7], and the Harvard University Press[8].

In the context of AAN, the Scribd network can be seen similar to SlideShare. A distinction can be made in the types of artifacts. SlideShare provides certain documents such as presentations. Scribd is more specialized in publishing artifacts, such as different sections or articles of books.

3.3 SUMMARY: NUMEROUS POSSIBILITIES TO APPLY THE CONCEPTS OF AAN

The concepts of AAN provide many variations of extension. This is caused by the consolidation of networks composed of artifacts and actors. Semantics and a hierarchical structure offers the possibility of diverse requests is given. Consequently, various types of information can be derived. For a practical application of AAN, a precondition is the existence of data. The networks Delicious, SlideShare, and Scribd provide useful data of different fields. This data could be used in AAN for exploring information.

6 The Springer Publishing Company at Scribd, http://www.scribd.com/Springer%20Publishing%20Company
7 O'Reilly Media at Scribd, http://www.scribd.com/O%27Reilly
8 The Harvard University Press at Scribd, http://www.scribd.com/harvard_press

4

TECHNICAL FOUNDATIONS

This chapter provides foundations, necessary for comprehension of the technical terms and relationships used in this thesis. The first section introduces the OSGi Service Platform, on which the implementation of the AAN framework (Section 4.2) is based. Section 4.3 serves as an insight to the resource formats, with which the data of AAN is described. The chapter is concluded by Section 4.4, which subsumes further essential technologies.

4.1 OSGI SERVICE PLATFORM

The foundation of the conception and implementation of the AAN framework is the OSGi Service Platform, which is specified by the OSGi Alliance, formerly known as Open Services Gateway initiative (OSGi) [OSG10]. The specification is applied in different fields. For instance it is implemented as Equinox, a framework developed by the Eclipse Foundation[1]. This is the implementation used for the AAN framework. It is based on Java [Ora10], thereby affording portability.

[WHKL08] gives an appropriate summary about the platform: "The OSGi Service Platform is Java-based software platform, which enables a dynamic integration and remote management of software components (bundles) and services. Without a need to stop or restart the platform as a whole, bundles and services can be installed, started, stopped, and deinstalled at runtime."

The core component of the platform is the OSGi Framework. It consists of four main layers: *Modules*, *Life-Circle*, *Service*, and *Se-*

1 Eclipse Foundation, Inc., http://www.eclipse.org/

curity. The *Security Layer* comprises all other layers. It defines execute permissions of components and similar security aspects.

Components of the OSGi Framework are specified by the *Modules Layer*. This is done via modulation and dependence management, which operates on versioning. Individual components are organized in bundles. Such bundles consist of resources and classes, which are not visible for other bundles by default. To define visible parts, bundles can be exported statically or they are provided dynamically as Java objects of service interfaces.

Bundles can be in different states, which are handled by the *Life-Circle Layer*. It defines the states *uninstalled*, *installed*, *resolved*, *starting*, *active*, and *stopping*. These states make the dynamic integration of the bundles possible.

One of the most important aspects of dynamism is provided by the *Service Layer*. Without interruptions of available services, bundles can be replaced at runtime. This possibility is given by decoupling services via interfaces. By implementation of a respective service interface, a service can be provided by different bundles, or bundle versions. Service interfaces have names, which are used to register the services at the *Service-Registry*. Further, offered services can be requested and used by bundles without a need for information of implementation details.

The OSGi Service Platform provides several standard services. For instance, the transmission of system information or debug messages can be handled by the defined *Log Service*. Another example is the *Event Service*, by which events can be propagated between bundles. This two services are used by the AAN framework, introduced in the next Chapter.

4.2 AAN FRAMEWORK

The practical realization of the AAN concepts were done by Varlemann, who designed and implemented the components based on the OSGi Service Platform [Var10]. Moi has extended the architecture by including components for similarity analysis of texts [Moi10].

The architecture of the AAN framework is shown in Figure 9. It consists of components, realized as OSGi bundles, which are

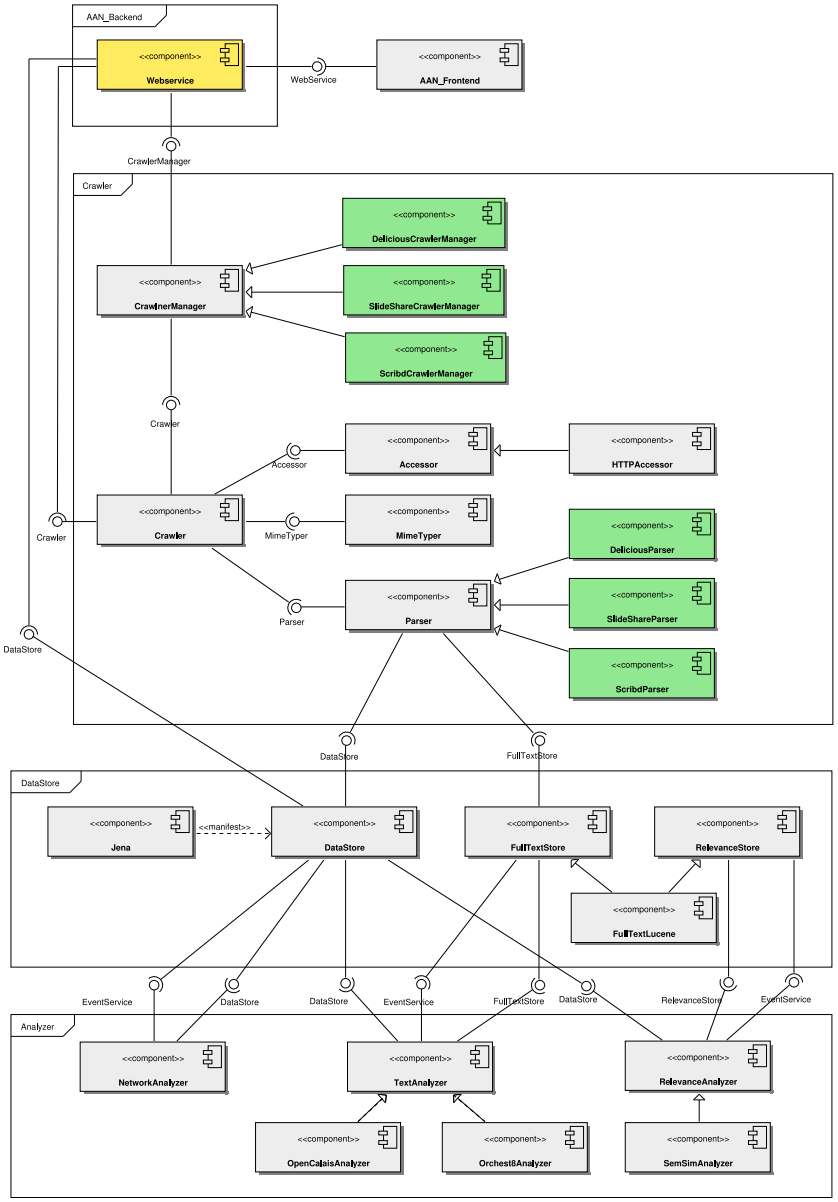

Figure 9: Architecture of the AAN framework

combined into the main blocks *Crawler*, *DataStore*, *Analyzer* and *AAN_Backend*. Colored components are parts of the integration of this work and described in Chapter 5.2. The component *Webservice* is also part of the current implementation. The functions of blocks and contained components are presented in the following.

4.2.1 Backend

The *AAN_Backend-Block* only consists of the component *Webservice*. The component functions as an interface between the AAN backend and users or applications at the frontend. It provides access to stored data and can be used to add new tasks, which mainly consist of an Uniform Resource Identifier (URI), to the *Crawler* or *CrawlerManager*.

4.2.2 Crawler

In the *Crawler-Block*, tasks are handled by the *Crawler* component. This is done by a working chain, consisting of *Accessor*, *MimeTyper*, and *Parser*. A sequence diagram of the working chain is displayed in Figure 18 on page 60. The *Crawler* creates a *CrawlerDataObject*, which is used by the components of the working chain, to store the state and information of the crawling process.

The *Accessor* component allows access to data sources by protocols like Hypertext Transfer Protocol (HTTP) or File Transfer Protocol (FTP). Loaded resources are stored locally, and references to created files are returned for re-working.

In the next step, the *MimeTyper* component detects one or several MIME types (like *text/html* or *text/xml*) of the loaded resource. MIME types and the parser type are used to choose a suitable parser.

There are two types of the *Parser* component: special and general. By the choice of a suitable parser, special parsers are preferred. Special parsers provide functionalities to extract data of a special type of resource. Such a special type could be a user-specific website of the Delicious network, which provides metadata, that can be extracted by a *DelicousParser*. A general parser, which

also could parse the resource is the *HTML Parser*. It mainly would follow hyperlinks without extracting network-specific metadata.

The last component of this block is the *CrawlerManager*. *Crawler-Managers* provide special functionality to crawl resources with individual properties. This could be websites or networks, of which structures require special information to extract metadata. A *CrawlerManager* makes use of *Crawler* services.

4.2.3 Datastore

The *Datastore-Block* is subdivided into the components *DataStore*, *Jena*, *FullTextStore*, *RelevanceStore*, and *FullTextLucene*. Data, for the most part, is stored by *Accessor* and *Parser* components, and provided by *Webservices*.

In the current architecture, the *DataStore* component is combined with the *Jena* component. This combination is used to store and provide Resource Description Framework (RDF) triples, which are extracted via the crawling process. This data forms the model of AAN.

Via the *FullTextStore*, complete texts, such as the source code of websites, are stored. This is done during the crawling process by *Accessor* components.

The component *FullTextLucene* provides storing functionalities to calculate similarities. It is a combination of the *FullTextStore* and the *RelevanceStore*. The *RelevanceStore* is explicitly used to store relevancies, whereas the *FullTextStore* is used for full texts during the computation of relevances.

4.2.4 Analyzer

Finally, the *Analyzer-Block* contains three different components for the analysis of networks, texts, and relevancies. Analyzer components react to events, which are fired by components of the *Datastore-Block*.

Components of the type *NetworkAnalyzer* are calculating with the data of the *DataStore* component. Their computations are based on data consisting of the RDF format and can be used for structural analysis.

TextAnalyzers, such as the planned *OpenCalaisAnalyzer*, based on OpenCalais[2], are using the *FullTextStore* to extract metadata from full text sources. Extracted data can be stored using the *DataStore* component.

The last group of analyzers components are of the type *RelevanceAnalyzer*. Components of this type provide functions to compute similarity between different artifacts.

In this work, the AAN framework has to be extended for an integration of data sources. Accordingly, for the access of data, *CrawlerManagers* and *Parsers* have to be developed. *CrawlerManagers* provide the functionality to crawl the structure of given networks and *Parsers* are used to extract relevant data of resources. The source networks are introduced in the next section.

4.3 RESOURCE FORMATS

This section introduces the formats, which are used to describe and store the data of AAN. It starts with syntactic definitions, afterwards semantics are added. Finally, all resource formats needed for comprehension of the structure of AAN are given.

4.3.1 Resource Description Framework

"The RDF is a language for representing information about resources [...]." [W3C10c] Such a resource is identified uniquely via an URI, which is a string of characters structured in the URI scheme. A popular example for an URI is an URL, which identifies a website.

RDF provides a formal description of information of resources. This description is done via triples. Such a triple comprises three parts in the following order: a *subject*, a *predicate*, and an *object*. These parts again can consist of different values:

- A *subject* is a resource. This is either an URI or a blank node, which describes an anonymous resource.
- A *predicate* also is a resource, blank nodes excluded.

2 OpenCalais, http://www.opencalais.com/

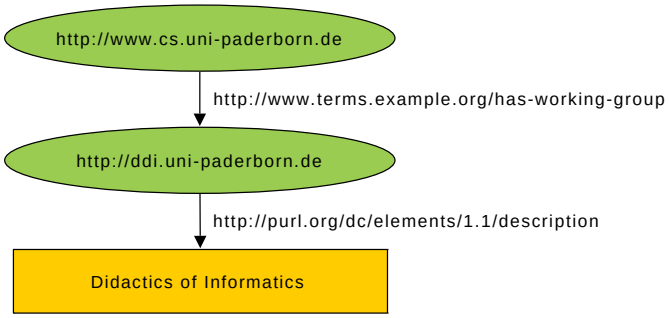

Figure 10: Example of two RDF triples

- An *object* can be a resource (also a blank node), or a literal, which is a constant value.

Figure 10 shows an example of two triples. The subject *http://www.cs.uni-paderborn.de* at the top is related with the object *http://ddi.uni-paderborn.de* by the predicate *[...] has-working-group*. This resource again is subject of another triple with the predicate *[...] description* and a literal *Didactics of Informatics* as object. By such chains of relations, directed RDF graphs come into being. Based on these graphs, the syntax of data of AAN is described.

Generally, concerning the terminology of RDF, it is differenced between *resources*, *properties*, and *property values*. In the context of the example above, a resource *DDI* can have a property *Description* with the value *Didactics of Informatics*.

4.3.2 RDF Schema

The RDF Schema (RDFS) is "RDF's vocabulary description language" [W3C10d], by which ontologies can be defined. Ontologies, in terms of the semantic web, deliver descriptions of the concepts of classes, relationships, and properties. They "define terms used to describe and represent an area of knowledge." [W3C10b]

RDF provides a syntax to describe resources, however a semantical description is not available. For instance, the predicate *[...] description* of the example, visualized in figure 10, has been set without any semantical description.

