Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften (Dr. rer. pol.)
von der Fakultät für Wirtschaftswissenschaften
der Universität Fridericiana zu Karlsruhe
vorgelegte Dissertation von
Dipl.-Inform.Wirt Holger Michael Lewen.

Facilitating Ontology Reuse Using User-Based Ontology Evaluation

Holger Michael Lewen

Tag der mündlichen Prüfung: 23.11.2010
Referent: Prof. Dr. Rudi Studer
Korreferent: Prof. Dr. Christof Weinhardt
2010 Karlsruhe

This document was created on November 26, 2010

Meiner Familie

Acknowledgements

This work would not have been possible without the support I got from many sides.

First of all I would like to thank my adviser, Prof. Dr. Rudi Studer, for providing this truly unique working environment and all the support over the years. I can hardly imagine a better place to be as a PhD student. Without the freedom to pursue my research interests and the funding to go to conferences, this thesis could not have been written.

Second I would like to thank my co-referent Prof. Dr. Christof Weinhardt for taking the time to serve on my examination committee. My profound gratitude also goes to Prof. Dr. Andreas Oberweis, who served as examiner on the committee, and Prof. Dr. Karl-Heinz Waldmann, who served as chair of the examination committee.

A special thanks goes to my two supervisors Dr. Peter Haase and Dr. Denny Vrandečić who advised me in my research over the last 5 years, and also proof-read part of my thesis.

I was blessed enough to work with amazing colleagues over the last years. Thanks to each and every one of you. Special thanks go to Dr. Markus Krötzsch and Dr. Sebastian Rudolph for helping me formalize the algorithms and check the more mathematical parts of my thesis. Moreover I want to thank Dr. Elena Simperl, Dr. Andreas Harth, Dr. Thanh Tran, Dr. Barry Norton, Dr. Sudhir Agarwal and Dr. Natasha F. Noy for their valuable feedback on my conference submissions.

Moreover I want to thank the members of the NeOn Project, with whom I had the pleasure of working. Especially Dr. Mathieu d'Aquin, with whom I had the pleasure of co-authoring several papers. Thanks to Prof. Dr. Enrico Motta for directing the NeOn project, which funded me for the longest part of my AIFB time, and for providing valuable advise, both scientifically, but also non-scientifically.

Last, and most importantly, I thank my wonderful wife Bessy for the loving support and the patience she had with me during the long nights of writing the thesis. A big thanks to my parents, grand-parents and brother for their support over the years.

Without all those people, this work would not have been possible. Each and every one deserves my profound gratitude.

Karlsruhe, November 2010.
Holger Lewen

Abstract

Ontology reuse saves costs and improves interoperability between ontologies. Knowing which ontology to reuse is difficult without having a quality assessment. We provide a framework that enables this quality assessment in the form of user reviews of ontologies, which are ranked based on inter-user trust. Open Rating Systems allow to collect user reviews, ratings, and information on trust between users, and to exploit this information for computing a personalized ranking of ontologies. However, in the traditional Open Rating System model, objects can only be rated as a whole, and other users can only be trusted globally. Since these limitations hinder the use of an Open Rating System for applications like ontology rating, we develop an extension, our Topic-Specific Trust Open Rating System. Our system features topic-specific trust and multi-faceted ratings. In a simulation, we show that our Topic-Specific Open Rating System provides a better result than the Open Rating System. In a user study, we show that having user ratings and result ranking based on our system significantly facilitates ontology selection for the end user, compared to the state of the art ontology search engines.

Short Table of Contents

Part I

Foundations

Chapter 1

Introduction and Overview

1.1 Introduction

Ontology reuse is a good practice in ontological engineering, because it reduces costs and development effort (Simperl *et al.*, 2009). Ontology engineers save time when they reuse existing ontological content—entire ontologies or parts thereof. In addition, the use of the same URIs for the same entities across ontologies increases interoperability and interlinkage between ontologies, since no ontology matching is needed between the vocabularies of the ontologies.

A typical problem users face today when they try to reuse existing ontological content is the selection of the most appropriate ontology or ontology axioms for their task. This is because the assessment of ontologies is a time-consuming and nontrivial task. Furthermore, there is, so far, no automatic evaluation technique that can judge the quality of an ontology the way a human expert can. The ability to rely on the experience of other users can lessen the assessment effort considerably.

In this thesis we investigate how user-based ontology evaluation in the form of reviews and ratings can be used to facilitate ontology reuse for the end user. Open Rating Systems provide a way to compute personalized ratings of objects based on user ratings and information on inter-user trust. In the thesis we analyze the shortcomings of the current Open Rating System model and provide our solution, the Topic-Specific Trust Open Rating System. We test our system both in a simulation and a user study. To facilitate ontology reuse for the end user, we furthermore integrate our solution into an ontology engineering environment, to provide tool-support for the complete ontology reuse process.

1.2 Overview

We start the thesis with an introduction to ontology engineering with an emphasis on the ontology life-cycle and ontology reuse in Chapter 2. Also related work on trust and Open Rating Systems is presented in this chapter.

In Part II, we introduce the conceptual foundation of this thesis, our Topic-Specific Trust Open Rating System (TS-ORS). We start with introducing our formal model in Chapter 3. In Chapter 4 we then present the algorithms needed for computing review rankings and overall ratings for objects in the TS-ORS. Chapter 5 explains how the TS-ORS can be adapted for ontology reuse, and provides an example for the computations performed within the TS-ORS. Chapter 6 discusses possibilities to deal with ratings on evolving objects, or evolving ratings, while Chapter 7 covers the initialization of the TS-ORS in other applications.

In Part III we cover the implementation of our system in Chapter 8, and its integration into our ontology hosting environment and repository Cupboard in Chapter 9. The integration of the Cupboard system with the ontology engineering environment NeOn Toolkit can be found in Chapter 10, while Chapter 11 contains information on data exchange between ontology repositories.

Three different evaluations are presented in Part IV. We start with a complexity analysis and system benchmark in Chapter 12, followed by an agent simulation comparing the Open Rating System with our Topic-Specific Trust Open Rating System in Chapter 13. We conclude the evaluations with a user study comparing our system against state of the art in Chapter 14.

The thesis concludes in Part V with a positioning against related work in Chapter 15, the conclusions in Chapter 16 Section 16.1, and open questions in Chapter 16 Section 16.2.

Chapter 2

Definitions

In this chapter we first define the term ontology in Section 2.1. We then present the ontology lifecycle in Section 2.2, with a special emphasis on ontology evaluation (covered in Section 2.3) and ontology reuse (covered in Section 2.4). Then the concept of trust is introduced alongside trust propagation techniques in Section 2.5. The chapter concludes with an introduction to Open Rating Systems in Section 2.6.

2.1 Ontology

Studer, Benjamins and Fensel define an ontology based on (Gruber, 1993; Borst, 1997) as follows:

> "An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of concepts of some phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group." (Studer *et al.*, 1998)

They stress that ontologies should be machine-readable and the consensual knowledge of a group.

Even though ideally the knowledge covered by an ontology is language independent, it needs to be represented in an ontology language. While a number of different ontology languages have been developed (for a roadmap of such languages see (Corcho and Gómez-Pérez, 2000)), in this thesis we focus on ontologies represented in one of the currently relevant ontology languages RDF(S)[1] (Brickley and Guha, 10 February 2004), OWL[2] (McGuinness and van Harmelen, 10 February 2004), or OWL 2 (W3C OWL Working Group, 27 October 2009). RDF(S) and OWL are the most important languages for the Semantic Web (Berners-Lee *et al.*, 2001), or Web of Data, as it is

[1] `http://www.w3.org/RDF/`, last checked on 24.11.2010
[2] `http://www.w3.org/2004/OWL/`, last checked on 24.11.2010

now also called. The general idea of the Semantic Web (Berners-Lee *et al.*, 2001) is to extend the current World Wide Web (WWW) so that information can be given well-defined meaning, in a way that is machine-processable.

Application areas of ontologies

While classically ontologies were used to model the world, this is only one of their use cases today. There are still ontologies which capture the consensual knowledge of a particular domain or the world in general. They are usually referred to as domain ontology in case the ontology is limited to a specific domain, and upper level ontology, if a more general view of the world is modeled. Prominent examples of upper level ontologies are SUMO (Niles and Pease, 2001), Cyc (Lenat, 1995), DOLCE (Gangemi *et al.*, 2002), and BFO (Smith and Grenon, 2004).

The main use case for ontologies nowadays is data/information integration (Blake and Bult, 2006; Calvanese *et al.*, 2007b; He and Ling, 2006; Ceravolo *et al.*, 2008; Cruz and Xiao, 2009). Especially with the rise of the World Wide Web and the idea of the Semantic Web the ability to integrate data from heterogeneous sources is one of the main benefits ontologies provide. In the biomedical domain, ontologies are mostly used for data annotation (Groth *et al.*, 2008; Jonquet *et al.*, 2008; Chebotko *et al.*, 2009; Barrell *et al.*, 2009). The most prominent example is the Gene Ontology (Ashburner *et al.*, 2000), which is used to annotate gene and gene products. But ontologies can also support intelligent applications (Masuoka *et al.*, 2003; Fu *et al.*, 2005; Lee and Wang, 2007; Tan *et al.*, 2008; Ayala, 2009). We propose an exemplary architecture for such a semantic application in (Tran *et al.*, 2007c), using a personalized knowledge portal as an example. Another use case is semantic search, where ontologies can be used to provide structure and background knowledge (Tran *et al.*, 2007a; Bloehdorn *et al.*, 2008) to better interpret the user's information need, and improve precision.

In the next section, we present the different stages of the ontology lifecycle, and show how they can be supported by an ontology engineering tool.

2.2 Ontology Lifecycle

Parts of this section are based on (Tran *et al.*, 2007b). The term ontology lifecycle was introduced in methodologies for ontology engineering (Gómez-Pérez *et al.*, 2003). The idea is that during its life, an ontology goes through different phases, ranging from a requirement analysis to the evaluation of the created ontology. While early ontology lifecycle models (Uschold and Gruninger, 1996; Fernandez-Lopez *et al.*, 1997; Staab *et al.*, 2001) focus only on the ontology engineering part of the ontology lifecycle, in (Tran *et al.*, 2007b) we propose an ontology lifecycle model that also takes ontology usage into account. Figure 2.1 depicts our ontology lifecycle. In our lifecycle, we distinguish

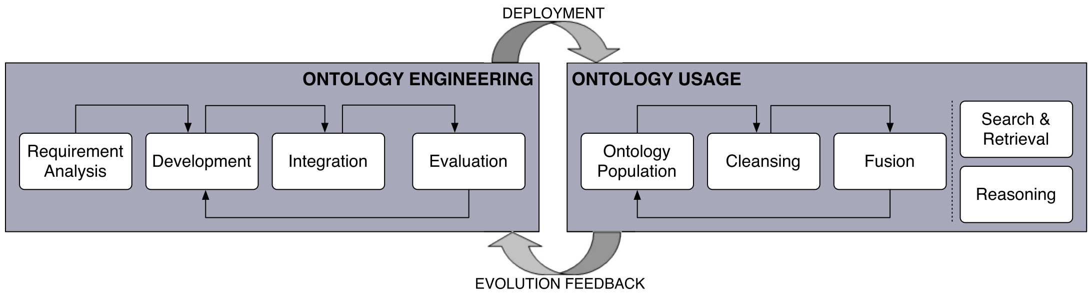


Figure 2.1: Our Ontology Lifecycle Model

between ontology engineering (the act of building an ontology) and ontology usage (the employment of a built ontology within an application).