The application of RDFS constructs delivers possibilities for the description of relationships. This is done by concepts of sev-

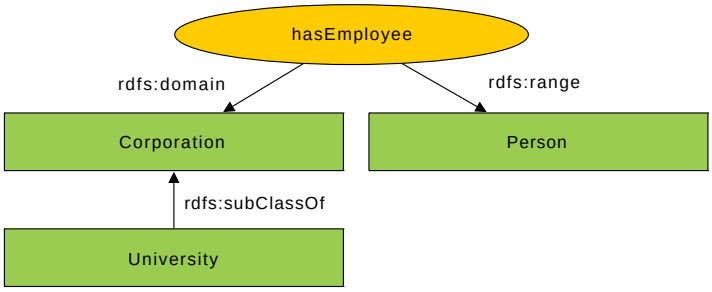

Figure 11: Example of RDFS properties

eral classes (*rdfs:Class*) and properties (*rdf:Property*), which are defined again by RDF. The basis of the vocabulary is formed by a hierarchy of classes, consisting of the *rdfs:Class* concept and several subclasses, e.g. *rdfs:Resource*, *rdfs:Literal*, or *rdf:Property*. Additionally, various instances of the class *rdf:Property* are defined. By those, for example, a relations origin (*rdfs:domain*) and objective (*rdfs:range*) can be determined. Further, hierarchical structures can be formed by the properties *rdfs:subClassOf* and *rdfs:subPropertyOf*.

Three examples of RDFS properties shows figure 11. The figure contains three classes (*Corporation*, *Person*, and *University*) and a property (*hasEmployee*). By RDFS the *domain* and *range* of *hasEmployee* are defined. Further, a hierarchy is given by the property *subClassOf* from *University* to *Corporation*. By this way, an implicit relationship "*University-hasEmployee-Person*" can be exploited for two instances of *University* and *Person*.

4.3.3 Web Ontology Language

The "Web Ontology Language (OWL) is intended to be used when the information contained in documents needs to be processed by applications [...]. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms." [W3C10a]

This quotation resembles the description of RDFS above. OWL in fact includes some classes and properties of RDFS. Indeed, it delivers more expressive constructs, also in support of applications.

There are three sub-languages in OWL: *OWL Lite*, *OWL DL*, and *OWL Full*. The languages differ in the expressiveness. Some features, which are included in all sub-languages, are following.

In comparison with RDFS, OWL provides some additional property characteristics. One example is the property *inverseOf*, which is used by the AAN ontology. Assume, a property *isBookmarkOf* is the *inverseOf* a property *hasBookmark*. Further, a resource *A* is related to a resource *B* by *hasBookmark*. Then *B* is related to *A* by *isBookmarkOf*.

In addition to this property characteristic, statements of transitivity or symmetry can be defined. Moreover, restrictions, e.g. in cardinality, can be done.

The AAN ontology, introduced in Section 3.1.3, is based on the possibilities of OWL. In the next parts, further used technologies are described. Some of them refer to the resource formats described in this section.

4.4 FURTHER USED TECHNOLOGIES

4.4.1 SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) "can be used to express queries across diverse data sources", if the given "data is stored [...] as RDF" [W3C10e].

In the AAN framework (see Chapter 4.2), SPARQL is used by web-services, which form interfaces between the framework and external frontends or users, which directly request queries.

Utilizing SPARQL, one can request queries by giving graph patterns containing conjunctions and disjunctions. By this, usual sets of results or RDF graphs can be returned.

SPARQL is supported by ARQ, a query engine for the Jena framework, introduced in the next section.

4.4.2 Jena

Jena is a "framework for building Semantic Web applications" [Jen10]. It is written in Java [Ora10] and was developed at the HP Labs[3]. Today, it is an open source framework.

Jena supplies RDF support by providing an Application Programming Interface (API), with which RDF graphs can be read and written. RDF graphs can be serialized and persistently stored in different ways, e.g. by rational databases (SDB), or by an specialized engine (TDB) with high performance. This is done by custom indexing and storage. Models, stored or in memory, can be queried via SPARQL and the query engine ARQ.

As Jena additionally supports the OWL vocabulary, it was integrated into the AAN datastore [Var10].

4.4.3 JavaScript Object Notation

"JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. [...] JSON defines a small set of formatting rules for the portable representation of structured data." [IET10]

JSON is used to serialize structured data and an alternative to the Extensible Markup Language (XML). Generally, JSON is smaller than non-compressed XML, because no closing tags are used.

In JSON, the *values array*, *number*, *object*, *string*, *true*, *false*, and *null* are defined. An *array* consists of an opening "[", a closing "]", and *values* divided by commas. An *object* begins with a "{", ends with a "}" and the main contents are given by pairs of *strings* and *values* in the form *string:value*, divided by commas.

JSON is used by Delicious feeds as an alternative to Really Simple Syndication (RSS).

3 Semantic Web Research at the Laboratories of the Hewlett-Packard Development Company, L.P., http://www.hpl.hp.com/semweb/

4.5 SUMMARY: A BASIS OF DYNAMICS AND SEMANTICS

The OSGi Service Platform provides a foundation for dynamic system components. Such components have been developed and form the AAN framework. It was designed to provide extensibility, following an integration of new components is supported.

RDF provides a representation format for information of resources. This is integrated into RDFS, by which ontologies can be defined. The opportunities for describing ontologies are extended by OWL. It provides a basis for defining a hierarchical structure of classes and semantical definitions. This data can be stored by Jena.

5

SOLUTION APPROACH

For a solution approach Delicious, SlideShare, and Scribd are analyzed to determine accessible data. Afterwards, the data is integrated into the ontology.

5.1 ANALYSIS OF RELEVANT DATA SOURCES

The three data sources are analyzed In this section. The analysis of the respective networks is divided into two parts: Firstly, possible data for extraction is collected by practical use of the network services and by analysis of the provided outcome of the websites. Secondly, at least one interface has to be chosen, which has to provide the desired data. The method used to integrate the data, by simply extending the given ontology, is covered by the next chapter.

When analyzing a network preliminarily, the classes, semantic relations, and the data properties of each source network to be extracted are classified. This data has to be relevant for the use of AAN. Relevant data correlates with the classes and semantic relations of the base ontology, introduced in Section 3.1.3. Classes of the ontology can be related with data properties. These data properties provide metadata, which describes the instances of classes. As extracted data is stored within the AAN system, only static data should be chosen for integration. Dynamic data, such as the current number of bookmarks of an actor, or the number of followers, refers to a point of time and could be outdated without notice. Furthermore, data which describes artifacts, is not generated automatically. In some cases, users have options to add data in their personal predilections. Inconsistency results when predetermined fields are used differently.

Summarized, certain criteria applied for data processing:

- Data has to be relevant for use of AAN.
- Only public data can be accessed.
- Values of data properties should be static.
- Data properties should be used consistently.

After determining which data is offered for extraction, the method of accessing must be defined. The public website of the service is an interface which is provided by each network. The HyperText Markup Language (HTML) code of a site could be parsed. This method of access has a disadvantage as the possibility exist that the code format is not well structured. Even if the HTML code is structured and contains describing IDs and names, changes in the structure of the website can result in a non-parsable code for a current parser. Therefore, other options for accessing should be analyzed.

An example of a preferable interface is an API, which is provided by serveral webservices. An API is well-structured and even if new versions of an API are published, old versions are further supported for a period of time, or old versions are integrated to newer ones.

The choice of an appropriate interface is dependent on these criteria:

- The interface must provide public data.
- All relevant data, which has been chosen in the first section of a networks analysis, has to be available.
- Restrictions of an interface should not hinder the process of crawling or parsing.
- A well-defined structure of the data is favorable.

It must be analyzed, which options of accessing the data is the most appropriate alternative, and these options defined for developers. The preferred choice is the use of an API, which provides all relevant and public data without restrictions. If such an interface is not offered, other suitable interfaces can be used. Thus, a combination of different interfaces is thinkable.

The analysis of the Delicious network contains procedures, which are applied in each of the networks. As the way of proceeding is introcuded by the first part, the analysis of Delicious is described more detailed.

5.1.1 Delicious

Data for Extraction

It must first be analyzed which fundamental data exists and has been published by Delicious network in order to extract the relevant data. The method of proceeding at this point, is to examine which data can be set by users and which public data will be published by the Delicious website afterwards.

Delicious is primarily an artefact network. Therefore, the focus here is on bookmarks. The relevant details which are set by users are *URL*, *title*, *notes*, *tags*, and *for*, when a bookmark is set. When viewed from the Artefact-Actor-Networks ontology, the details can be interpreted as follows:

- The field *URL* is construed as an semantic relation of the type *ART*[2] with domain *Artefact* and range *OnlineArtefact*. This means, that a created bookmark points to an artefact reachable by the WWW. These links should be included to the set of extracted data, to give options for accessing external artifacts during the crawling process.
- *Title* and *notes* can be seen as data properties of an *Artefact*. They form additional meta information of a bookmark.

  When a set of received bookmarks are analyzed, it may be discovered that the field *notes* are used inconsistently. Frequently notes of different users are equal, this could a result by the use of applications, which automatically set this field to the first headline or a describing meta-element in the head section of a HTML document. Such data is already explained by the *title* field. Secondly, different languages are used for taking notes. Therefore, values of the field *notes* will not be extracted.
- The field *tags* provide two points for each tag: Firstly, it describes an object of the class *Keyword*. Additionally two semantic relations can be extracted: The *hasKeyword* relation and the *isKeywordOf* relation.
- By filling in the field *for*, the additional option of notifying users of bookmarks is defined. An addressee could be another Delicious user, an email address or a notification by the social network twitter. As this data is not published, it is not relevant for this task.

Beside the creation of bookmarks and tags, Delicious users are able to add other users to their network. This can be done by using the hyperlink "Add a user to Network", if an user is logged in. By this option, actor networks are generated. This forms an additional point, which can be extracted:

- When crawling a *users network*, data which corresponds to the semantic relation *ACT*^{2} is provided.

The second source for potentially usable data of the Delicious network is given by analyzing the result of the Delicious website. In addition to the data properties above, the *author* of a bookmark can be accessed by the web. For each bookmark the *date* of the creation is published additionally. If a visitor looks up a users bookmarks, he can refine the result by limiting the users *bookmarks* to those *described with selectable tags*. This selection can also be done by hyperlinks.

Some more details about the network (e.g. the number of bookmarks for a specific URL) are provided by the Delicious website. This information affects points, which are not relevant for the current use of Artefact-Actor-Networks. When considering the example of the number of bookmarks for a specific URL, the information is uninteresting, because once the crawling process is concluded, all bookmarks of an URL should have been stored. Then the number of bookmarks can then be requested via SPARQL.

Summarized, we gather the following points for further options of extraction.

- *Author* corresponds to the class *Actor* of the base ontology.
- As an *Author* is creating bookmarks, this generates semantic relation between *Actor* and *Artefact*.
- *Date* is a data property of an Delicious *Artefact*.
- *Artifacts* can be selected by a combination of an user and tags. As the base ontology is build up without direct semantic relations between *Actors* and *Keywords*, this point cannot be used for the ontology itself. However, it provides a useful method for extraction.

Finally, here is an overview of data which can be potentially extracted:

Classes:

- *Bookmark* (*Artefact*) with the data properties: *Title* and *Date*
- *Tag* (*Keyword*) with the data property: *Name*
- *Actor* with the data property: *Name*

Semantic relations:

- *ACT*^{2} with domain *Actor* and range *Actor* from actor A to actor B
- *ACT*^{2} with domain *Actor* and range *Actor* from actor B to actor A
- *hasArtefact* of type *AA*, with domain *Actor* and range *Artefact*
- *isArtefactOf* of type *AA*, with domain *Artefact* and range *Actor*
- *ART*^{2} with domain *Artefact* and range *Online Artefact*
- *ART*^{2} with domain *Online Artefact* and range *Artefact*
- *hasKeyword* with domain *Artefact* and range *Tag*
- *isKeywordOf* with domain *Tag* and range *Artefact*

The classes, relations and data properties collected in this section should be offered by the chosen interface. The next section examines possible methods of data access.

Choice of Interface

Delicious offers multiple options for developers to access the available data [Yah10d]. Depending on the desired result, one can choose from different interfaces, e.g. an API, feeds, [Yah10b] or linkrolls. In this section three applicable alternatives to access the Delicious network are examined: The Delicious API, the use of feeds and the possibility to parse HTML code of the Delicious Website.

DELICIOUS API Generally, one of the most applied methods to access data is by the use of an API. This option was also tested for applicability to the AAN system. The offered API-methods are custom-made for the access and use of a users personal data. A user can create, edit and receive personal bookmarks, tags

INTERFACE	PUBLIC DATA	WELL-DEFINED
API	–	✓
Feeds	✓	✓
HTML parser	✓	–

Table 1: Comparison of Delicious Interfaces

and tag bundles. All these actions require an user authentication. The AAN was developed to extract and analyze public data. However, the need for the users authentication by the API presents an hindrance. The alternatives must offer a public interface to access the data. Other than that, the hindrance must be completely eliminated.

DELICIOUS FEEDS A more usable approach to access data is the use of Delicious feeds. Feeds are offered in JSON (see Section 4.4.3) and RSS format, and provide an access to public data. It is possible to get the latest bookmarks, tags and network members of a specified user. Furthermore, requests for bookmarks can be refined by combining a specific username and tags. Generally, recent bookmarks for an URL can also be retrieved. This forms an extensive base for information retrieval.

HTML PARSER The last examined method is the use of a HTML parser. Parsing the Delicious website is a method of accessing the largest amount of data. Most of the data created by users is published on the web, except the data described as private. The HTML code of the network is structured and provides a lot of class and id statements. A disadvantage of this interface is, that the code is not well-defined. As a result an implemented parser would not work, if changes in the structure of the code were done.

COMPARISON AND DECISION Taken together, the use of Delicious feeds is the best option to access the network. Table 1 shows the advantages: Feeds provide the necessary public data and a well-defined structure. The chosen format of the feeds is JSON. The JSON feeds provide almost the same amount of data as feeds in RSS format in a clearly more compressed form. Furthermore one feed address (*urlinfo*) is provided only in this format.

The use of the Delicious JSON feeds is described more detailed in the next section.

Use of Delicious Feeds

Delicious provides 23 feed addresses to receive network data. A complete list of URL patterns and descriptions are shown in Appendix A.2. The list was taken from [Yah10b]. For a better assignment, the feed addresses are numbered.

FEED ADDRESSES WITHOUT USE 10 of the provided 21 feed addresses are not used. The addresses affected are explained in this section.

- The patterns 1, 2, and 4 provide data about bookmarks which are not requested by arguments consisting of an *Actor*, *Tag* or web URL. Therefore they do not provide specific data and are useless for the usage of AAN.
- The feed address 6 is reserved for Delicious site alerts.
- The URL patterns 8, 10, 15, 17, and 19 provide private data. This data only can be accessed, if a key is handed over. As such private keys are unknown hence, private data cannot be accessed.
- Feed address 11 returns static user information. Return values are the number of bookmarks, network members, and network fans of a given user. This data will already be available, if user-related feeds are crawled completely.
- Pattern 14 delivers recent bookmarks from user subscriptions. As these are not associated with a users bookmarks, they are not in use.
- Finally the feeds 16 and 18 provide the latest bookmarks form a users network. As the network will be crawled by the system, this is redundant data.

FEED ADDRESSES IN USE The method of classifying and utilizing the 10 used Delicious feeds are analysed in this section. The URL patterns can be classified by examining the schemes of the return values. Table 2 shows an overview of the used feeds. The feeds are named for a better understanding. Further, the needed input values and the resulting output schemes are listed,

FEED NAMING	INPUT	OUTPUT SCHEME	REF.
bmPopularByTag	Tag	Bookmarks	5
bmByTags	Tag(s)	Bookmarks	3
bmByUrl	URL	Bookmarks	22
bmByUser	User	Bookmarks	7
bmByUserAndTags	User, Tag(s)	Bookmarks	9
networkFans	User	Users	21
networkMembers	User	Users	20
tagsByUser	User	Tags	12
tagsByUserAndTags	User, Tag(s)	Tags	13
urlInfo	URL	URL	23

Table 2: Schemes of used Delicious feeds

and references to the Delicious feed patterns (see Appendix A.2) are given.

For example, the feed *bmByUserAndTags* needs two arguments: An user name and a set of tags. The returned scheme provides a set of bookmarks. Each of these bookmarks were created by the user which was handed over and was described with each of the given tags.

Four different types of feeds can be examined by classification: *Bookmarks*, *Users*, *Tags*, and *URL*. Each of these schemes provides different sets of values, which refer to classes of the ontology. On the one hand, the scheme types themselves differ, on the other hand, the number of returned values of a scheme can also be different.

The feed schemes can be seen at Appendix A.1. Here is an overview of the scheme properties is listed:

- The scheme *Bookmarks* counts up to 100 bookmark entries. Every entry consists of one *OnlineArtefact*, one *Author*, and a set of *Tags*.
- The scheme *Tags* delivers a set of *Tags*.
- The scheme *Users* delivers a set of *Actors*.
- The scheme *URL* consists of one *OnlineArtefact* and a set of *Tags*.

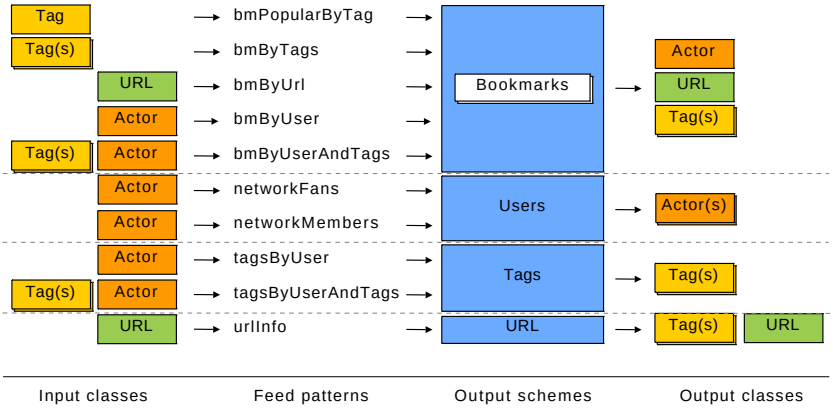

Figure 12: Inputs and outputs of Delicious feed patterns

To visualize the data flow, input classes, feed patterns, output schemes, and output classes are shown schematically in Figure 12. For instance, if two objects of the classes *Actor* and *Tag* are passed to the feed *bmByUserAndTags*, the corresponding return is a set of data of the scheme *Bookmarks*. This set may contain some instances of the class *Bookmark*. Each of the *Bookmark* objects provides data about the creating *Actor*, the *URL* of the referred *OnlineArtefact* and *Tags*, which provide some meta data.

DISADVANTAGES OF FEEDS The use of feeds to access data of the Delicious network poses two difficulties.

The first disadvantage is, that the number of returned feed entries is limited to 100. Additionally, no possibility to start at an older entry (e.g. by an argument "start at entry 101") is given. The reason for this is that the originate in the spreading of news. Therefore the normal case is, that only recent news are interesting and provided. This affects the return of methods of the scheme *Bookmarks*.

The second restriction is, that only one request per second is allowed. As a result, the access by feeds should be limited to one thread. Furthermore every request becomes valuable by this limitation.

5.1.2 SlideShare

Data for Extraction

SlideShare is primarily a platform to share presentations and documents. For each of these resources it has to be determined, which data can be extracted. Available data is created during the process of publication. In this process it has to be examined, which data of a resource is extracted by SlideShare. Furthermore, some additional data can be entered by users. This data could also be published. Summarized, it has to be determined, which metadata is relevant for the AAN system.

When uploading files[1], SlideShare users can choose between various of file types, e.g. PDF, PPT (files of presentation programs such as Microsoft PowerPoint), DOC (files of document preparation systems such as Microsoft Word), or other office application formats. Some metadata can be added for these artifacts: a title, a description, a list of tags, a category, the language, a privacy statement (public, private, or visible for followers), and it can be determined, if the document is available for download. After concluding editing document details, a license can be chosen.

A published resource is displayed on the public website, where additional data is made available. The extracted text version of the resource is the only usable part. The rest of the displayed data consists of the sharing options and dynamic statistics. It is not practicable for the use in AAN.

In addition to the options of uploading resources, a user can also follow other users. These users may also follow other users. Therefore, the SlideShare network contains actor networks.

Possible elements for integration into the ontology can be determined at this point:

- The classes *Actor*, *Artefact*, *Tag*, and *Category* can be created for the storage of SlideShare.
- Semantic relations between an *Actor* and an *Artefact*, which has been added by the actor, are generated.
- Semantic relations between an *Artefact* and its *Category* and *Tags* are defined.

1 http://www.slideshare.net/singleupload

- Semantic relations between *Actors* are provided, whilst other users are followed.
- The data properties *title*, *description*, *language*, *download URL*, *license*, and *full text* of an *Artefact* can be extracted.

The privacy statement is not relevant for the extraction of data. If it is set to "private", or "visible for followers", it is not public and can not be accessed. Otherwise, it is public and will be extracted.

The next step is to clarify, how the data can be accessed.

Choice of Interface

SlideShare provides an RESTful[2] API for developers. [Sli10c] It is available free of charge for non-commercial use. Developers can apply for an API key, which is needed to access the interface.

The SlideShare API provides 14 Methods. They are listed in table 3. Seven methods can be used to access personal data of users. The authenticity is a result of the users name and password. As only public data is of interest, methods commented with "Method is private" are ignored.

Two methods are related to SlideShare groups. As the possibility of determining a users followers is given, these groups and related methods are not used. The next point is the scale of return values, which is extensive. If resources are returned, all data provided by *get_slideshow* is included. Following, there is no need for this method. At least, resources can be accessed by requests of tags and users. As a result, the method *search_slideshows* is not needed.

Three methods of the SlideShare API provide all relevant data: *get_slideshows_by_tag*, *get_slideshows_by_user*, and *get_user_contacts*. This is caused by the extensive scale of return values. The relationships of methods and classes of the AAN ontology are schematically visualized in Figure 13. Continuous lines mark access via API methods and dotted lines present returned data in XML.

To understand the relations given by the returned XML code, the response formats of the SlideShare API can be seen in Ap-

2 A RESTful API uses the principles of Representational State Transfer (REST). See http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

METHOD	USE	COMMENT
edit_slideshow	No	Method is private
delete_slideshow	No	Method is private
get_slideshow	No	Return values given by other used methods
get_slideshows_by_group	No	No use of SlideShare groups
get_slideshows_by_tag	Yes	Used for requesting slideshows by tag
get_slideshows_by_user	Yes	Used for requesting slideshows by user
get_user_campaign_leads	No	Method is private
get_user_campaigns	No	Method is private
get_user_contacts	Yes	Used for requesting contacts by user
get_user_groups	No	No use of SlideShare groups
get_user_leads	No	Method is private
get_user_tags	No	Method is private
search_slideshows	No	Return values given by other used methods
upload_slideshow	No	Method is private

Table 3: Use of SlideShare API Methods

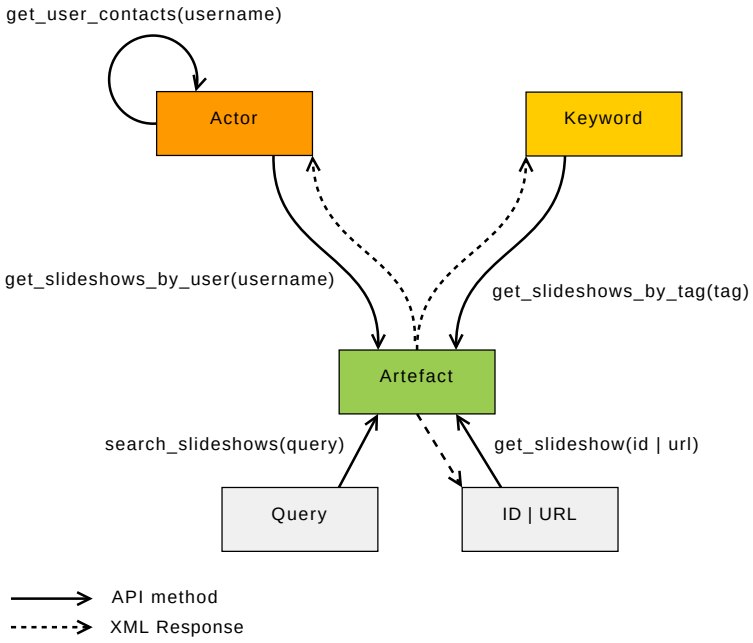

Figure 13: Use of SlideShare API methods

INTERFACE	PUBLIC DATA	WELL-DEFINED
API	✓	✓
HTML parser	✓	–

Table 4: Comparison of SlideShare Interfaces

pendix A.3. The return of a single slide show contains 32 fields. The fields include an actor (*Username*), a set of keywords (*Tag*), an id (*ID*), and an url (*URL*). The methods *get_slideshows_by_tag* and *get_slideshows_by_user* return whole sets of such slideshows. The method *get_user_contacts* mainly provides a set of usernames.

The full text, which is displayed on the website of SlideShare artifacts, is not delivered by the XML code of the API. If this is needed, a HTML parser could be developed. The HTML code of the website of SlideShare is generated dynamically and not well-defined (see Table 5). Following, changes in the code could produce interferences in the process of extraction. As artifacts of the SlideShare network are described by *title*, *description*, *tags*, and *category*, an access via a parser is not implemented.

In addition to the data properties determined in Section 5.1.2, some more properties can be extracted from the XML code and used for AAN:

- The ID of the resource (*ID*)
- The web permalink (*URL*)
- A link to a thumbnail image (*ThumbnailURL*)
- The time of creation (*Created*)
- The format of the resource (*Format*)
- The type of a resource (*SlideshowType*), which can be a presentation, a document, or a video
- The number of slides (*NumSlides*)

The property *license* is not provided by the API. Hence, it will not be integrated.

The classes, relations, and data properties delivered by the SlideShare API are integrated into the AAN ontology in Section 5.2.

5.1.3 Scribd

Data for Extraction

The choice of accessible and relevant data is composed of the data, which can be given by publishers of artifacts and the generated website output.