2.2.1 Ontology Engineering

Even though individual methodologies vary, most agree on the main lifecycle activities, which are *requirement analysis*, *development*, *evaluation*, and *maintenance* , in addition to orthogonal activities. In our lifecycle model, we also include these agreed-upon activities, and include maintenance in the context of usage-related activities.

Requirement Analysis:

In the requirement analysis step, domain experts and ontology engineers define and document modeling requirements based on the intended usage scenario of the ontology. Depending on whether the ontology should be used for retrieval or reasoning tasks, the modeling has to be adapted.

Development:

Because methodologies vary on what to do in the development step, we present an aggregation on the different proposals. An important step before starting the actual development is finding and assessing reusable ontological content. Reuse is a common best practice that reduces costs and development effort (Simperl *et al.*, 2009; Simperl *et al.*, 2006; Uschold *et al.*, 1998). We analyze ontology reuse in more depth in Section 2.4. After reusable ontologies are found and selected, they have to be adapted according to the results of the requirement analysis phase. Modifications may include backward and forward engineering of the reusable ontologies. Backward engineering means, for example, restructuring or modifying the ontologies, while forward engineering means, for instance, extending the ontologies. Depending on the results of the requirement analysis phase, the ontologies are then translated to the target representation language, and their degree of axiomatization is tweaked according to the expressivity and scalability requirements of the intended application.

Collaboration between ontology engineers is an important aspect of ontology development. Recent work focuses on the definition of workflows for ontology engineering (Tudorache *et al.*, 2008; Sebastian *et al.*, 2008; Palma *et al.*, 2008) that are also supported in tools. Apart from these approaches, (Gangemi *et al.*, 2007) introduced an ontology-based collaboration method. The DILIGENT methodology focuses on argumentation-based ontology engineering (Tempich *et al.*, 2007). The idea here is that reasons for modeling decisions are formalized and discussions can follow a given structure.

Integration:

Following the componentization encountered in software engineering, recent ontology development approaches advocate the construction of modularized ontologies, rather than one monolithic ontology (Wang *et al.*, 2007; Pathak *et al.*, 2009). Via import declarations and alignment rules, these modules then have to be integrated. In this step, the resulting linked modules are also integrated with external ontologies if this is required for interoperability with external applications.

Evaluation:

The result of the integration phase has to be evaluated to find errors or deviations from the requirements of the requirement analysis phase. Evaluation ranges from simple checks like finding inconsistencies in the ontology to a more difficult human based assessment with respect to the defined requirements. We explore ontology evaluation in more depth in Section 2.3. In case deficiencies are detected in this phase, they are addressed by moving back in the lifecycle, for example to the development phase.

2.2.2 Ontology Usage

Ontologies are engineered to serve a purpose. That means that after they are engineered, they are used, for example in applications. In our lifecycle model we have grouped the activities performed with an ontology after its creation under ontology usage. In comparison to prior lifecycle models which were mainly static, our model focuses on dynamic aspects. Normally, when an ontology meets the requirements, it is deployed and the lifecycle continues with ontology evolution or maintenance. In the evolution phase, when new requirements arise, they are fed back into the loop, which leads to an update release which is then redeployed. The activities involved in actual usage of ontologies are however not covered by early lifecycle models. We show how based on the presented activities the lifecycle can be dynamic, that means feeding back into the engineering stage.

Search and Retrieval and Reasoning:

One common use case for ontologies is information access in applications, for example via search and retrieval functionality. Ontology Information Systems (OIS) typically involve reasoners to infer implicit knowledge. The schema provided by the ontology can be combined with instance data to support advanced retrieval inside the application. Schema knowledge can be exploited for query enhancement (refinement, expansion), and A-Box reasoning to also retrieve inferred knowledge.

Ontology Population:

In order to populate the knowledge base, either information is loaded automatically or has to be collected from the user, for example by means of input forms. In order to reduce the overhead imposed to the user when all instance data is created manually (semi)-automatic population of the knowledge base can be employed. In this step of the lifecycle, instances are not only added, but also manipulated and deleted.

Cleansing and Fusion:

Knowledge extracted automatically is often noisy and can therefore not be assumed to have the necessary quality to be employed in an application directly. Common cleansing tasks include the identification and merging of conceptually identical instances as well as a fusion on a higher level, for example merging redundant statements.

Both population and fusion steps can lead to inconsistencies which have to be identified and resolved. Inconsistencies can prevent an application to produce sensible output at runtime. When inconsistencies are identified, for example by a reasoner or by a user, the inconsistencies are fed back to the debugging and the development-phase of the ontology lifecycle. That means that ontology evolution (the loop from usage back to engineering activities) is not only depending on changes in requirements but is also a necessity for the runtime usage of ontologies.

2.2.3 Application Support for the Ontology Lifecycle

In order to facilitate the usage of methodologies for the ontology engineers, it is important to not only provide guidelines, but also to support the lifecycle in the ontology engineering environment. That means, that there exists tool support for certain activities performed at certain steps of the ontology lifecycle. In the context of the NeOn Project[3] we have developed several so-called plugins, that support the ontology engi-

[3]http://www.neon-project.org, last checked on 24.11.2010

neer in different stages of the ontology lifecycle. Figure 2.2 provides a mapping from our plugins to the lifecycle activities. The NeOn Toolkit and plugin descriptions can be found online.[4]

2.3 Ontology Evaluation

As mentioned in Section 2.2, ontology evaluation is a crucial part of the ontology reuse process and also of the ontology lifecycle in general. Without being able to assess the quality of an ontology, it is hard to select the best ontology from a set of potential candidate ontologies.

Ontology evaluation has been approached from many different angles, ranging from (semi-)automatic methods to fully manual methods (Brank *et al.*, 2005).

One approach is employing an ontology in an application and evaluating the results (Porzel and Malaka, 2004). The idea here is that the quality of the ontology is directly correlated with the outcome of the application. In case the outcome of the application can then be evaluated, for example against a gold standard, the quality of the ontology for that task is evaluated as well. This approach is called task-based evaluation.

Data driven ontology evaluation is presented in (Brewster *et al.*, 2004). Here the ontology is compared with a source of data, like a collection of documents, that represents the domain to be covered by the ontology.

While the former approaches can be performed (semi-)automatically, another line of research covers human evaluation of ontologies. Here, certain criteria or requirements are used to assess how well the ontology performs based on these criteria (Tello and Gómez-Pérez, 2004). A plethora of methods and guidelines on how to evaluate certain parts of ontologies exist. Some focus on the logical consistency or satisfiability (Parsia *et al.*, 2005; Gómez-Pérez, 2004). Others check whether a term in an ontology is properly grounded, meaning whether it is understood by the ontology users (Jakulin and Mladenić, 2005).

Apart from this coarse categorization, a more fine-grained evaluation framework can be found in the literature. Gangemi and colleagues (Gangemi *et al.*, 2006) developed a meta-ontology to provide a framework in which existing ontology evaluation techniques are grouped. They identify three main dimensions among which ontologies can be measured: Structural (having a focus on syntax and formal semantics), functional (having a focus on the intended use and function in a context), and usability-profiling (focusing on annotations and pragmatics).

In summary, while some aspects of an ontology can be evaluated automatically (like logical consistency), most evaluation tasks still rely on human involvement. Further-

[4]`http://neon-toolkit.org`, last checked on 24.11.2010

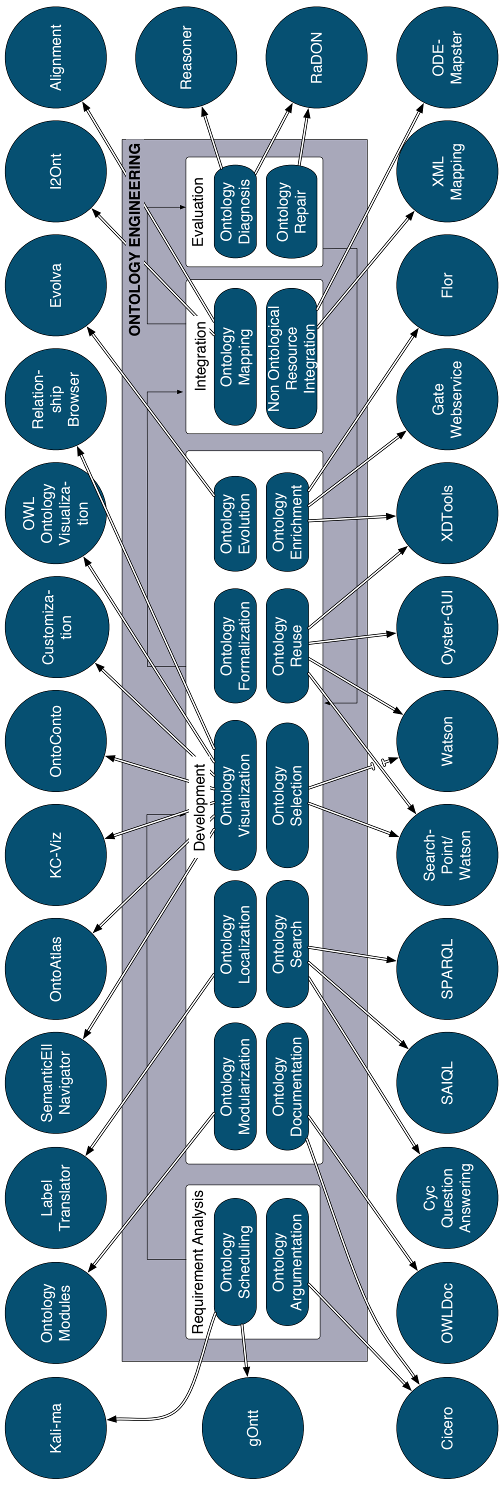


Figure 2.2: NeOn plugins providing support for the ontology lifecycle model

more one should note that an evaluation of the same ontology can vary depending on its planned task, and on the person evaluating the ontology.

2.4 Ontology Reuse

In software engineering, reuse is common to save development effort and improve the quality of the resulting code by reusing high quality software (McIlroy, 1976; Krueger, 1992). The reuse of preexisting libraries that offer certain functions has become best practice (Pressman and Ince, 1987), and basic functionality is normally not reimplemented without a compelling reason.

In the context of ontology engineering, the benefits of ontology reuse have been stressed in several works (Simperl *et al.*, 2009; Simperl *et al.*, 2006; Uschold *et al.*, 1998). Yet, reuse happens very scarcely, which hinders the uptake of the semantic web (Simperl, 2009). We claim that this is partly due to the high level of human effort required in the different ontology reuse steps.