Users can make some statements by uploading and editing an artefact. They have two options: The candidate is asked to fill in the missing words or a selection can be made from a multiple choice proposal. The title and the description is entered in the Şmissing wordŤ field, whereby tags are separated by commas. Selected items are classified under one of the following; privacy (public or private), category and sub-category, language, default view-mode, the possibility of downloads, download formats, enabling comments, the license, and permissions of printing and copying text.

The date of publication of an artefact, as well as a download link is also disclosed on the website.

In addition, users can subscribe to other users. Such relationships build up actor networks.

The following candidates of ontology elements can be determined with this data:

- The classes *Actor*, *Artefact*, *Category*, and *Tags*.
- Semantic relations between the classes *Actor* and *Artefact*, by the publication of an resource.
- Semantic relations between an *Artefact* and its *Categories* and *Tags*.
- Semantic relations between two *Actors*, caused by subscriptions.
- Some Data properties of an Artefact: the *title* and *description*, the *language*, a *download URL*, the *download format*, a *license*, and the *publishing date*.

The field "privacy" is not relevant, because only public data can be accessed.

INTERFACE	PUBLIC DATA	WELL-DEFINED	COMPLETE DATA
API	✓	✓	–
HTML parser	✓	–	✓

Table 5: Comparison of Scribd Interfaces

Choice of Interface

Scribd provides a RESTful API similar to SlideShare. [Scr10c] This is the only alternative provided to Parsing the HTML code. A comparison of the two interfaces is shown in Table 5. The Scribd API provides methods for uploading, converting, editing, deleting, and searching documents. Three alternatives for authentication are given:

1. No authentication. Just the API key has to be passed.
2. API sessions. A session can be started by entering a users name and password.
3. Use of an additional parameter. This is for integration of ones own system.

Alternative three provides options for integration of a system with ones own user management. This option can only be used in further handling to store data outside the AAN system, but not to extract data within the local AAN. Alternative two can be used for systems to extend Scribd services. This is not the intention of the concept of AAN. Hence, the first method has been chosen.

The Scribd API mainly provides methods for document management of authorized users. An overview of the provided methods is given in Table 6. Only the method *docs.search* can be used efficiently. It contains a parameter query, which can be extended by the parameters *title*, *content*, *tags*, and *description*. It returns a set of XML fields, listed in Appendix A.4.

The methods *docs.getConversionStatus* and *thumbnail.get* could be used for additional information. The return value of *docs.getConversionStatus* provides information about the current state of access. It is returned, if the resource is in the initial *PROCESSING* state, already *DISPLAYABLE*, or if the document is fully indexed and the upload is completely *DONE*. The method *thumbnail.get* returns an URL to an image, which can be used

METHOD	USE	COMMENT
docs.upload	No	Document management
docs.uploadFromUrl	No	Document management
docs.getList	No	Requires user authorization
docs.getConversionStatus	Yes	Status of upload
docs.getSettings	No	Requires user authorization
docs.changeSettings	No	Document management
docs.getDownloadUrl	No	Requires user authorization
docs.getStats	No	Provides only the number of reads
docs.delete	No	Document management
docs.search	Yes	Provides requests for artifacts
docs.getCategories	No	Extracted at runtime
docs.featured	No	Not related to categories or tags
docs.browse	No	Browsing of categories
docs.uploadThumb	No	Document management
docs.getCollections	No	Document management
docs.addToCollection	No	Document management
docs.removeFromCollection	No	Document management
thumbnail.get	Yes	Image for visualization
user.login	No	User administration
user.signup	No	User administration
user.getAutoSigninUrl	No	User administration
security.setAccess	No	Document management
security.getDocumentAccessList	No	Document management
security.getUserAccessList	No	Document management

Table 6: Use of Scribd API Methods

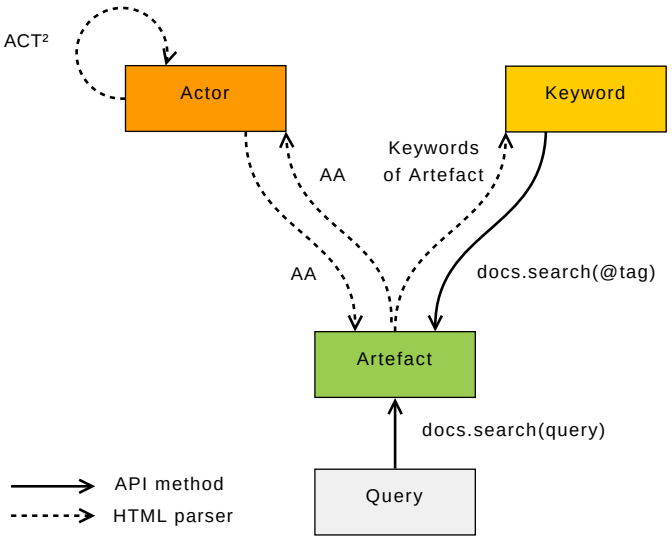

Figure 14: Use of Scribd interfaces

for visualization of the data. It is also provided by the method *docs.search*.

As the API mainly provides an access by the input of tags, an additional HTML parser has to be developed. It has to provide all types of access, excepted the access of artifacts by tags. Figure 14 shows the use of the interfaces.

After an analysis of the structure of the HTML code, the following elements will be integrated:

- The classes *Actor*, *Artefact*, and *Tag*.
- Semantic relations of the type *ACT²*, which describe subscriptions of *Actors*.
- Semantic relations of the type *AA*, between *Actors* and *Artefacts*.
- Semantic relations of the type *AA*, between an *Actor* and an *OnlineArtefact*, which describes the personal website of the actor.
- Semantic relations between *Actors* and *Tags*.
- A set of data properties for an *Actor*: *name*, *location*, and *biography*.
- A set of data properties for an *Artefact*: *document id*, *title*, *description*, *license*, *thumbnail*, *number of pages*, *download formats*, *upload date*, and *download URL*.

The next step is to integrate the available elements into the ontology.

5.2 SOLUTION DESIGN

Before the solution design is described, some enhancements of the ontology are introduced. Afterwards, determined classes, relations, and data properties of the data sources are integrated into the ontology. Finally, software components for the respective services are developed.

5.2.1 Enhancements of the ontology

Various possibilities for improvement arose during the development and use of AAN and during research on this thesis.

The idea of refining the class *Keyword* in the base ontology was derived during the integration of additional networks.

Furthermore, compatibility with several metadata vocabularies was necessary. Consequently, a new version of the ontology was developed by the AAN team and external participants.

Figure 15 shows the base ontology of the revised version 2. The class *Keyword* has now been divided into two subclasses, *Category* and *Tag*. By these means it can be differentiated, whether predetermined categories or arbitrary tags are needed. Moreover, numerous semantic relations were added to provide a higher level of compatibility with some proven vocabularies: Dublin Core Metadata Initiative (DCMI) Metadata terms [Dub10], the Friend of a Friend (FOAF) project [FOA10], and the terms of Semantically-Interlinked Online Communities (SIOC) [SIO10]. This enhancements provide uniformity for the work with different components. For instance, two semantic relations *hasSection* and *hasChapter* of different external sources can be handled uniformly by using or specializing the relation *hasPart*. If relations of the class *hasPart* are requested later, a set of returned data provides a higher number of results.

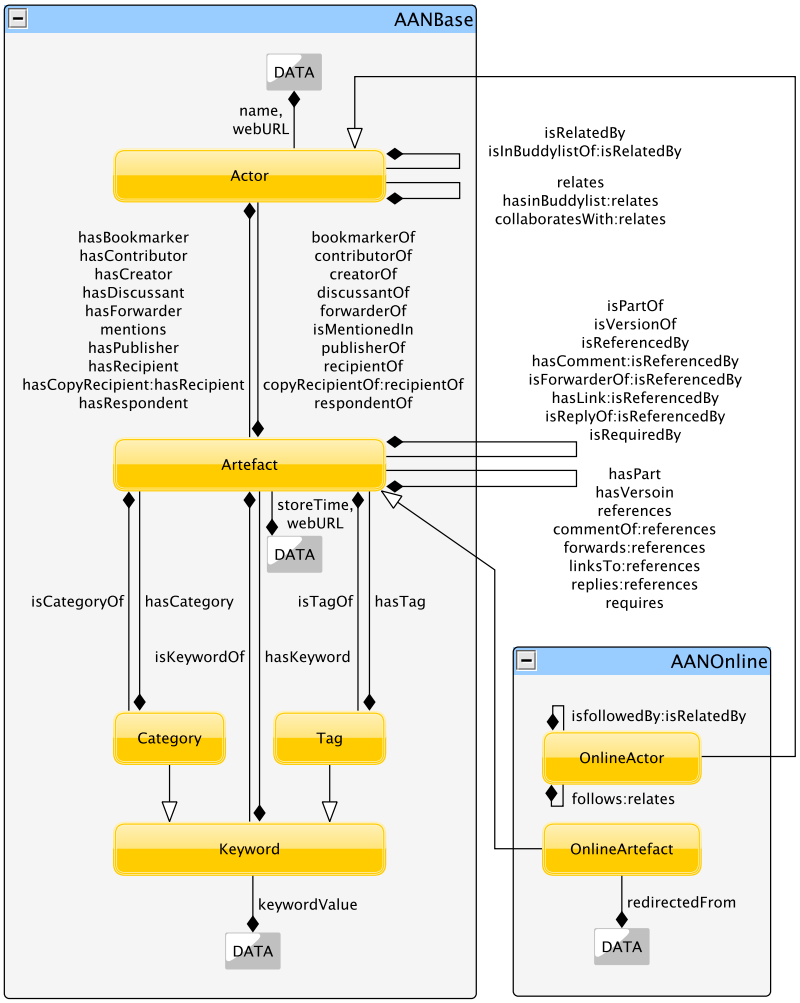

Figure 15: Version 2 of AAN ontology

5.2.2 Integration of Delicious

Integration into the AAN Ontology

A concept for the integration of Delicious data, analyzed in Chapter 5.1.1, into the AAN ontology is worked out in this section.

For this, the existing ontology has to be extended. It has to be initially determined, which existing classes and relations of the base ontology can be used. Furthermore, the expandability of the system has to be taken into account. During the design process, created classes should be reusable. Figure 16 shows a visualization of the completed integration of Delicious.

USE AND MAINTENANCE OF EXPANDABILITY Attention has to be payed as to the expandability of the data structure for future application. Possibly, other bookmarking services such as Delicious will be integrated.

As Delicious can be seen as a web service and all classes can be accessed via the WWW, they will have be classified as subclasses of the *AANOnline* block in the AAN ontology.

On the other hand, the Delicious block itself has to be a subsection of a general block for bookmarks. Social bookmarking services such as Delicious, Simpy[3], or Mister Wong[4] mostly are using tags to describe bookmarks. Additionally, actors and the bookmarks themselves form parts of these networks. As a result, a new sub-block *AANSocialBookmarks* of *AANOnline* was inserted. *AANSocialBookmarks* consists of three classes: *BookmarkActor*, *BookmarkArtefact*, and *BookmarkTag*. The new classes extend classes of *AANOnline*, only *BookmarkTag* is extending the class *Tag* of the block *AANBase*, because no corresponding class is defined for *AANOnline*.

INTEGRATION OF EXTRACTED DATA After the integration of new classes for the field of social bookmarking, specialized sub-classes for Delicious have to be converted. For this purpose, a block *AANDeliciousBookmarks* consisting of the classes *DeliciousActor*, *DeliciousBookmark*, and *DeliciousTag* has been created. These classes

3 Simpy, http://www.simpy.com/
4 Mister Wong, http://www.mister-wong.de/

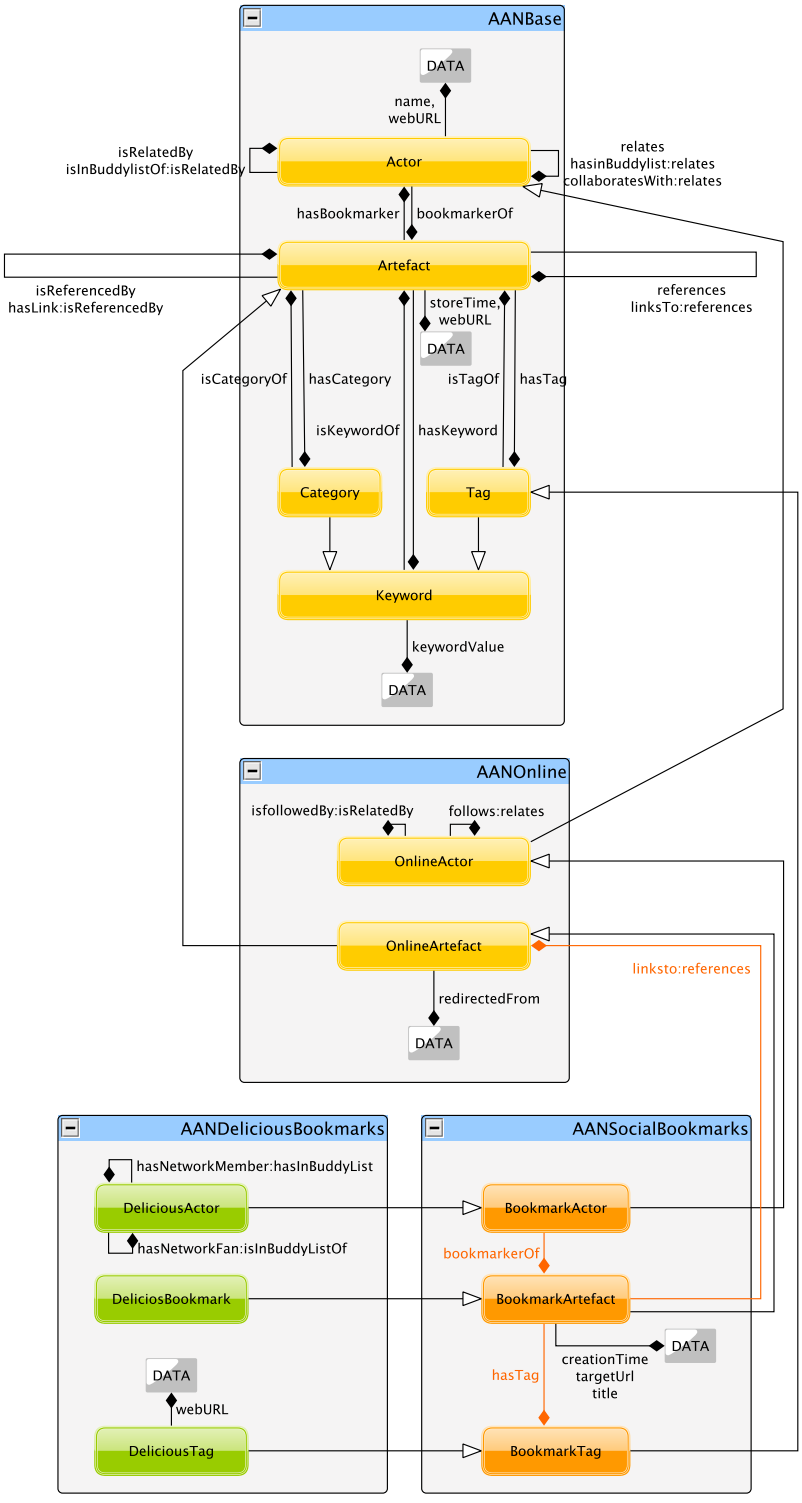

Figure 16: AAN ontology: Integration of Delicious

are direct sub-classes of the three classes in the *AANSocialBookmarks* block.

Semantic relations The semantic relations found in Chapter 5.1.1 have to be integrated before storing the networks relationship.

Two existing semantic relations (and their inverses) can be used through inheritance from *AANBase* and *AANOnline*. A *BookmarkActor* is a *bookmarkerOf* (inverse: *hasBookmarker*) a *BookmarkArtefact*. This corresponds to the relations *hasArtefact* (and *isArtefactOf*) of the analysis. Further, a *BookmarkArtefact* is related via *hasTag* (inverse: *isTagOf*) to a *BookmarkTag*. This corresponds to the relations *hasKeyword* and *isKeywordOf* of the analysis.

Four additional semantic relations have to be added to the ontology. The two ACT^2 relations are included by *hasNetworkMember* and *hasNetworkFan* with domain and range *DeliciousActor*. They are specializations of the relations *hasInBuddyList* and *isInBuddyList* of the base ontology.

Finally, the semantic relations between artifacts have to be realized. The respective artifacts of domain and range are of different types. Bookmarks of the Delicious network are instances of the class *DeliciousBookmark*. On the other hand, an artefact which is related by a bookmark is more general. As this could be any document in the WWW, the range of the relation is of the type *OnlineArtefact*. The types of the semantic relations are chosen from the base ontology: *linksTo* inherits from *references* and *hasLink* inherits from *isReferencedBy*.

Data properties Data properties of the classes have to be defined to complete the integration. Some properties are given by the hierarchy:

- Actors already have the data properties *name* and *webUrl*.
- Artifacts take over the properties *storeTime* and *webUrl*.
- Tags have the property *keywordValue*.

The property webUrl has also been defined for tags, allowing actors as well as artefacts and tags to have an access point to retrieve information via the WWW. For instance, a stored *webUrl* for the Delicious tag "socialmedia" is `http://delicious.com/tag/socialmedia`.

In addition to the existing data properties, some more relevant metadata to describe bookmarks is delivered by the Delicious feeds. This data should also be delivered by other social bookmarking networks, therefore it is referenced by *BookmarkArtefact* instead of *DeliciousArtefact*. The data consists of the properties *creationTime*, *targetUrl* and *title*. All properties are delivered by the scheme "bookmarks" of the JSON return values, listed in Appendix A.1.

5.2.3 Integration of the document networks

SlideShare and Scribd provide data regarding documents. This common ground should be reflected in the ontology. As a result, the classes of both networks become sub-parts of a common block for document networks. Figure 17 visualizes the document part of the ontology.

The document block *AANDocument* consists of two classes, *DocumentActor* and *DocumentArtefact*. They are derived form *OnlineActor* and *OnlineArtefact* of *AANOnline*. This was done, because all resources of the networks can be accessed in the WWW. The class *DocumentArtefact* is related to several data properties. These properties exist in the artefact classes of SlideShare and Scribd. By inheritance, *SlideShareArtefact* and *ScribdArtefact* are related to the data properties *creationTime*, *description*, *downloadUrl*, *formats*, *id*, *pages*, *title*, and *thumbnailUrl*.

Additionally, each network has its own definitions. These are described in the next sections.

SlideShare

Further classes for tags and categories have been defined in addition to the classes *SlideShareActor* and *SlideShareArtefact*. SlideShare tags are defined by the class *SlideShareTag* and categories by *SlideShareCategory*. They are directly derived form the base classes *Tag* and *Category*. Artifacts of the SlideShare network contain details, which are not given by inheritance form the class *DocumentArtefact*. Accordingly, two additional data properties *type* and *language* have been added.

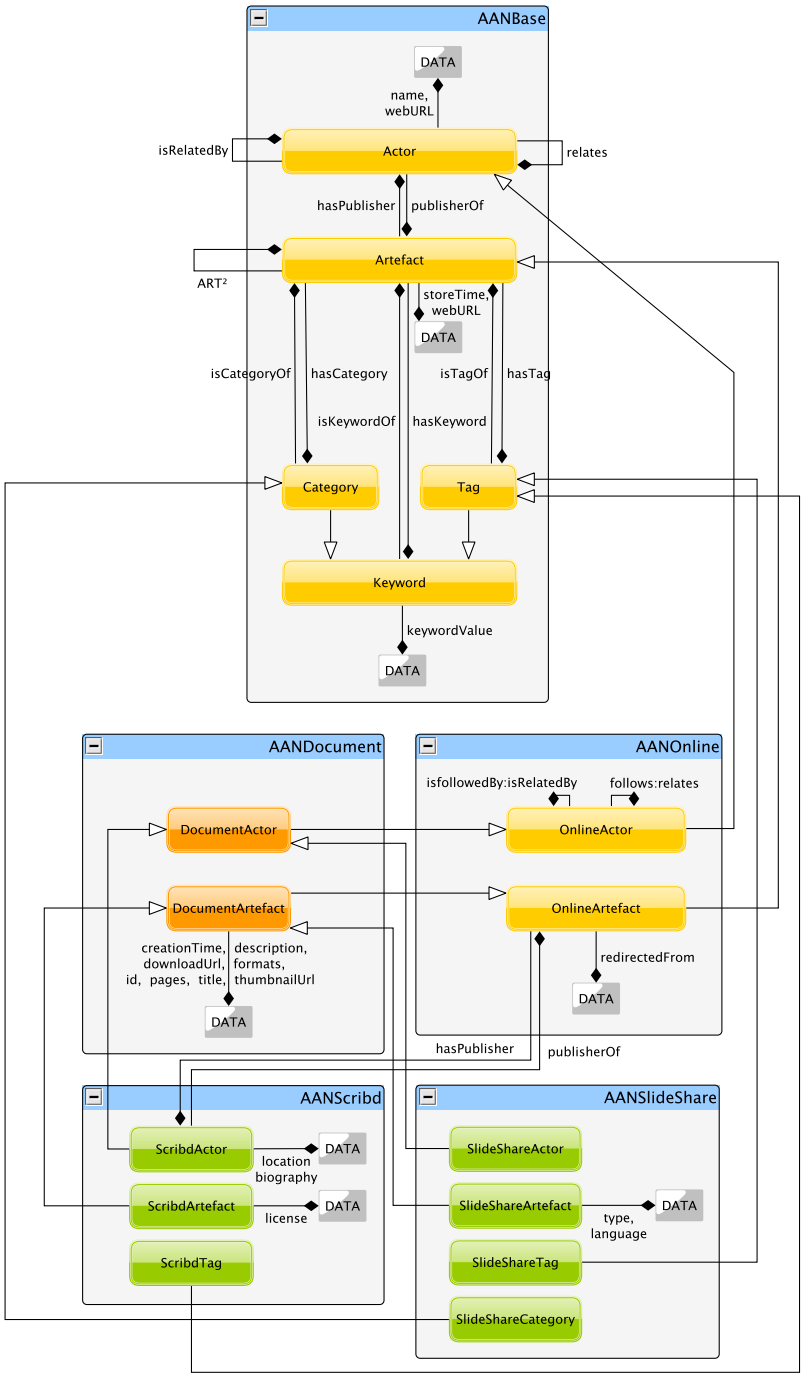

Figure 17: AAN ontology: Integration of document networks

Scribd

The *AANScribd* block contains an additional class *ScribdTag*, which is derived from the class *Tag* of the base ontology. Further, some data properties are defined. Artifacts of the Scribd network contain a license. Hence, the class *ScribdArtefact* has a data property *license*. For actors in the Scribd network, the details location, biography, and website can be defined. As a result, the class *ScribdArtefact* has two data properties *location* and *biography*. The website of a Scribd actor has been defined in another way. In the ontology, websites are represented by *OnlineArtefacts*. As a result, actor websites have been realized as semantic relations between *ScribdActor* and *OnlineArtefact*.

5.2.4 Draft of software components

In this section some preparation for a practical implementation is provided. Following, a draft of the components is described by the definition of interfaces.

The realization of the components is oriented to the architecture shown in Figure 9 on page 25. It consists of two OSGi bundles for each of the three networks.

On the one hand, a *CrawlerManager* controls the process of crawling and is responsible for the handling of new crawling tasks. Additionally, it uses data, extracted by a *Parser*, to control the crawling process. The webservice, displayed in the architecture in Figure 9, is also a component of the *CrawlerManager* bundle.

On the other hand, a *Parser* is necessary and responsible for the extraction of data. A *Parser* determines and extracts objects of classes, contained in the ontology. Furthermore, it extracts semantical relations and data properties.

A *Crawler* component is not necessary and will not be developed. The *CrawlerManagers* are using existing *Crawlers* to access data. An insight to the crawling process is defined before the components are planned.

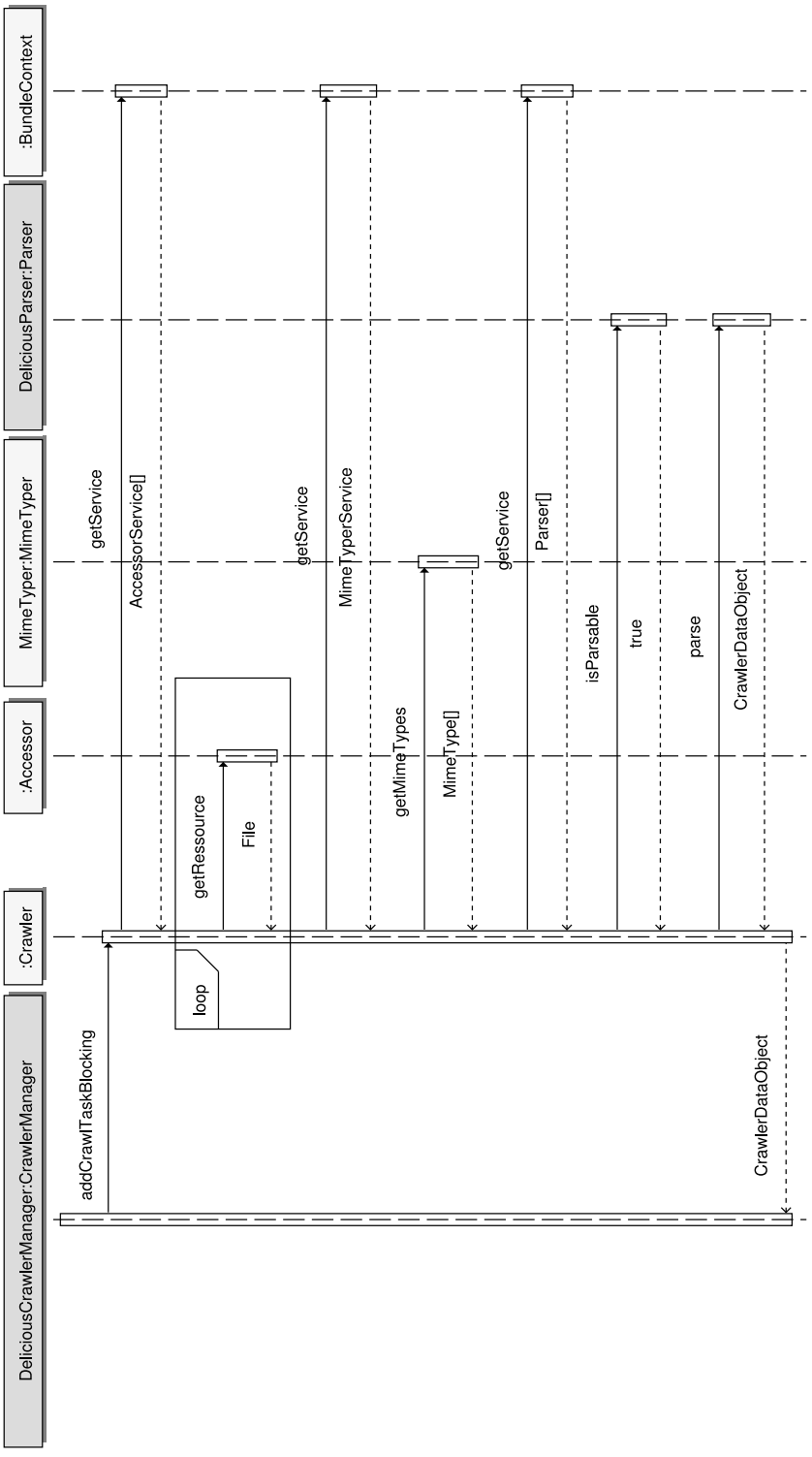

Figure 18: Working chain: Successful Crawl of a Delicious resource

The crawling process

The crawling process integrates several components which are part of a working chain as described here by a visualization. Figure 18 depicts an instance of a successful crawl of a Delicious resource. Firstly, the *DeliciousCrawlerManager* adds a new crawl task. The task is started and controlled by a *Crawler* component which needs access to bundles of different types. The accessing is done by provided services. References to the services can be accessed via the OSGi framework.