In general, there is a trade-off between the reusability of an ontology and the specificity towards a task or application. Upper level ontologies, which try to provide a layer on top of all domains, aim to be highly reusable by abstracting from specific problems or applications. An ontology employed and developed for a specific domain will most of the times provide more domain knowledge than an upper level ontology, but might require more reengineering during the reuse process.

2.4.1 Ontology Reuse Process

A common ontology reuse process (Simperl, 2009) consists of the following three steps:

- *Ontology Discovery:* In this step candidate ontologies that are suitable to be reused are searched. Currently the most common way to find available ontologies is using conventional search engines,[5] Semantic Web search engines (e.g., (Ding *et al.*, 2004a; d'Aquin *et al.*, 2008d)), or ontology libraries (e.g. (Noy *et al.*, 2009; Viljanen *et al.*, 2009)). Also standard bodies can be a source of information. In case enough domain knowledge can be found, an ontology can be constructed semi-automatically employing ontology learning techniques.

- *Ontology Selection:* In this step the quality and applicability of the candidate ontologies has to be assessed. Based on this assessment (the evaluation of the ontologies, compare also Section 2.3), the best ontology or ontologies are selected for reuse. This step is the most complicated one, since without reliable automatic evaluation techniques, the user has no help in the process of selecting the best

[5]For example `http://www.google.com`, last checked on 24.11.2010, or `http://www.bing.com/`, last checked on 24.11.2010

ontology. Manually assessing all candidate ontologies is extremely tedious, if not impossible. In certain cases, an agreed upon ontology to reuse is available, and there is no real selection due to the lack of alternatives. This is often encountered in the biomedical domain, where, for example, the Gene Ontology (Ashburner *et al.*, 2000) has become the de facto standard to annotate gene and gene product attributes. Also for other biomedical fields, groups are developing upper level ontologies they want to push as standards for these fields (Smith *et al.*, 2006). An exemplary collection of these upper level ontologies can be found in the NCBO Bioportal.[6]

- *Ontology Integration:* After suitable ontologies have been selected, they have to be integrated with the ontology being developed. Often, this requires re-engineering of the ontology that should be reused. The matching concepts in the new ontology and the reused ontology have to be identified to ensure a proper integration. In case only parts of an ontology are needed, these can be extracted, for example using state of the art modularization techniques (Pathak *et al.*, 2009). While ontology integration can be a cumbersome task, the main problems still lie within the selection step of the ontology reuse process.

Tools and Applications Supporting the Ontology Reuse Process

Currently there are some tools and applications available to support the user during the ontology reuse process. Based on which step of the ontology reuse process the user is currently in, varying help is provided.

Ontology Discovery: The first step, ontology discovery, is currently supported by a plethora of search engines like Google or Bing, semantic web search engines, ontology libraries, and ontology repositories.

One of the first systems that indexed and exposed ontologies was the Ontokhoj system (Patel *et al.*, 2003). It is currently not available anymore. Shortly after, Swoogle[7] (Ding *et al.*, 2004a) was developed at the ebiquity group at UMBC university. In its current index, it has around 1.8 million error-free Semantic Documents.[8] Ontosearch 2 (Thomas *et al.*, 2007) allows both keyword and SPARQL queries against indexed ontologies. Sindice[9] (Tummarello *et al.*, 2007) indexes not only ontologies, but all semantic web content it can find. It mainly relies on Linked Open Data (Bizer *et al.*, 2008) and thus interlinked RDF data. VisiNav[10] (Harth *et al.*, 2009) follows

[6] `http://bioportal.bioontology.org/`, last checked on 24.11.2010
[7] `http://swoogle.umbc.edu/`, last checked on 24.11.2010
[8] `http://swoogle.umbc.edu/index.php?option=com_swoogle_stats`, last checked on 24.11.2010
[9] `http://sindice.com/`, last checked on 24.11.2010
[10] `http://visinav.deri.org/`, last checked on 24.11.2010

a similar approach as Sindice, also relying on Linked Open Data. Falcons[11] (Cheng *et al.*, 2008) focuses on searching and browsing entities on the Semantic Web. Watson[12] (d'Aquin *et al.*, 2008d) provides various interfaces to access its indexed semantic data. It currently indexes at least 27,000 ontologies.

Ontology libraries (Ding and Fensel, 2001) and repositories (Hartmann *et al.*, 2009) are still an important place to look for ontologies, especially because many of the ontologies stored there are of high quality, and often enough there is metadata available to enable a better search. Currently, NCBO's BioPortal[13] (Noy *et al.*, 2009) stands out for biomedical ontologies. The Protégé Ontology Library[14] provides a list of ontologies without a search functionality. The Tones ontology repository[15] provides around 200 ontologies. Within the Tones repository, ontologies can be filtered by DL expressivity and by the number of ontological axioms they contain. Other repositories can be considered legacy since they contain ontologies in outdated ontology languages not supported by current tools, or are no longer accessible on the Web, like Onthology (Hartmann *et al.*, 2005; Hartmann, 2006) or KnowledgeZone (Supekar *et al.*, 2007).

Most of the libraries and repositories have been indexed by one of the aforementioned ontology search engines, so that search engines might still be the preferred point of entry for the regular user.

Ontology Selection: In the end, the ontology developer has to decide which ontology to select from a set of potential candidate ontologies. In theory, this would require the developer to assess each candidate ontology individually, which is often unfeasible due to time constraints. Most ontology search engines thus employ a ranking mechanism, which should assist the developer in the selection process by ranking ontologies that are more relevant higher in the result list.

The Ontokhoj system (Patel *et al.*, 2003) featured ontology ranking based on connections between the ontologies. The authors used a derivation of the PageRank algorithm (Page *et al.*, 1998) for ranking ontologies, based on the idea that analyzing their link graph can derive the importance and quality of ontologies. The creators of SWOOGLE (Ding *et al.*, 2004a) extended this idea by taking more connection types into account, and indexing Semantic Web Documents automatically at larger scale. Ontoselect (Buitelaar *et al.*, 2004) combines a repository with ranking mechanisms relying on measures. Employed measures are based on coverage and structure of the ontologies in the repository, and their interlinkage.

[11] `http://ws.nju.edu.cn/falcons/`, last checked on 24.11.2010
[12] `http://watson.kmi.open.ac.uk/`, last checked on 24.11.2010
[13] `http://bioportal.bioontology.org/`, last checked on 24.11.2010
[14] `http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library`, last checked on 24.11.2010
[15] `http://owl.cs.manchester.ac.uk/repository/browser`, last checked on 24.11.2010

There is, however, one problem faced by all the approaches that rely on links between ontologies to derive their importance or quality, and use this information for ontology ranking: The majority of ontologies outside the Linked Open Data initiative are only sparsely interlinked and the popularity of an ontology does not necessarily correlate with its quality (Alani *et al.*, 2006). Therefore, instead of relying on PageRank derivations, Alani et al. developed methods to rank ontologies found by search engines according to a combination of several measures. Their approach, ActiveRank (Alani *et al.*, 2006), is based on measures like: how well concepts in the ontology match the search terms, how well the searched concepts are represented in the ontology (number of subclasses), or how central they are to the ontology. Similar measures are used in the Ontoselect repository (Buitelaar *et al.*, 2004). Ontosearch2 (Thomas *et al.*, 2007) allows both keyword and SPARQL queries against indexed ontologies and ranks the results based on the sum of weightings for object/keyword pairings that matched in the ontologies.

Since computing complex measures for all ontologies in a result set is resource-intensive, not all measures can be computed at runtime. This hinders their use in online search engines, unless results for all potential search keywords are cached.

Lucene[16] has become a popular framework for indexing documents and providing search functionality over them. The ranking of results is based on statistical measures normally used in information retrieval, such as TF/IDF (Frakes and Baeza-Yates, 1992), not taking into account the special nature of semantic documents. Since Lucene provides a very scalable indexing mechanism, it is used in large-scale semantic web repositories, such as Watson (d'Aquin *et al.*, 2008d).

Other search engines focusing on the linked data approach, like Sindice (Tummarello *et al.*, 2007), or VisiNav (Harth *et al.*, 2009), combine information such as provenance of the triple (the hostname and external rank of the website hosting the ontology) with TF/IDF-like measure when ranking their results. Falcons (Cheng *et al.*, 2008) uses a combination of TF/IDF-like measures and the popularity (in terms of incoming links) of the ontology.

Another idea is to rely on users to evaluate ontologies and provide ratings on their quality (Noy *et al.*, 2005). These ratings and inter-user trust can then be used to compute personalized rankings of ontologies that can help the ontology engineer during the selection process (Guha, 2004). We have analyzed this approach, which was also used in the Knowledge Zone repository (Supekar *et al.*, 2007), and have identified limitations (which are described in detail in Section 2.6.4). Our approach towards overcoming identified shortcomings is one of the main contributions of this thesis. We will describe the basic Open Rating System as used in Knowledge Zone in Section 2.6. Our extension, the Topic-Specific Trust Open Rating System (TS-ORS) will be presented in Part II. Because trust and trust propagation are a central aspect of this

[16] `http://lucene.apache.org/`, last checked on 24.11.2010

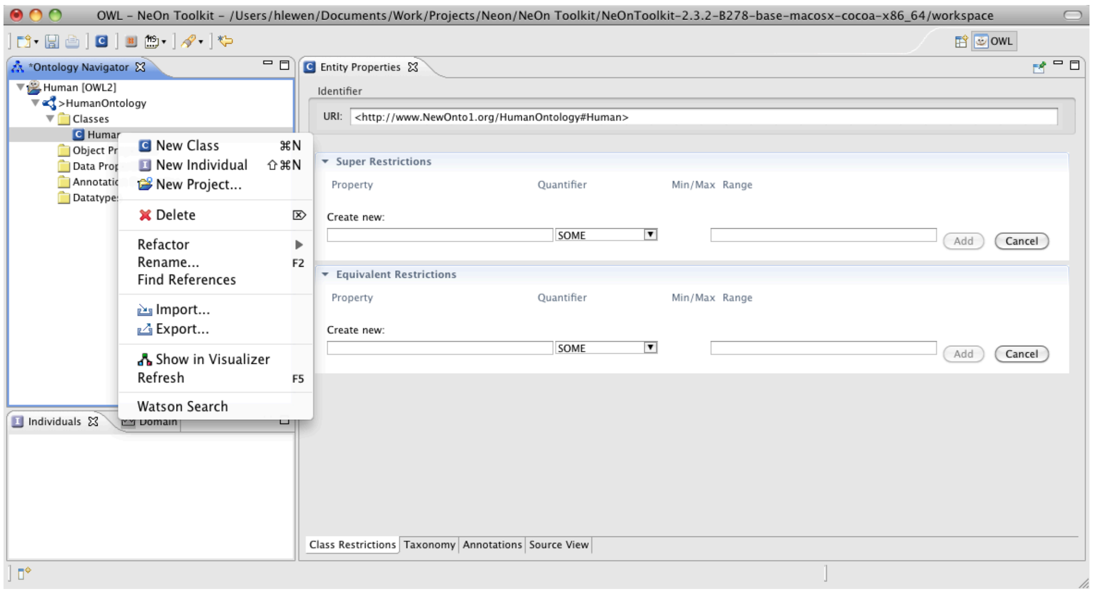
approach, we present an introduction to the field in Section 2.5.