The first part of the working chain is to access the resources. It is been done by an *Accessor* component. After the *Accessor* component stored the resource locally, the *MimeTyper* detects the MIME types of it. The MIME types are used to choose a specialized parser to process the resource. If no specialized parser is found, a general parser would process the task. In the example, the method *isParsable* is called at the *DeliciousParser*. As the *DeliciousParser* can process the resource and returns the value *true*. If it would return *false*, the method *isParsable* would be called at other parsers. Finally, the *Crawler* component starts the parsing process by calling the method *parse* at the *DeliciousParser*. A *CrawlerDataObject* with information of the parsing process is returned to the *DeliciousCrawlerManager*.

The controlling component CrawlerManager

A *CrawlerManager* controls the process of crawling. It can be accessed by a webservice and is specialized for a specific type of network. In this thesis, three types of networks should are integrated. The three types of *CrawlerManagers* developed are as follows: A *DeliciousCrawlerManager*, a *SlideShareCrawlerManager*, and a *ScribdCrawlerManager*. Each of these CrawlerManagers are realized as an OSGi bundle and provide services which are accessed by interfaces. The procedure of accessing several networks are similar and the interfaces provide the same methods. For each network, a method to add, get, and remove a task is defined. Furthermore, methods for the special tasks of crawling an actor, artefact, and tag will be implemented. Additionally a method for list all unfinished tasks is defined. The Scribd network differs from the other networks, because it has

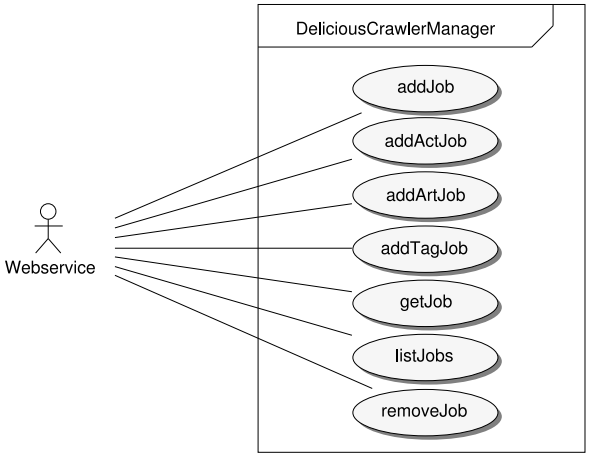

Figure 19: Use Cases of component DeliciousCrawlerManager

an additional class *Category*. A crawl of a whole category is no special task, it can be started by adding a general task.

For categories no additional use case is defined. Figure 19 shows the provided methods in the form of use cases.

The seven use cases provide the following functions:

ADDJOB Adds a new task. Tasks mainly consist of an URI of a resource, which has to be examined.

ADDACTJOB Adds a new task for crawling an *Actor*. By this, related properties, artifacts and other actors are examined.

ADDARTJOB Adds a new task for crawling an *Artefact*. By this, related properties, actors and tags are examined.

ADDTAGJOB Adds a new task for crawling an *Tag*. By this, related artifacts are examined.

GETJOB Returns the CrawlerDataObject related to the task.

LISTJOBS Returns a list of tasks, which are not finished.

REMOVEJOB Removes the given task from the set of unfinished tasks.

The *CrawlerManagers* control the extraction of data. The extraction itself is done by *Parsers*.

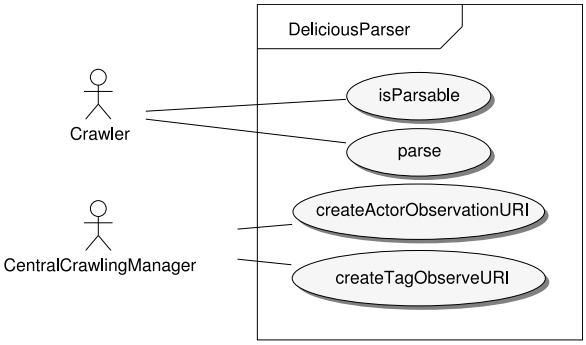

Figure 20: Use Cases of component DeliciousParser

The extraction component Parser

A *Parser* extracts relevant information of a resource. Information is added to the model, structured by the ontology. Additionally, data for the crawling process is returned.

A parser has to be developed for each of the networks: a *DeliciousParser*, a *SlideShareParser*, and a *ScribdParser*. Parser provide an interface, which is mainly used by two components: The *Crawler* and the *CentralCralingManager*.

The *Crawler* is controlled by a *CrawlerManager*. The CentralCrawlingManager has been developed for observation and to persist data. It needs an interface for actors and tags, which provides methods to return URIs. Later, these URIs can be passed to *Parsers* for extracting data of the respective actors and tags. Figure 20 displays the provided methods in the form of use cases.

The four use cases provide the following functions:

ISPARSABLE Returns a boolean value, which gives evidence about the suitability of parsing the resource.

PARSE Starts the parsing process, in which data is extracted and information about the parsing process is returned.

CREATEACTOROBSERVATIONURI Returns an URI, which can be used to parse a given actor.

CREATETAGOBSERVEURI Returns an URI, which can be used to parse a given tag.

The described interfaces only provide a small set of methods. Numerous opportunities for crawling processes can be developed by specifying parameters.

5.3 SUMMARY: SUCCESSFUL PREPARATION FOR A PRACTICAL APPLICATION

The analysis of the networks data and interfaces provides a detailed preparation for the practical integration of data. Firstly, relevant data of the several sources have been chosen. In the second step, interfaces were chosen. The combination of the desired data with interfaces, provided by the networks, was limited by restrictions. Despite this, the majority of the necessary data has been integrated into the ontology.

New components for crawling networks and parsing data have been developed. This was possible, because the AAN framework was conceived to accommodate expandability. The next step is an implementation of the defined component interfaces. In this practice the network interfaces and the integration of objects into the AAN model are demonstrated.

6

DETAILS OF IMPLEMENTATION

The description of the implementation starts with some general aspects. Afterwards, the implementation of Delicious components is described more detailed as the document networks, as the access of data by Delicious feeds is more complex. Finally, a practical application is introduced.

6.1 THE IMPLEMENTATION IN GENERAL

The implementation is based on the interfaces defined in Section 5.2.4. The interfaces of the components *CrawlerManager* and *Parser* are implemented for each of the networks. Firstly, some general aspects of the implementation are described.

6.1.1 Access by webservices

The *CrawlerManager* components can be accessed by webservices. Webservices are provided for the tasks *addJob*, *addActJob*, *addArtJob*, *addTagJob*, *getJob*, *listJobs*, and *removeJobs*. URLs provided by the webservices are listed in Appendix A.5.

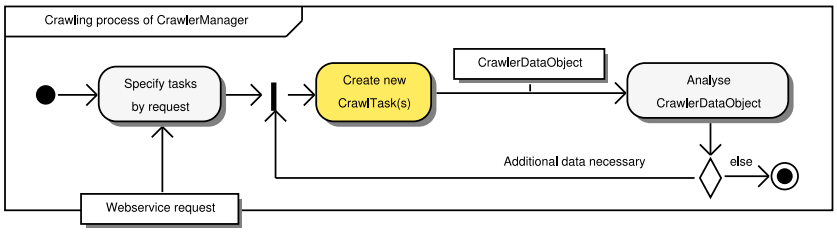

Figure 21: General crawling process of CrawlerManagers

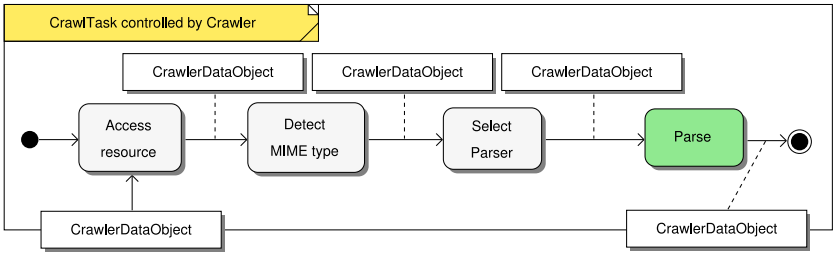

Figure 22: CrawlTask controlled by a Crawler

6.1.2 CrawlerManager and Crawler

The structure of a crawling process and the working chain were introduced in Section 4.2 and visualized in Section 5.2.4 by a sequence diagram. The following activity diagrams are explaining the implementation.

Figure 21 shows a general workflow of the implemented *CrawlerManagers*. A crawling process is initialized by the access of a webservice. Firstly, a *CrawlerManager* analyzes the parameters of a webservice and creates initial *CrawlTasks*. A finished *CrawlTask* returns a *CrawlerDataObject*, which can be analyzed to start further *CrawlTasks*.

A *CrawlTask* is handled by a Crawler, as shown in Figure 22. A resource is accessed, examined and parsed. All stages are using the *CrawlerDataObject* to share data. Finally, it is returned to the *CrawlerManager*. The process of parsing is relevant for the implementation and described for each integrated data source.

6.1.3 Parser

A Parser is implemented for each data source. It is the last component of the working chain. *Parsers* analyze resources, store data, and add data to the *CrawlerDataObject*, which is used by *CrawlerManager*s to control the process of crawling.

6.2 DELICIOUS

Data of the Delicious network is accessed by feeds. This is a disadvantage, because only the 100 latest bookmarks of a request are provided. During the process of crawling, all related objects to a requst should be parsed. Therefore, the *DeliciousCrawlerManager* was designed to make gradual refinements. Furthermore the feeds and feed patterns (see Figure 12 on page 43) are combined.

6.2.1 The DeliciousCrawlerManager

The workflow of the *DeliciousCrawlerManager* is visualized in Figure 23. Figure 23-1 shows the initial activities. Firstly, it is determined, if a special actor, artefact, tag, or a general task was requested. General requests are handled as described in Section 6.1.2. For the remaining types of tasks, special workflows have been implemented.

The crawling process for an actor is presented in Figure 23-2. The network of an actor is crawled by using the feeds *networkFans* and *networkMembers*. A *CrawlTask* is created for each feed. The *CrawlTasks* are controlled by a *Crawler* (see Section 6.1.2). At the end of the working chain, the feeds are parsed by the *DeliciousParser*. Bookmarks of the *Actor* are requested by the feed *bmByUser*. After the request, it is determined, if 100 bookmarks were received. If not, the crawling process is finished. If the feed provided 100 bookmarks, possibly more bookmarks could be requested. In that eventuality, a *CrawlTask* for the feed *tagsByUser* provides all tags of the actor. Next, a recursive crawl is started.

Figure 23-5 shows the activities of a recursive crawl for bookmarks of an actor. At first, the username of the actor and his tags are extracted from the *CrawlerDataObject*. If the set of tags is empty, no bookmarks are described with tags and the task is finished. Otherwise, each tag is combined with the username of the actor and bookmarks related to this combination are requested by the feed *bmByUserAndTags*. If the result of this request counts less than 100 bookmarks, all related tags have been received and this task is finished. If the returned set of bookmarks contains 100 entries, the request will be refined. A *CrawlTask* for tags related to the current combination of username and one or more

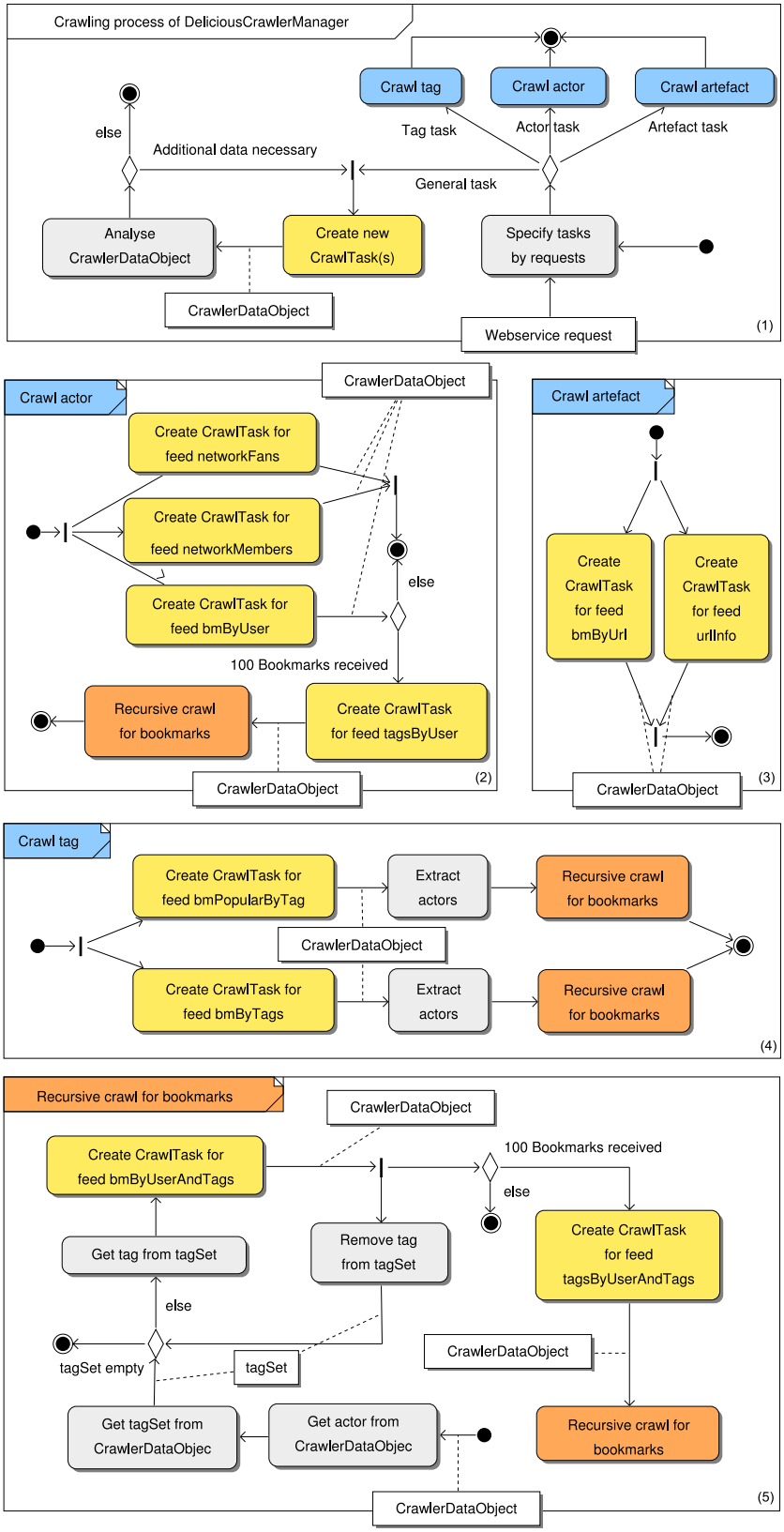

Figure 23: Crawling process of the DeliciousCrawlerManager

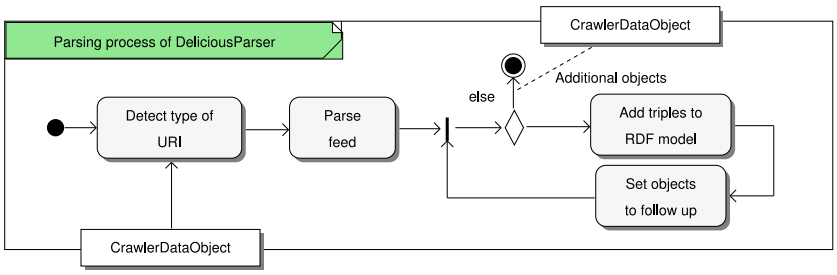

Figure 24: Parsing process of the DeliciousParser

tags is started. The corresponding feed is *tagsByUserAndTags*. At least, a recursive crawl with this new set of tags on the one hand, and the username and tags of the current request on the other hand is started. This refinement is repeated until all requests return less than 100 bookmarks or no additional tags can be determined.

The next special task is a crawl of an artefact. Figure 23-3 shows the requests of *bmByUrl* and *urlInfo*. The returned data provides up to 100 bookmarks for a requested URL and a set of popular related tags.

Finally, the workflow for the crawling process of a tag is presented in figure 23-4. The two feeds *bmPopularByTag* and *bmByTags* provide sets of bookmarks. Each of the bookmarks was created by an actor. The usernames of these actors and combined with the initial tag to start recursive crawls.

Each of the used *CrawlTasks* is finally parsed by the *DeliciousParser*, which extracts data.

6.2.2 The DeliciousParser

Each *CrawlTask* mainly consists of an URI. The URI of a task is stored in the *CrawlerDataObject* and analyzed by the *DeliciousParser* to detect a feed pattern as presented in Figure 24. If a pattern was detected, the *DeliciousParser* parses the content of the feed. The content is also contained in the *CrawlerDataObject*. If data of interest is found, RDF triples are extracted are stored. Furthermore, actors, bookmarks, or tags are saved for further proceeding. Finally, if all data is parsed, the *CrawlerDataObject* is returned.

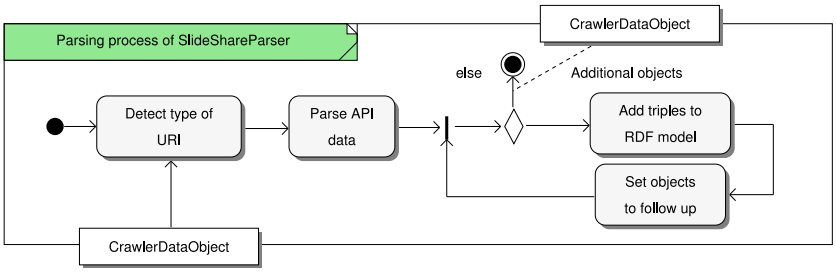

Figure 25: Parsing process of the SlideshareParser

6.3 SLIDESHARE

6.3.1 The SlideShareCrawlerManager

Data of SlideShare is accessed by an API. This provides a comfortable access. The workflow of the *SlideShareCrawlerManager* was designed how described in Section 6.1.2. Requests are structured accordingly the methods, introduced in Section 5.1.2.

6.3.2 The SlideShareParser

Activities of the *SlideShareParser* are shown in Figure 25. Firstly, the URL is analyzed. The API of SlideShare uses the REST principles. Hence, the content can be classified without problems. The content itself is based on XML. The SlideShare data is parsed and data for the *SlideShareCrawlerManager* is returned by the *CrawlerDataObject*.

6.4 SCRIBD

6.4.1 The ScribdCrawlerManager

The *ScribdCrawlerManager* was implemented as described in in Section 6.1.2. Data of the Scribd network is accessed by two interfaces. Accordingly, CrawlTasks for Scribd are specified by two alternatives: If artefacts are requested and the input of the request consists of tags, an URI for an API call is generated. Oth-

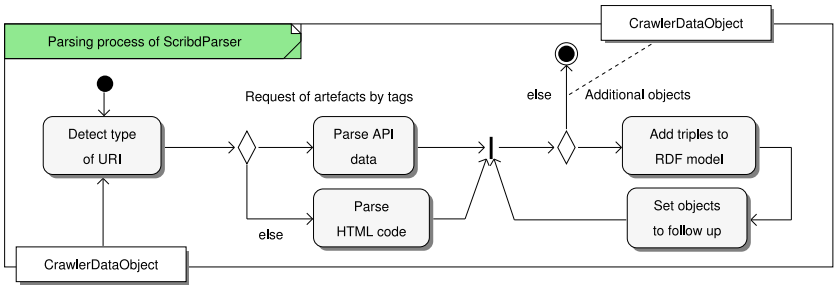

Figure 26: Parsing process of the ScribdParser

erwise, an URL of the Scribd website, related to the requested object, is used.

6.4.2 The ScribdParser

As shown in Figure 26, the process of parsing is build up similar to the *ScribdCrawlerManager*. If the URL of the current resource refers to the API, an artefact has been requested. Following, the properties *document id*, *title*, *description*, *license*, *thumbnail*, *number of pages*, *download formats*, *upload date*, and *download URL* are extracted. Otherwise, HTML code is parsed. This is done by following the hierarchical DOM structure of the website. The code is structured and nodes provide IDs. the desired data can be parsed.

6.5 PRACTICAL APPLICATION

To evaluate the developed components, a small test set was chosen. In the following, the sets are described and results presented.

6.5.1 Test sets

Three test instances were chosen:

- A crawl task of the Delicious network without restrictions
- A crawl task of the SlideShare network without restrictions

- A crawl task of the SlideShare network, restricted to the tag *fsln10*

These instances were chosen to test the Delicious components and the components of a document network. Furthermore, a comparison of the test sets is possible. The results were received using webservices of the *Datastore* component:

- datastore/actor/count
- datastore/artefact/count

The test instances were started with an internal crawl-depth of *20* and a search input *fsln10*. The system was running for 10 minutes.

6.5.2 Test results

Delicious: Performance

As a result of the Delicious crawl task, *280* actors and *950* artifacts were received. The low numbers could be caused by restrictions of the Delicious feeds. In a previous test set, *1849* actors could be received in 5 minutes.

SlideShare: Performance

During the SlideShare crawl task, *2149* actors and *3092* artifacts were received.

SlideShare: Completeness

The restricted SlideShare crawl task received *8* actors and *9* artifacts. This result was verified by the website `http://www.slideshare.net/search/slideshow?q=%40tag+fsln10`. All recources were received.

6.6 SUMMARY: DIVERSE LEVELS OF COMPLEXITY

The implementation of the components varied in complexity. The implemented Delicious components required extensive workflows, whereas the document networks could be accessed directly. Tests of the SlideShare components provided satisfactory results, artifacts of a requested tag were crawled completely. The Test results of Delicious components provided little satisfaction.

7

CONCLUSION AND OUTLOOK

7.1 CONCLUSION

During the work on this thesis, I acquired various kinds of knowledge. This includes theoretical, technical, and practical topics. Theoretical topics are Science 2.0 and awareness. Some more technical is the field of semantics and the approaches of representation. Technical areas are the architecture of the OSGi platform and the AAN framework. I gained some practical experience by designing extensions of the AAN ontology and the implementation of OSGi components of the AAN system.

The conception and practical implementation of the access of APIs, feeds and HTML code was differently. Data offered by APIs were received directly. The implementation of the HTML parser seemed to be difficultly, but the implementation was supported by Java components. The application of Delicious feeds was more complex, caused by different restrictions.

Finally, now I know how much time it takes, to write a thesis in another than ones native language.

7.2 OUTLOOK: FUTURE WORKS

In future works, the developed components can be extended to store full texts. The SemSim analyzer [Moi10] can be used to compute similarities between documents. The Delicious components can be extended by a HTML parser, to receive all artifacts related to tags more efficiently. A visualization of stored artifacts is another option for extension. Such a visualization could provide a view of all resources of a person, including the ac-

tors in different networks. The AAN system can be extended by some more social networks specialized of communication. On the other hand, a development of an interface to offer stored data to other systems is also possible. The AAN ontology can be extended to provide direct relations between the classes *Actor* and *Keyword*. If data can be persisted, a comparison of states of different points of time would be possible. By such states, the possibility of simulations arises. Maybe one day the AAN system can be used to make predictions of a users behavior.