Ontology Integration: After an ontology has been selected for reuse, it has to be integrated into the ontology that is constructed. Depending on its intended usage, it is necessary to integrate either the complete ontology, ontology modules (i.e., parts of the ontology), or even only single axioms. In case an ontology is not integrated in its entirety, a number of modularization techniques exist (Stuckenschmidt *et al.*, 2009; Pathak *et al.*, 2009) to dissect the ontology. Many modularization techniques are also implemented within ontology engineering environments. The ontology editor SWOOP, for example, has logic-based modularization techniques built in (Grau *et al.*, 2009a; Grau *et al.*, 2009b; Grau *et al.*, 2007). For the NeOn Toolkit, plugins exist to facilitate ontology integration. The modularization plugin for the NeOn Toolkit (d'Aquin *et al.*, 2008b; d'Aquin *et al.*, 2008a) allows users to extract,[17] partition,[18] and compose[19] ontologies.

In case there are overlapping concepts, these have to be mapped to ensure a proper integration. Ontology Alignment techniques (Shvaiko *et al.*, 2009; Euzenat, 2008; Euzenat *et al.*, 2008) are available to support the user in the task of mapping classes of the developed ontology and the reused ontology. Within the NeOn Toolkit, the Alignment Plugin[20] (Duc *et al.*, 2008) provides tool assistance to the user.

All of these techniques have no direct integration with the steps of ontology discovery and ontology selection. To overcome this problem, the Watson Plugin for the NeOn Toolkit[21] (d'Aquin *et al.*, 2008c) integrates tool support for all three phases into one tool. From within the plugin, users can search Watson by right-clicking an entity in the ontology and triggering the Watson search (see Figure 2.3). The results of the search process will then be displayed within the plugin (see Figure 2.4). Users can browse the list and select the axioms to reuse (see Figure 2.5). By simply clicking an add button, the axiom is added to the ontology (see Figure 2.6). The use of alignment technologies is not necessary, since the axiom always connects to the selected entity in the ontology. If, for example, the class "Human" was used to trigger the search, all results will contain "Human" at some part of the axiom. In case a user clicks the add button, the concept "Human" in the developed ontology, and "Human" in the reused axiom are considered equal. The main limitation of the Watson Plugin is that the result ranking of Watson is based on Lucene, which uses basic statistical measures that do not correlate with the quality of an ontology. The ranking of results within Watson is thus of no big help in the ontology selection process, which is the most time-consuming part of the whole ontology reuse process. We will show how to

[17] `http://neon-toolkit.org/wiki/Ontology_Module_Extraction`, last checked on 24.11.2010
[18] `http://neon-toolkit.org/wiki/Ontology_Module_Partition`, last checked on 24.11.2010
[19] `http://neon-toolkit.org/wiki/Ontology_Module_Composition`, last checked on 24.11.2010
[20] `http://neon-toolkit.org/wiki/2.3.1/Alignment`, last checked on 24.11.2010
[21] `http://watson.kmi.open.ac.uk/editor_plugins.html`, last checked on 24.11.2010



OWL - NeOn Toolkit - /Users/hlewen/Documents/Work/Projects/Neon/NeOn Toolkit/NeOnToolkit-2.3.2-B278-base-macosx-cocoa-x86_64/workspace

URI: <http://www.NewOnto1.org/HumanOntology#Human>

Figure 2.3: When a user right-clicks an entity, the Watson Search can be triggered from the context menu.

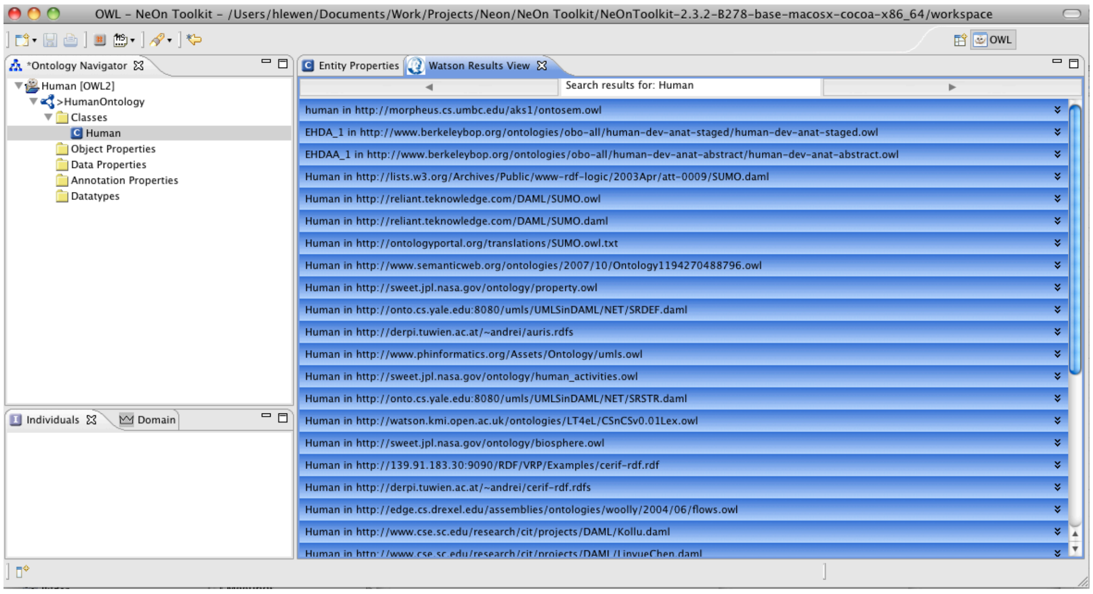
combine the advantages of the Watson Plugin with the ranking results of our TS-ORS system in Chapter 9.

2.5 Trust and Trust Propagation

One of the main requirements for a working Open Rating System (see Section 2.6) is the trust or distrust users place in each other, and the way in which trust and distrust can be propagated. In this section we will first introduce the relevant notions of trust in computer science in Section 2.5.1, and then provide an overview of trust propagation techniques in Section 2.5.2. Parts of this section are based on (Kubias *et al.*, 2007).

2.5.1 Trust in Computer Science

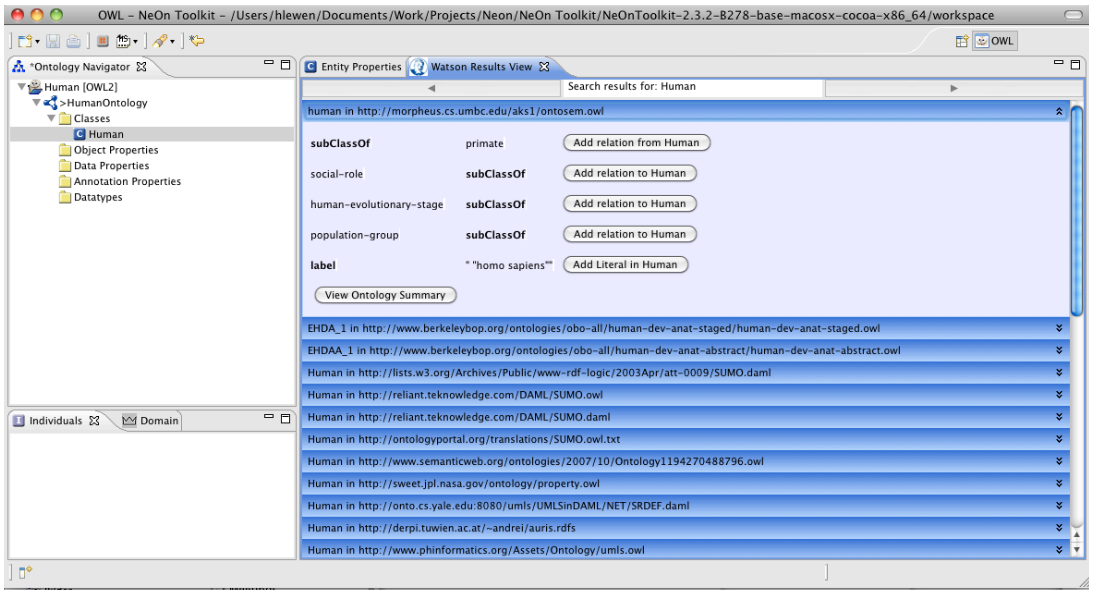
The concept of trust has been explored in many different fields of science, for example sociology (Fukuyama, 1996; Coleman, 1994), psychology (Misztal, 1996), or economics (Berg *et al.*, 1995; Zak *et al.*, 2004). In this thesis we focus on the use of trust in computer science. While an overview on trust languages and the topic of trust



Search results for: Human

```
human in http://morpheus.cs.umbc.edu/aks1/ontosem.owl
EHDA_1 in http://www.berkeleybop.org/ontologies/obo-all/human-dev-anat-staged/human-dev-anat-staged.owl
EHDAA_1 in http://www.berkeleybop.org/ontologies/obo-all/human-dev-anat-abstract/human-dev-anat-abstract.owl
Human in http://lists.w3.org/Archives/Public/www-rdf-logic/2003Apr/att-0009/SUMO.daml
Human in http://reliant.teknowledge.com/DAML/SUMO.owl
Human in http://reliant.teknowledge.com/DAML/SUMO.daml
Human in http://ontologyportal.org/translations/SUMO.owl.txt
Human in http://www.semanticweb.org/ontologies/2007/10/Ontology1194270488796.owl
Human in http://sweet.jpl.nasa.gov/ontology/property.owl
Human in http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRDEF.daml
Human in http://derpi.tuwien.ac.at/~andrei/auris.rdfs
Human in http://www.phinformatics.org/Assets/Ontology/umls.owl
Human in http://sweet.jpl.nasa.gov/ontology/human_activities.owl
Human in http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRSTR.daml
Human in http://watson.kmi.open.ac.uk/ontologies/LT4eL/CSnCSv0.01Lex.owl
Human in http://sweet.jpl.nasa.gov/ontology/biosphere.owl
Human in http://139.91.183.30:9090/RDF/VRP/Examples/cerif-rdf.rdf
Human in http://derpi.tuwien.ac.at/~andrei/cerif-rdf.rdfs
Human in http://edge.cs.drexel.edu/assemblies/ontologies/woolly/2004/06/flows.owl
Human in http://www.cse.sc.edu/research/cit/projects/DAML/Kollu.daml
Human in http://www.cse.sc.edu/research/cit/projects/DAML/LinyueChen.daml
```

Figure 2.4: On the right hand side, the Watson Results View shows up with a result list from Watson. The List shows the URIs of the ontologies.



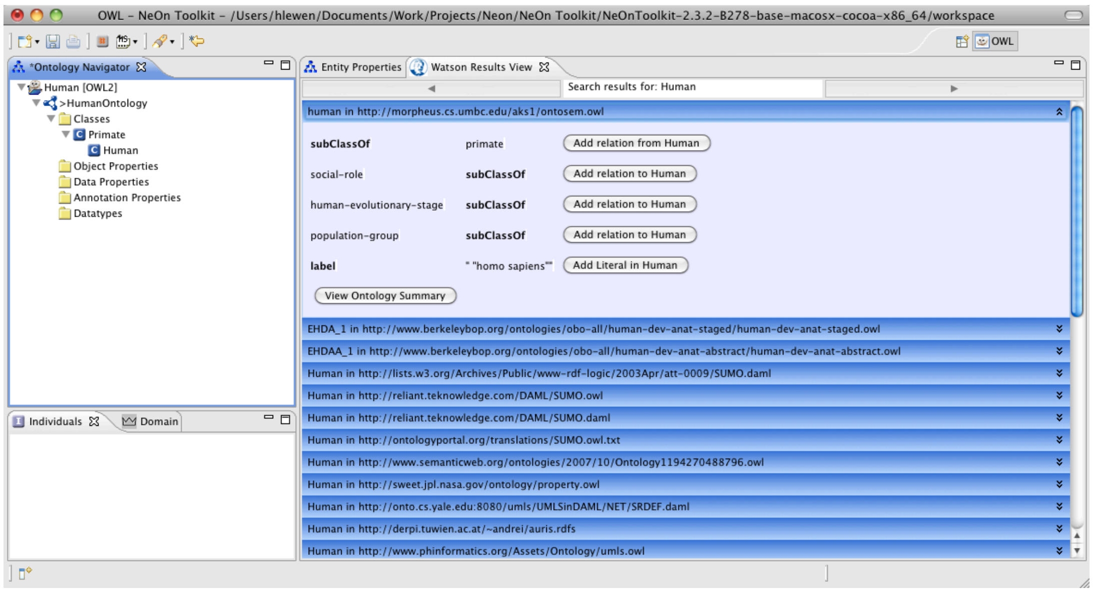
Search results for: Human

```
human in http://morpheus.cs.umbc.edu/aks1/ontosem.owl

subClassOf              primate
social-role             subClassOf
human-evolutionary-stage subClassOf
population-group        subClassOf
label                   " "homo sapiens""

EHDA_1 in http://www.berkeleybop.org/ontologies/obo-all/human-dev-anat-staged/human-dev-anat-staged.owl
EHDAA_1 in http://www.berkeleybop.org/ontologies/obo-all/human-dev-anat-abstract/human-dev-anat-abstract.owl
Human in http://lists.w3.org/Archives/Public/www-rdf-logic/2003Apr/att-0009/SUMO.daml
Human in http://reliant.teknowledge.com/DAML/SUMO.owl
Human in http://reliant.teknowledge.com/DAML/SUMO.daml
Human in http://ontologyportal.org/translations/SUMO.owl.txt
Human in http://www.semanticweb.org/ontologies/2007/10/Ontology1194270488796.owl
Human in http://sweet.jpl.nasa.gov/ontology/property.owl
Human in http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRDEF.daml
Human in http://derpi.tuwien.ac.at/~andrei/auris.rdfs
Human in http://www.phinformatics.org/Assets/Ontology/umls.owl
```

Figure 2.5: By clicking on the arrows next to the URI, axioms from the ontology that matched the search are displayed.



OWL – NeOn Toolkit – /Users/hlewen/Documents/Work/Projects/Neon/NeOn Toolkit/NeOnToolkit-2.3.2-B278-base-macosx-cocoa-x86_64/workspace

```
Human [OWL2]
  >HumanOntology
    Classes
      Primate
        Human
    Object Properties
    Data Properties
    Annotation Properties
    Datatypes
```

Search results for: Human

human in http://morpheus.cs.umbc.edu/aks1/ontosem.owl

subClassOf	primate	Add relation from Human
social-role	**subClassOf**	Add relation to Human
human-evolutionary-stage	**subClassOf**	Add relation to Human
population-group	**subClassOf**	Add relation to Human
label	" "homo sapiens""	Add Literal in Human

EHDA_1 in http://www.berkeleybop.org/ontologies/obo-all/human-dev-anat-staged/human-dev-anat-staged.owl
EHDAA_1 in http://www.berkeleybop.org/ontologies/obo-all/human-dev-anat-abstract/human-dev-anat-abstract.owl
Human in http://lists.w3.org/Archives/Public/www-rdf-logic/2003Apr/att-0009/SUMO.daml
Human in http://reliant.teknowledge.com/DAML/SUMO.owl
Human in http://reliant.teknowledge.com/DAML/SUMO.daml
Human in http://ontologyportal.org/translations/SUMO.owl.txt
Human in http://www.semanticweb.org/ontologies/2007/10/Ontology1194270488796.owl
Human in http://sweet.jpl.nasa.gov/ontology/property.owl
Human in http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRDEF.daml
Human in http://derpi.tuwien.ac.at/~andrei/auris.rdfs
Human in http://www.phinformatics.org/Assets/Ontology/umls.owl

Figure 2.6: When clicking on “Add relation from Human”, the super class “Primate” is added to the ontology automatically. Note the change in the taxonomy on the left hand side.

in general can be found in (Kubias *et al.*, 2007), we present in this section the key findings of our analysis.

In the field of computer science, one can distinguish roughly two general types of trust, namely reputation-based trust and credential-based trust. Reputation-based trust (Agarwal, 2007; Zacharia and Maes, 2000) differs from credential-based trust in that it is based on a user's experience with another user. Normally an assessment is based on past interactions or observations, either directly with the entity assessed, or as recommended by others. In contrast to reputation-based trust, credential-based trust works with authorized entities giving out credentials based on predefined policies. A real life example for credential-based trust could be someone completing a university degree and receiving the official certificate as a credential. In credential-based trust systems a user can improve trust by obtaining more credentials.

Reputation-based Trust

One of the fundamental ideas of reputation-based trust is that a user's behavior in the past can be used to predict future behavior. Let us assume we are interested in whether user A can be trusted. To determine that, we would look at our personal experiences with A, and also take the experience of other users interacting with A into account, to predict future behavior of A. Since most users on the World Wide Web have not met or interacted directly before, reputation mechanisms can provide help when deciding on whether the information an unknown user provides is correct and trustworthy. In order to make use of this information computationally, a conceptual framework is needed.

Conceptual Framework: In (Zhang and Yu, 2004), a conceptual framework for reputation-based trust is provided alongside a classification scheme for reputation-based trust functions. The framework described in (Zhang and Yu, 2004) models a decentralized environment, where different entities interact through transactions. Transactions are unidirectional, meaning there is always a service-provider and a service-receiver. Four terms relevant for the framework are defined:

- *Trustworthiness* gauges the quality of a service an entity provides. In many trust models, the domain of trustworthiness is assumed to be in the interval $[0, 1]$, with either discrete or continuous values.
- *Feedback* is given by the client on the service provider after a transaction has been completed. The feedback can be multidimensional.
- An *opinion* is derived from the feedback of multiple transactions and represents the general impression of a user towards a service provider.
- *Source and Destination of Trust Evaluation:* If A enquires about the trustworthiness of entity B, A is the source and B the destination of a trust evaluation.

In this framework, trust is modeled as a directed multigraph with a set of vertices and a set of labeled edges.

Based on above trust framework, (Zhang and Yu, 2004) proposes a classification scheme for trust functions. It is composed of four dimensions:

- *Subjective Trust vs. Objective Trust:* The first dimension states that in case the quality of an entity can be objectively measured (for example providing information that can also be verified elsewhere), the trust is called objective trust, and the trust is user-independent, meaning it is independent of the source of the trust evaluation. In case the quality can not be objectively measured, the trust is called subjective trust (for example movie recommendations, which are specific to one's point of view). Subjective trust can vary considerably depending on the source of the trust evaluation. The general distinction between subjective and objective trust corresponds to the classification of trust semantics proposed in (Jøsang *et al.*, 2007).
- *Transaction-Based vs. Opinion-Based:* While some trust models rely on the information gained from individual transactions, other trust models rely on opinions. Opinions are normally based on a user's experiences with the service

provider over the last transactions. In general, transaction-based trust functions need more data to infer an entity's trustworthiness than opinion-based trust functions.

- *Complete Information vs. Localized Information:* In case every entity has access to all transaction information or opinions, the trust function is called global trust, and is based on all information. In case the trust function is only applied to a subgraph of the complete graph, the trust function is called local or localized. Because with localized trust functions, each entity has access to different information, a local trust function is also subjective.
- *Rank-Based vs. Threshold-Based:* This dimension is used to distinguish different approaches for deciding whether an entity is to be trusted or not. While threshold-based functions usually check whether the value is above or below a certain threshold, regardless of values for other entities, the rank-based approaches compare the scores for all entities and use this information to provide a relative trustworthiness score for all entities.

Trust Models: Over the years, different trust models and algorithms have been proposed. The seminal work on trust in computer science has been (Marsh, 1994). In his work, Marsh models direct trust between two agents without taking into account recommendations provided by other agents. He uses knowledge, utility, importance, risk, and perceived competence for modeling trust between two agents. With his model he can answer with whom an agent should cooperate, when, and to which extent. Trust values are expressed as real numbers with range $[-1, \dots, 1]$ using threshold based decision making. Marsh defined three kinds of trust:

- *Dispositional trust* is the trust of an agent x independent of a situation or a specific partner. A high value can be interpreted as a high unconditional trust, meaning the agent is in general trustworthy.
- *General trust* is the trust an agent x expresses in an agent y, but independently of a situation.
- *Situational trust* is the trust of an agent x in an agent y given a situation z.

The probability that an entity behaves in an improper or bad way (the risk) is calculated based on a comparison of costs and benefits of the considered engagement. Based on the perceived risk and the competence of a potential interaction partner a cooperation threshold is computed. In case the computed situational trust is above the computed cooperation threshold, the cooperation takes place, otherwise not.

Jennifer Golbeck defines a reputation-based trust model called TidalTrust (Golbeck, 2005). It is intended for social networks. Using a discrete scale, Golbeck allows users to state trust on the interval $[1, \dots, 10]$. The approach is evaluated using a social network called Filmtrust, which is based around users rating movies. One can rate friends based on the question on whether in one's opinion this person shares the movie taste.

Another approach (Abdul-Rahman and Hailes, 2000) models trust between two agents based on (Marsh, 1994). In order to determine whether another agent A should be trusted, the system takes into account both an agent's direct experience with A and recommendations of other agents on A.

Most of the aforementioned models rely on rather simple algorithms and mathematical constructs like weighted summation. Mathematically more sophisticated trust models usually involve aspects of probability theory. In (Jøsang, 2001), elements of belief theory are combined with elements of Bayesian probability theory to form what is referred to as "subjective logic". The belief theory constructs allow to take uncertainty into account.