A

APPENDIX

A.1 RETURN VALUES OF DELICIOUS FEEDS

Listing 1: Scheme of JSON return values for bookmarks

```
[{
"u":"<URL>",
"d":"<DESCRIPTION>",
"t":["<TAG 1>","<TAG 2>",*],
"dt":"<YYYY-MM-DDTHH:MM:SSZ>",
"n":"<NOTES>",
"a":"<AUTHOR>"},
*]
```

Listing 2: Scheme of JSON return values for tags

```
{
"<TAG 1>":<NUMBER OF USES>,
"<TAG 2>":<NUMBER OF USES>,
*}
```

Listing 3: Scheme of JSON return values for users in network

```
[
{"user":"<USERNAME 1>","dt":"<YYYY-MM-DDTHH:MM:SSZ>"},
{"user":"<USERNAME 2>","dt":"<YYYY-MM-DDTHH:MM:SSZ>"},
*]
```

Listing 4: Scheme of JSON return values for url information

```
[{
"hash":"<MD5 OF URL>",
"title":"<TITLE>",
"url":"<URL>",
"total_posts":<NUMBER>,
"top_tags":{
 "<TAG 1>":<NUMBER OF USES>,
 "<TAG 2>":<NUMBER OF USES>,
*}
}]
```

A.2 DELICIOUS FEED PATTERNS

NO.	URL PATTERN	DESCRIPTION
1	/	Bookmarks from the hotlist
2	/recent	Recent bookmarks
3	/tag/{tag[+tag+...+tag]}	Recent bookmarks by tag
4	/popular	Popular bookmarks
5	/popular/{tag}	Popular bookmarks by tag
6	/alerts	Recent site alerts
7	/{username}	Bookmarks for a specific user
8	/{username}?private={key}	Private bookmarks for a specific user
9	/{username}/{tag[+tag+...+tag]}	Bookmarks for a specific user by tag(s)
10	/{username}/{}?private={key}	Private bookmarks for a specific user by tag(s)
11	/userinfo/{username}	Public summary information about a user
12	/tags/{username}	A list of all public tags for a user
13	/tags/{username}/{tag[+tag+...+tag]}	A list of related public tags for a user tag comination
14	/subscriptions/{username}	Bookmarks from a user's subscriptions
15	/inbox/{username}?private={key}	Private feed for a user's inbox bookmarks from others
16	/network/{username}	Bookmarks from members of a user's network
17	/network/{username}?private={key}	Bookmarks from members of a user's private network
18	/network/{username}/{tag[+tag+...+tag]}	Bookmarks from members of a user's network by tag
19	/network/{username}/{}	Bookmarks from members of a user's private network by tag
20	/networkmembers/{username}	A list of a user's network members
21	/networkfans/{username}	A list of a user's network fans
22	/url/{url md5}	Recent bookmarks for a URL
23	/urlinfo/{url md5}	Summary information about a URL

Table 7: URL patterns of Delicious feeds [Yah10b]

A.3 RESPONSE FORMATS OF THE SLIDESHARE API

Listing 5: Response XML Format for method get_slideshow of SlideShare API [Sli10c]

```
<Slideshow>
  <ID>{ slideshow id }</ID>
  <Title>{ slideshow title }</Title>
  <Description>{ slideshow description }</Description>
  <Status>{ 0 if queued for conversion, 1 if converting, 2 if converted,
            3 if conversion failed }
  </Status>
  <Username>{ username }</Username>
  <URL>{ web permalink }</URL>
  <ThumbnailURL>{ thumbnail URL }</ThumbnailURL>
  <ThumbnailSmallURL>{ URL of smaller thumbnail }</ThumbnailSmallURL>
  <Embed>{ embed code }</Embed>
  <Created>{ date slideshow created }</Created>
  <Language>{ language, as specified by two-letter code }</Language>
  <Format>ppt (or pdf, pps, odp, doc, pot, txt, rdf) </Format>
  <Download>{ 1 if available to download, else 0 }</Download>
  <DownloadUrl>{ returns if available to download }</DownloadUrl>
  <SlideshowType>{ 0 if presentation, 1 if document, 2 if ,
                3 if video }</SlideshowType>
  <InContest>{ 1 if part of a contest, 0 if not }</Download>

  <UserID>{ userID }</UserID>
  <ExternalAppUserID>{ ExternalAppUserID if uploaded using an
                              external app }</ExternalAppUserID>
  <ExternalAppID>{ ExternalAppID for the external app }</ExternalAppID>
  <PPTLocation>{ PPTLocation }</ExternalAppUserID>
  <StrippedTitle>{ Stripped Title }</StrippedTitle>
  <Tags>
  <Tag Count="{ number of times tag has been used }" Owner="{ 1 if owner
                has used the tag, else 0 }">{ tag name }
  </Tag>
  </Tags>
  <NumDownloads>{ number of downloads }</NumDownloads>
  <NumViews>{ number of views }</NumViews>
  <NumComments>{ number of comments }</NumComments>
  <NumFavorites>{ number of favorites }</NumFavorites>
  <NumSlides>{ number of slides }</NumSlides>
  <RelatedSlideshows>
    <RelatedSlideshowID rank="{ rank, where 1 is highest}">
    { slideshow id } </RelatedSlideshowID>
  </RelatedSlideshows>
  <PrivacyLevel>{ 0, or 1 if private }</PrivacyLevel>
  <SecretURL>{ 0, or 1 if secret URL is enabled }</SecretURL>
  <AllowEmbed>{ 0, or 1 if embeds are allowed }</AllowEmbed>
  <ShareWithContacts>{ 0, or 1 if set to private, but contacts can view
                       slideshow }
  </ShareWithContacts>
</Slideshow>
```

Listing 6: Response XML Format for method get_slideshows_by_user of SlideShare API [Sli10c]

```
<User>
  <Name>{ username_for }</Name>
  <Count>{ Number of Slideshows }</Count>
  <Slideshow>
    { as in get_slideshow }
  </Slideshow>
  ...
</User>
```

Listing 7: Response XML Format for method get_slideshows_by_tag of SlideShare API [Sli10c]

```
<Tag>
  <Name>{ Tag Name }</Name>
  <Count>{ Number of Slideshows }</Count>
  <Slideshow>
    { as in get_slideshow }
  </Slideshow>
  ...
</Tag>
```

Listing 8: Response XML Format for method get_user_contacts of SlideShare API [Sli10c]

```
<Contacts>
  <Contact>
    <Userame>{ Username }</Name>
    <NumSlideshows>{ Number of Slideshows }</Name>
    <NumComments>{ Number of Comments }</Name>
  <Contact>
  ...
</Contacts>
```

A.4 RESULTS OF THE SCRIBD API METHOD DOCS.SEARCH

See Scribd API method docs.search:
http://www.scribd.com/developers/api?method_name=docs.search

TYPE	METHOD	COMMENT
list	result_set	List of results returned by the search query.
list	result	Contains each individual result
integer	doc_id	Scribd has a unique document ID.
string	secret_password	Secret password for private documents.
string	access_key	Embed a document on an external site.
string	title	–
string	description	–
string	tags	Comma separated list of tags.
string	license	See Creative Commons License.
URL	thumbnail_url	Link to a JPG.
integer	page_count	The number of pages in the document.
list	download_formats	List of downloadable extensions.
float	price	The price at which it is sold.
integer	reads	Number of reads.
string	uploaded_by	User who uploaded this document.
datetime	when_uploaded	The date of upload.
datetime	when_updated	The date of last update.
boolean	available_on_api	Indicates availability API.

Table 8: Results of the method docs.search

A.5 CRAWLERMANAGER WEBSERVICES AND PARAMETERS

Listing 9: Webservice for adding general jobs

```
addJob?URI=uriToRessource
&starttime=startTimeInMilliSeconds
&depthExternLinks=searchDepthInternal
&depthInternalLinks=searchDepthExternal
&doobserve=observeUri
&rescan=rescanResources

Return value: CrawlerJob
```

Listing 10: Webservice for adding actor jobs

```
addActJob?actor=nameOfActor
&starttime=startTimeInMilliSeconds
&depthExternLinks=searchDepthInternal
&depthInternalLinks=searchDepthExternal

Return value: CrawlerJob
```

Listing 11: Webservice for adding artefact jobs

```
addArtJob?actor=urlOfArtefact
&starttime=startTimeInMilliSeconds
&depthExternLinks=searchDepthInternal,
&depthInternalLinks=searchDepthExternal

Return value: CrawlerJob
```

Listing 12: Webservice for adding tag jobs

```
addTagJob?actor=nameOfTag
&starttime=startTimeInMilliSeconds
&depthExternLinks=searchDepthInternal
&depthInternalLinks=searchDepthExternal

Return value: CrawlerJob
```

Listing 13: Webservice for receiving job information

```
getJob?uri=uriOfJob

Return value: CrawlerJob
```

Listing 14: Webservice for listing jobs

```
listJobs

Return value: CrawlerJob[] (Array of CrawlerJobs)
```

Listing 15: Webservice for removing jobs

```
removeJob?uri=uriOfResource

Return value: void (Nothing returned)
```

BIBLIOGRAPHY

[Dav10] Ian Davis. RELATIONSHIP. http://vocab.org/relationship/, accessed 11 July 2010, 2010.

[Dub10] Dublin Core Metadata Initiative. Dublin Core. http://dublincore.org/, accessed 11 July 2010, 2010.

[FOA10] FOAF. The friend of a friend (foaf) project. http://www.foaf-project.org/, accessed 10 July 2010, 2010.

[IET10] IETF, Internet Engineering Task Force. The application/json Media Type for JavaScript Object Notation (JSON). http://www.ietf.org/rfc/rfc4627, accessed 13 July 2010, 2010.

[Jen10] Jena team. Jena – A Semantic Web Framework for Java. http://openjena.org/, accessed 13 July 2010, 2010.

[KL09] Barbara Kieslinger and Stefanie N. Lindstaedt. Science 2.0 Practices in the Field of Technology Enhanced Learning. In *EC-TEL 2009*, 2009.

[Mik07] Peter Mika. *Social Networks and the Semantic Web*. Springer Science+Business Media, New York, 2007.

[Mit69] J Clide Mitchell. *Social networks in urban situations*. University press, University of Manchester, Manchester, 1969.

[Moi10] Matthias Moi. Anwendung und Evaluierung von Artefact-Actor-Networks im Kontext von Wikis – Entwicklung von MediaWiki-Erweiterungen zur Extraktion und Visualisierung semantischer änlichkeiten in MediaWikis, 2010. Diploma thesis, University of Paderborn, Didactics of Informatics, 2010.

[Ora10] Oracle Corporation. Java. http://java.sun.com/, accessed 13 July 2010, 2010.

[OSG10] OSGi Alliance. OSGi Technology. http://www.osgi.org/About/Technology, accessed 14 July 2010, 2010.

[RMV09] Wolfgang Reinhardt, Matthias Moi, and Tobias Varlemann. Artefact-Actor-Networks as tie between social networks and artefact networks. In *Proceedings of CollaborateCom 2009*, 2009.

[Scr10a] Scribd Inc. About Scribd. http://www.scribd.com/, accessed 22 July 2010, 2010.

[Scr10b] Scribd Inc. Scribd. http://www.scribd.com/, accessed 10 July 2010, 2010.

[Scr10c] Scribd Inc. Scribd API Overview. http://www.scribd.com/developers/api, accessed 23 July 2010, 2010.

[Shn08] Ben Shneiderman. Science 2.0. *SCIENCE*, 319:1349–1350, March 2008.

[SIO10] SIOC-Project. SIOC, Semantically-Interlinked Online Communities. http://sioc-project.org/, accessed 11 July 2010, 2010.

[Sli07] SlideShare Inc. Happy Birthday SlideShare. http://blog.slideshare.net/2007/10/05/happy-birthday-slideshare/, accessed 22 July 2010, 2007.

[Sli10a] SlideShare Inc. About us. http://www.slideshare.net/about, accessed 22 July 2010, 2010.

[Sli10b] SlideShare Inc. Slideshare. http://www.slideshare.net/, accessed 09 July 2010, 2010.

[Sli10c] SlideShare Inc. Slideshare API Documentation. http://www.slideshare.net/developers/documentation, accessed 09 July 2010, 2010.

[Twi10] Twitter Inc. Twitter. http://twitter.com/, accessed 27 July 2010, 2010.

[UWS+10] Thomas Ullmann, Fridolin Wild, Peter Scott, Eric Duval, Bram Vandenputte, Gonzalo Parra, Wolfgang Reinhardt, Nina Heinze, Peter Kraker, Angela Fessl, Stefanie Lindstaedt, Til Nagel, and Denis Gillet. A Science 2.0 Infrastructure for Technology-Enhanced Learning. In *EC-TEL 2010*, 2010.

[Var10] Tobias Holger Varlemann. Konzeption und Entwicklung einer Architektur zur semantischen Analyse, Speicherung und Bereitstellung von Daten aus Blogs und Microblogs in Artefact-Actor-Networks, 2010. Bachelor's thesis, University of Paderborn, Didactics of Informatics, 2010.

[W3C10a] W3C, World Wide Web Consortium. Owl Web Ontology Language. http://www.w3.org/TR/owl-features/, accessed 13 July 2010, 2010.

[W3C10b] W3C, World Wide Web Consortium. OWL Web Ontology Language – Use Cases and Requirements: What is an ontology? http://www.w3.org/TR/webont-req/#onto-def, accessed 14 July 2010, 2010.

[W3C10c] W3C, World Wide Web Consortium. RDF Primer. http://www.w3.org/TR/rdf-primer/, accessed 13 July 2010, 2010.

[W3C10d] W3C, World Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, accessed 13 July 2010, 2010.

[W3C10e] W3C, World Wide Web Consortium. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, accessed 13 July 2010, 2010.

[Wal08] M. Mitchell Waldrop. Science 2.0. *Scientific American*, 298:68–73, May 2008.

[WHKL08] Gerd Wütherich, Nils Hartmann, Bernd Kolb, and Matthias Lübken. *Die OSGi Service Platform*. dpunkt.verlag GmbH, Heidelberg, 2008.

[Wik10] Wikimedia Foundation Inc. MediaWiki. http://www.mediawiki.org/, accessed 16 July 2010, 2010.

[Yah10a] Yahoo! Inc. Delicious. http://delicious.com/, accessed 05 July 2010, 2010.

[Yah10b] Yahoo! Inc. Delicious Feeds. http://delicious.com/help/feeds, accessed 05 July 2010, 2010.

[Yah10c] Yahoo! Inc. Delicious is 5! http://blog.delicious.com/blog/2008/11/delicious-is-5.html, accessed 22 July 2010, 2010.

[Yah10d] Yahoo! Inc. Delicious Tools. http://delicious.com/help/tools/, accessed 05 July 2010, 2010.

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Paderborn, July 27, 2010

Adrian Wilke