Linguistically fuzzy concepts can also model trust and reputation. In this case, membership functions determine to what extent an agent is trustworthy or not trustworthy. Reasoning with these fuzzy values is realized with fuzzy logic. Exemplary approaches relying on fuzzy logic are (Sabater and Sierra, 2002a; Sabater and Sierra, 2002b).

Credential-based Trust

Since credential-based trust is not essential for the approach presented in this thesis, we will only give a short overview. In general, credential-based trust systems are used to regulate access to resources. That means the credentials can be used to gain access to content. The concept of credential-based trust management is thus mainly concerned with finding ways to verify credentials and control access to resources based on pre-defined policies. Credentials in this context can be verified either directly or through a Web of Trust (WOT).

In relation with the World Wide Web, access control and security mechanisms are increasingly important, not only for businesses. Security related aspects discussed in literature usually fall into one or more of the following three categories: confidentiality, integrity, and availability (Bishop, 2003; Samarati and di Vimercati, 2001). Since acquiring credentials can be bothersome and cost-intensive, many Web-pages use a simple identity-based authentication, which means that the user must be known to the service or Web site provider, for instance by a previous registration. In case a user does not want to disclose his identity, or a reliable authentication is not possible, "authorization"-based access control of Web services is proposed. Authorization-based access controls can act as a supporting architecture for credential-based trust systems, also on the Semantic Web.

Most credential-based trust system need a working public key infrastructure (PKI), which can be provided for example by the Simple Public Key Infrastructure/Simple Distributed Security Infrastructure (SPKI/SDSI) (Ellison *et al.*, 1999; Clarke *et al.*, 2001). Pretty Good Privacy (PGP) is the best known PKI infrastructure which was created by Zimmermann (Zimmermann, 1995). PGP is a software solution that offers both cryptographic privacy (encryption) and authentification (allowing to sign digital content). It does not offer a hierarchical certification infrastructure, but relies on a WOT that is build by users signing digitally the public keys of other users. The idea is that if enough users sign the key of user A, the signed key is trustworthy and belongs to user A. In order to avoid attacks, the trusting users should ideally also be trustworthy themselves.

2.5.2 Propagation of Trust

After giving an overview on common trust models and architectures in computer science in Section 2.5.1, we now focus on properties of trust and algorithms making use of trust information.

Properties of Trust

Especially when trust is modeled as a graph, certain assumptions have to be made about general properties of trust in order to be able to use trust within computations. Jennifer Golbeck presents four properties of trust that are of central importance, namely transitivity, composability, personalization, and asymmetry (Golbeck, 2005).

Transitivity: The most basic and most widely accepted property of trust in computer systems is transitivity (Guha, 2004; Guha *et al.*, 2004; Golbeck, 2005). Transitivity is needed to infer trust relationships between users who are not directly connected. The general idea is that if A trusts B and B trusts C, A should also trust C to some extend, unless information to the contrary is known to the system (for example a distrust statement from A to C). Another general assumption is that trust is decaying when being propagated. That means that my trust in someone I trust directly is higher than in someone I do not know directly and trust because of propagated trust. Most systems working with trust therefore model a decay factor when propagating trust (Guha, 2004; Guha *et al.*, 2004). The general idea of transitive trust stems from observing human behavior. It is common to turn to friends one trusts when seeking advice. Let us assume that we turn to our friend B for advice, but B cannot help us. Nonetheless, B knows and trusts C, who is then recommended to us. In that scenario, we are likely to also trust C's advice, even though we do not know C directly. Bad advice in the past is likely going to change our attitude towards the recommender, thus penalizing misuse of trust.

Composability: In the real world, entities receiving recommendations or trust from multiple sources are perceived more trustworthy. The idea is that trust can be composed from several sources. The situation becomes more difficult in case not only trust, but also distrust statements should be combined towards a common trust score.

Personalization: In most cases, trust is something subjective. Somebody we trust can be distrusted by others and vice versa. Therefore it is important to allow personalized trust. Systems employing a WOT usually allow personalized trust, since it is possible to store trust scores for each user to user relation. The results of applying the PageRank algorithm (Page *et al.*, 1998), for example, are not personalized, since it is a global measure, meaning it will be the same for all users.

Asymmetry: Trust does not have to be mutual. Especially in anonymous settings like the WWW it is important to model trust as unidirectional relations. I can trust someone without this person knowing me, so it would be counter-intuitive to assume that trust connections are symmetric by default. Furthermore, only by assuming asymmetry one is immune against people trying to trust others for their own benefit, for example to boost their own trustworthiness by having more symmetric trust connections. While cases of symmetric trust exist (every time two users trust each other), this should rather be modeled as two unidirectional links than one bidirectional link, taking into account the formal independence of the trust statements.

Semantics of Trust

In (Guha, 2004), Guha gives two more formal equations of the semantics of trust.

$$\mathit{trust}(A, B) \Rightarrow (P_A(s) \ll P_A(s|\mathit{believes}(B, s))) \tag{2.1}$$

where $\mathit{trust}(A, B)$ means that A trusts B, $P_A(s)$ refers to the a priori probability assigned by A to statement s, $\mathit{believes}(B, s)$ means B believes that s is true, and $P_A(s|\mathit{believes}(B, s))$ refers to the probability assigned by A to s based on B stating s were true.

In other words, people are more likely to believe a statement is true when it is confirmed by someone they trust. The second equation covers decay of propagated trust.

$$\mathit{trust}(A, B) \Rightarrow (P_A(s)) = \alpha \times P_B(s)) \tag{2.2}$$

where $P_B(s)$ is the probability assigned to statement s by B, and α is a decay factor such that $\alpha^m \approx 0$ and m being the number of steps trust should decay out in a Web of Trust.

This equation represents the decay of trust over propagation chains. B assigns a certain probability to a statement s, and A trusts B. The probability A assigns to the statement s is smaller (by factor α) than the probability B assigns to s.

Propagation of Trust

The seminal work on the propagation of trust and especially also of distrust is (Guha *et al.*, 2004). Since their work also strongly influenced the algorithms used in this thesis, we explain their work in more detail. In general, for their paper Guha et al assume the existence of trust and potentially also distrust information between users. The challenge is to combine both trust and distrust and try to infer as many new trust connections as possible from the given data. For that, the basic ideas of transitivity, composability and decay of trust are used.

Basic Trust Propagation Types: The propagation itself is performed on matrices storing the trust and distrust. Simple direct trust propagation can be seen as a matrix multiplication of the trust matrix with itself. In order to get the most information out of potentially sparse trust data, four basic types of trust propagation are defined (for a visualization see Figure 2.7):

- Direct propagation: If A trusts B, then someone trusted by B should also be trusted by A. This type of propagation is based on the basic transitivity assumption.

- Co-citation: If A trusts B and C, then D trusting C should also trust B. This propagation type is based on the idea that by trusting the same person, A and D behave similarly and thus are likely to trust the same individuals.

- Transpose trust: If A trusts B, someone trusting B should also trust A. This propagation type takes the idea of co-citation one step further, assuming that if two people trust the same person, they should also trust each other.

- Trust coupling: If A and B trust C, someone trusting A should also trust B. For the final type of trust propagation, the idea is to find people similar to the people one trusts and assume one should also trust them.

Of course, except for the more or less agreed upon concept of direct propagation, the other propagation steps are debatable and make bigger assumptions about the nature of a trust network. For that reason, the four different techniques are combined using a linear combination into a final propagation matrix, which is then used for the computation of trust. Depending on how the weights are set, more emphasis can be put on the less debatable propagation techniques.

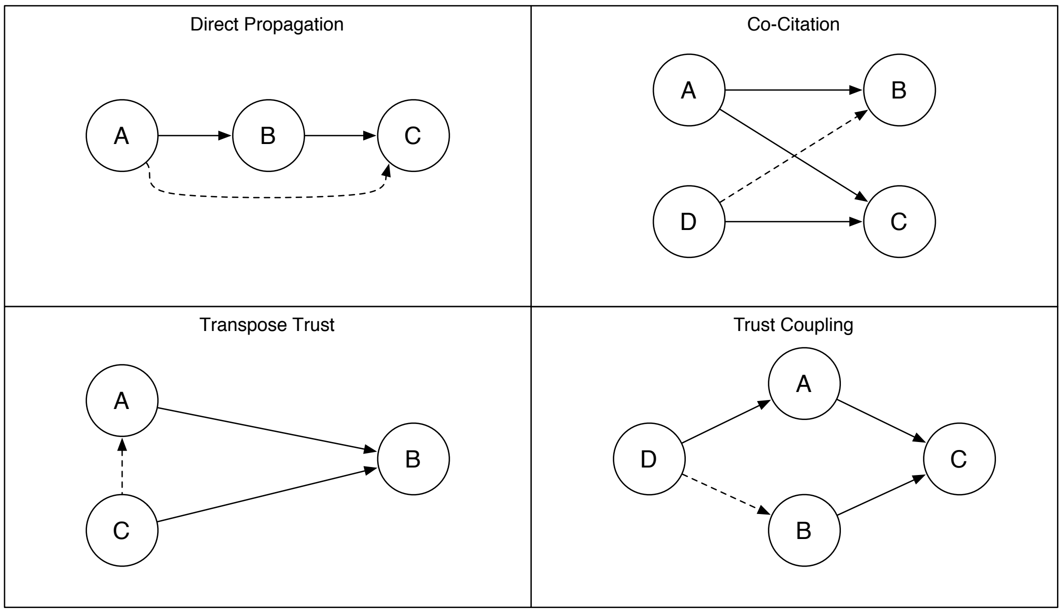


Figure 2.7: The four basic trust propagation types visualized. Dotted lines indicate propagated or inferred trust.

Propagation of Distrust: While the propagation of trust is relatively straight forward, the propagation of distrust is more problematic. Mainly because distrust is not necessarily transitive. Sometimes we say that your enemy's enemy is your friend. This would imply that if A distrusts B, and B distrusts C, A should trust C. Another possible interpretation is taking trust as a form of evaluation of somebody's ability to perform a certain task. In this case, it makes sense to even compose distrust values over propagation chains. The idea here is that if A distrusts B, and B distrusts C, A should distrust C more than B. Guha et al solve this dilemma by determining by means of a user evaluation that the best results can be achieved if distrust is only propagated one step. We will explain their algorithm in more detail in Chapter 4.

2.6 Open Rating Systems

An Open Rating System (ORS) (Guha, 2004) is a system that allows users to write reviews and give ratings on arbitrary content. Other users can then trust or distrust the reviewers. Based on the trust information and the reviews collected, the system can generate a ranking for both reviews and reviewed content. A key assumption within an ORS is that users are influenced by opinions from people they trust (Tversky and

Kahneman, 1974). A prominent example for an Open Rating System is Epinions,[22] which promises "unbiased reviews by real people" about "millions of products and services". Epinions allows users to write reviews on any indexed product or service. Other users can comment on the reviews, and decide to trust or distrust the reviewers.

The concept of ratings and meta-ratings (other users rating the ratings) has found wide adoption in the e-commerce world. In Amazon,[23] users can enter product reviews, and other users can click whether they find the reviews helpful or not.

For a better understanding on how an ORS works, we explain the ORS model in the following. One of the key components of the ORS model is the trust users express towards each other. The Web of Trust (WOT) these user-to-user trust statements form can be used for both *local* (user-specific) and *global* (user-agnostic) trust computation, as shown in Chapter 4 Section 4.2.

2.6.1 Model

Guha's ORS model (Guha, 2004) consists of the following components:

1. A set of objects $O : \{o_1, o_2, o_3, \ldots\}$ that can be rated. These can for example be products, services, or any other entity that can be rated.

2. A set of agents $A : \{a_1, a_2, a_3, \ldots\}$ that participate in the ORS. Normally these are users, but theoretically machines could participate as well, if they have an algorithm for automatically evaluating the objects.

3. A set of possible values for ratings of objects $D : \{d_1, d_2, \ldots\}$. These values determine what is the range of ratings on objects. Often, a five star scale is used, where 1 star means bad and 5 stars mean very good.

4. A set of possible values for trust ratings of agents $T : \{t_1, t_2, \ldots\}$. These values determine the range of trust statements. In many systems only binary decisions, for example trust or distrust, helpful or not helpful, are allowed.

5. A partial function $R : A \times O \rightarrow D$ corresponding to agent ratings on objects.

6. A partial function $W : A \times A \rightarrow T$ corresponding to inter-agent trust.

A depiction of the model can be found in Figure 2.8.

[22]`http://www.epinions.com/`, last checked on 24.11.2010
[23]`http://www.amazon.com/`, last checked on 24.11.2010

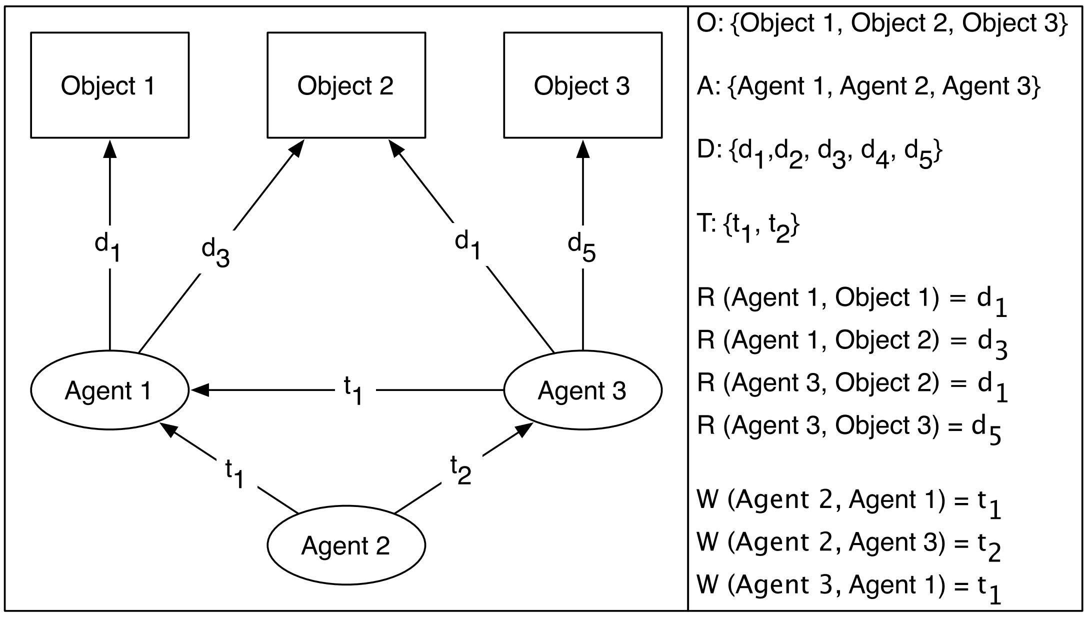

Object 1
Object 2
Object 3
d1
d3
d1
d5
Agent 1
t1
Agent 3
t1
t2
Agent 2
O: {Object 1, Object 2, Object 3}
A: {Agent 1, Agent 2, Agent 3}
D: {d1,d2, d3, d4, d5}
T: {t1, t2}
R (Agent 1, Object 1) = d1
R (Agent 1, Object 2) = d3
R (Agent 3, Object 2) = d1
R (Agent 3, Object 3) = d5
W (Agent 2, Agent 1) = t1
W (Agent 2, Agent 3) = t2
W (Agent 3, Agent 1) = t1

Figure 2.8: A depiction of Guha's ORS model.

2.6.2 Rating and Ranking

Within the ORS, two main problems, namely that of *rating* and *ranking* have to be addressed. *Rating* refers to the task of predicting how an agent A would rate an Object O, given that the agent has not rated the object yet. *Ranking* refers to the task of arranging a set of objects in descending importance in a manner that is personally optimal for an agent A. Because normally only the top N items are of interest, the users can state how many objects should appear in their result set.

Guha addresses the task of rating by exploring an agent's WOT. Let us assume that the system has to provide a rating for agent A_i on object O_j. The system now checks the WOT if A_i has trusted users that have rated O_j. If yes, their ratings can be used to infer a likely rating for A_i. If not, a global measure is used to determine the overall most trusted agent that has rated O_j.

After studying and presenting several algorithms for computing a global trust measure, Guha presented two equations which he called *TrustRank* (see Equation 2.3) and *DistrustRank* (see Equation 2.4).

$$\mathit{TrustRank}_{N+1}(A_u) = \sum\nolimits_{v \in T_v} \frac{\mathit{TrustRank}_N(v)}{N_v} \tag{2.3}$$

where A_u is the user whose *TrustRank* is computed, T_v is the set of users trusting A_u, $v \in T_v$ is a user from T_v trusting A_u, N_v is the total number of users user v

trusts, and N is the number of iterations the algorithm has run. The TrustRank values can be initialized as 1. Intuitively speaking, *TrustRank* assigns trust to users based on how many other users trust them and how trusted the users trusting them are. Readers familiar with the PageRank algorithm (Page *et al.*, 1998) will note both the resemblance of the algorithm, but also the absence of a damping factor. Indeed, it is unclear why Guha has not incorporated a damping factor into the equations, since without it, the algorithm runs into problems when there are sinks in the graph. These sinks will acquire almost all of the trust.

$$\mathit{DistrustRank}(A_u) = \sum\nolimits_{v \in B_v} \frac{\mathit{TrustRank}(v)}{N_v} \tag{2.4}$$

where A_u is the user whose *DistrustRank* is computed, B_v is the set of users distrusting A_u, $v \in B_v$ is a user from B_v distrusting A_u, and N_v is the total number of users user v distrusts. *DistrustRank* is taking into account who distrusts a user and how high the *TrustRank* of the distrusting users are.

Guha proposes to use either TrustRank alone for determining a global rank of users, or a combination of TrustRank and DistrustRank, depending on the application.

2.6.3 ORS for Ontology Rating

The idea to exploit ORS for ontology ranking dates back to (Noy *et al.*, 2005). The model is adapted for rating ontologies by taking ontologies as the objects, and users of the ORS as agents. As a rating scale for the ontologies, the common 5 star rating scale can be used. KnowledgeZone (Supekar *et al.*, 2007) which focused on biomedical ontologies, build upon the ORS as well to enable a user rating of ontologies. KnowledgeZone is not available anymore, since it was superseded by the NCBO's Bioportal, which also plans to employ a rating system allowing users to contribute to the ontology evaluation (Noy *et al.*, 2008). However, the approach presented in (Guha, 2004) and adapted in (Supekar *et al.*, 2007) has some limitations that hinders its application without modifications in specialized scenarios like ontology rating. We present the limitations we found and confirmed by analyzing data in the following.

2.6.4 Limitations of the ORS

The ORS model as presented by Guha in (Guha, 2004) has some limitations when applied in specialized domains like ontology rating.

First, there is no concept of a multi-faceted rating, that means, it is only possible to provide an overall rating for an object, and not for the different aspects of an object (the rating function R corresponds to agents rating entire objects). In case certain aspects of an object are good, and others are bad, this subtlety is lost. In the context of ontology evaluation, for example, an ontology can be highly reusable, while only covering a limited part of the intended domain. In this case, reusability and domain coverage represent aspects that can be reviewed independently. Analyzing data from Cupboard (Cupboard is our ontology hosting system which is introduced in Chapter 9), we discovered that indeed many ontologies have a variance in the ratings on their different aspects. As an example, in the corresponding ontology space[24] user Jérôme rated the biosphere ontology 5 stars on domain coverage, but only 2 stars on reusability. More generally, in this ontology space we looked for cases where a reviewer provided reviews for every aspect of an ontology, allowing to compute an average rating this reviewer might have given in case only an overall rating would have been allowed. Cases where reviewers only provided a rating for some, but not all of the five properties were dismissed. The data can be found in Table 2.1. Based on 145 ratings, which would correspond to 29 overall ratings (single faceted reviews) in the ORS, we computed the average variance between aspects as 0.85. Within the reviews, variances ranged from 0 to 2.2. In order to keep this information, multi-faceted ratings have to be allowed.

Second, reviewers do not always write either good or bad reviews, but depending on the ontology and aspect they review, they can be trusted differently. Indeed, we observed several users that trust a review from a particular reviewer and distrust another review from that same reviewer. The easiest way to analyze the problem here, without sacrificing the privacy of the users, is to compare the ratings of different users for the same ontology. The idea here is that in case two users give the same rating, they will trust each other for that rating, and in case the rating differs, they do not agree and thus do not trust each others rating. In short, we analyze the agreement of users with respect to their ratings on the same ontology. Problematic for the basic ORS model are all cases where there are some agreements and some disagreements. In case the users share a common perception of the quality of an ontology and give the same ratings, we can assume that the users trust each other for all their ratings. In case the ratings do all differ, we can assume the users distrust each other. These both cases can be handled by the basic ORS model, because there are no conflicting trust statements.

Analyzing the data shown in Table 2.2, we find that indeed there is a heterogeneity in the rating agreement between the reviewers. In most cases, they agree on some ratings, but disagree on others. This cannot be modeled in the ORS model, which only allows one trust value for an agent-to-agent trust connection. The trust function W corresponds to global trust or distrust (covering all reviews of a user). Even if the range of possible trust statements were big enough to cover all possible combinations of trust and distrust, by combining the trust into a single, global value, the information on which rating was trusted and which not is lost. We argue that this is an unnecessary

[24] http://cupboard.open.ac.uk:8081/cupboard/Experiment1, last checked on 24.11.2010

Table 2.1: The data from Cupboard shows the variance in ratings on the different ontology aspects.

User	Ontology	Reusa-bility	Correct-ness	Com-plexity	Domain Coverage	Mod-eling	Aver-age	Vari-ance
1	1	4	5	4	4	4	4.2	0.16
2	1	4	4	3	5	4	4	0.4
1	2	4	4	4	4	4	4	0
1	3	2	5	3	3	3	3.2	0.96
3	3	4	4	4	2	2	3.2	0.96
1	4	2	4	3	3	3	3	0.4
1	5	5	5	5	3	4	4.4	0.64
1	6	4	4	4	2	5	3.8	0.96
2	6	5	4	4	4	5	4.4	0.24
1	7	4	4	5	5	5	4.6	0.24
4	7	2	5	2	5	5	3.8	2.16
1	8	2	4	2	1	1	2	1.2
3	8	3	4	4	2	3	3.2	0.56
6	8	2	4	2	5	3	3.2	1.36
1	9	3	4	2	2	2	2.6	0.64
4	9	3	2	3	1	2	2.2	0.56
1	10	1	4	3	1	2	2.2	1.36
1	11	4	5	5	3	5	4.4	0.64
1	12	2	4	3	1	2	2.4	1.04
6	12	2	4	1	2	1	2	1.2
1	13	4	5	4	2	2	3.4	1.44
5	13	5	2	4	1	4	3.2	2.16
1	14	1	1	1	1	1	1	0
6	14	1	1	1	1	1	1	0
1	15	2	5	3	3	3	3.2	0.96
1	16	2	5	3	3	3	3.2	0.96
1	17	2	3	3	3	1	2.4	0.64
1	18	2	4	3	1	4	2.8	1.36
2	18	1	3	4	1	3	2.4	1.44

constraint and that there is a need for a more fine-grained trust management, including a way to remember which review was trusted and which was not trusted. Otherwise either important information about trust, or about distrust is lost.

Table 2.2: The data from Cupboard compares the ratings of users who have rated the same ontology and shows on how many ratings they agreed, and on how many they disagreed.

User	Ontology	Reusability	Correctness	Complexity	Domain Coverage	Modeling	Agreement	Disagreement
1	1	4	5	4	4	4	2	3
2	1	4	4	3	5	4	2	3
1	3	2	5	3	3	3	0	5
3	3	4	4	4	2	2	0	5
1	6	4	4	4	2	5	3	2
2	6	5	4	4	4	5	3	2
1	7	4	4	5	5	5	2	3
4	7	2	5	2	5	5	2	3
1	8	2	4	2	1	1	1	4
3	8	3	4	4	2	3	1	4
1	8	2	4	2	1	1	3	2
6	8	2	4	2	5	3	3	2
3	8	3	4	4	2	3	2	3
6	8	2	4	2	5	3	2	3
1	9	3	4	2	2	2	2	3
4	9	3	2	3	1	2	2	3
1	12	2	4	3	1	2	2	3
6	12	2	4	1	2	1	2	3
1	13	4	5	4	2	2	1	4
5	13	5	2	4	1	4	1	4
1	14	1	1	1	1	1	5	0
6	14	1	1	1	1	1	5	0
1	18	2	4	3	1	4	1	4
2	18	1	3	4	1	3	1	4

Part II

Topic-Specific Trust Open Rating System For Ontology Reuse

Chapter 3

Topic-Specific Trust Open Rating System Model

Motivated by the limitations discussed in Chapter 2 Section 2.6.4, we investigated how the behavior of an Open Rating System can be improved. The two main requirements for an ORS extension are enabling multi-faceted rating of objects, and a fine-granular trust management. The trust management has to allow users to trust a user A for some objects, but distrust A for the rest of the objects. Furthermore, the information which review was considered trustworthy and which review was considered not trustworthy should be stored and exploited during the trust computation.

First ideas towards an extension were published in (Lewen, 2005; Lewen *et al.*, 2006). We have refined the ideas to come up with the extension presented in the following, the Topic-Specific Trust Open Rating System (TS-ORS), which has been published at EKAW 2010 (Lewen and d'Aquin, 2010). In contrast to the previously published ideas, the work presented in this thesis has been implemented in a real-world system (see Part III) and evaluated in both a simulation and a user study (see Part IV).

In this chapter, we first present the TS-ORS Model in Section 3.1 and the idea behind meta-trust statements in Section 3.2. We then present possible extensions of the TS-ORS Model in Sections 3.3 and 3.4. Parts of this chapter are based on (Lewen and d'Aquin, 2010; Lewen *et al.*, 2006; Lewen, 2005; Sabou *et al.*, 2007).

3.1 TS-ORS Model

In order to make the transition from an existing ORS to the TS-ORS as easy as possible, we decided to adapt the ORS model only where needed, while keeping the basic notions. In order to allow multi-faceted ratings, we first introduce rateable aspects of an object, and then extend the rating function R. We furthermore adapt the trust function W by changing its signature to allow trust statements at the level of an aspect of an object. Since trust is used to connect users in a WOT, it is advisable to also allow trust statements for aspects of objects a user has not reviewed, because these trust connections can be used during the trust propagation phase. Classifying the trust according to the framework of (Zhang and Yu, 2004) presented in Chapter 2 Section 2.5.1, the trust between users in the TS-ORS is subjective, because there are no objectively correct ratings, and the same user can be trusted by some users and distrusted by other users. The users themselves do not have complete information on the trust network, they only know who they trust and distrust, but not who trusts them or any other trust relations between users. Furthermore, with the introduction of topic-specific trust, trust can now be defined in a context (the aspect of an object) and not only globally (for all objects and aspects).

Compared to the model presented in Chapter 2 Section 2.6.1, for our TS-ORS model we reuse O, A, D, T, introduce object aspects X and change the signature of R and W. Our TS-ORS model thus consists of:

1. A set of objects $O : \{o_1, o_2, o_3, \ldots\}$ that can be rated. In our use case, we assume the objects to be ontologies or parts of ontologies.

2. A set of agents $A : \{a_1, a_2, a_3, \ldots\}$ that can provide ratings and trust statements. Agents can be both human users or automatic agents.

3. A set of object aspects $X : \{x_1, x_2, x_3, \ldots\}$. How the aspects are instantiated in an implementation of the model depends on the intended use case. For Cupboard, we use reusability, correctness, complexity, domain coverage and modeling as object aspects (see Chapter 5).

4. A set of possible values for ratings of objects $D : \{d_1, d_2, \ldots\}$. We assume that only interval rating scales are used which allow statistical computations with the values. We furthermore assume that reviewers will know how to interpret the rating scale, i.e., know the order of ratings from bad to good. For ontology rating, we employ the common 5 star rating scale, and assume that the different star ratings are equidistant and the rating scale is an interval scale (meaning it is permitted to compute the average of 5 stars and 3 stars as 4 stars).

5. A set of possible values for trust ratings of agents $T : \{t_1, t_2, \ldots\}$. While the algorithms can handle non-discrete values, for Cupboard we just use trust and distrust.

6. A partial function $R : A \times O \times X \to D$. R corresponds to the rating of an agent on one certain aspect of an object. Let $B_R \subseteq A \times O \times X$ denote the set of all triples for which R is defined, i.e., for which ratings exist. In the context of Cupboard, R covers ratings from users on one aspect of an ontology, for example on the complexity of the Gene Ontology.

7. A partial function $W : A \times A \times O \times X \to T$, which corresponds to the trust of an agent in another agent for a specific object–aspect combination. In Cupboard, a user can trust another user for example to rate the complexity of the Gene Ontology, and not more.

We assume all sets to be finite. In this thesis we use the term *user* or *reviewer* to refer to agents from A. In our model, a reviewer has to justify each rating R with a textual review explaining the motives for giving that rating. We refer to the textual justification as the *review*, and to the actual D value as the *rating*. A depiction of our TS-ORS model can be found in Figure 3.1.

We separate the description of the model (found in this chapter) and algorithms (found in Chapter 4) from our concrete instantiation in Chapter 5. The adaptation of the TS-ORS for ontology ranking alongside default values for parameters from the following algorithms can be found in Chapter 5.

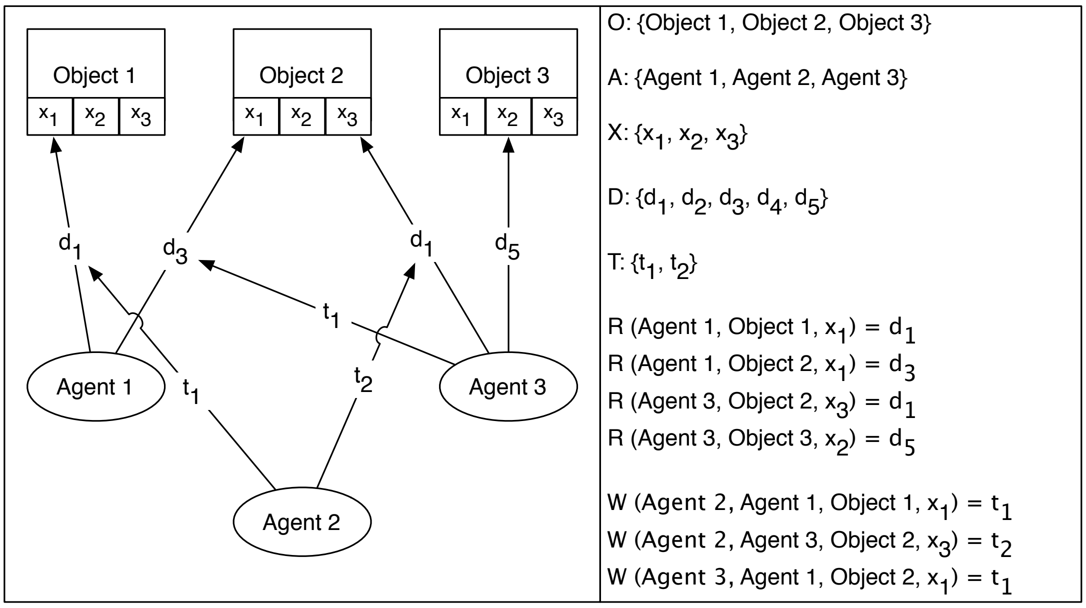

Object 1
Object 2
Object 3
x1 x2 x3
d1
d3
d1
d5
t1
t1
t2
Agent 1
Agent 2
Agent 3
O: {Object 1, Object 2, Object 3}
A: {Agent 1, Agent 2, Agent 3}
X: {x1, x2, x3}
D: {d1, d2, d3, d4, d5}
T: {t1, t2}
R (Agent 1, Object 1, x1) = d1
R (Agent 1, Object 2, x1) = d3
R (Agent 3, Object 2, x3) = d1
R (Agent 3, Object 3, x2) = d5
W (Agent 2, Agent 1, Object 1, x1) = t1
W (Agent 2, Agent 3, Object 2, x3) = t2
W (Agent 3, Agent 1, Object 2, x1) = t1

Figure 3.1: A depiction of our TS-ORS model.

3.2 Meta-trust Statements in the TS-ORS Model

Per default, users express trust on the level of one reviewer of an aspect of an object (see definition of W). For the convenience of users, we define and allow the use of meta-trust statements, which are trust statements covering more than one object–aspect combination. They can be seen as shortcuts to making many W statements (see Table 3.1). The intuition here is that it is cumbersome to make several fine granular trust statements when the decision to trust another user on a coarser level has been made. Let us assume, a user wants to trust another user for a complete object.

Table 3.1: Allowed Meta-trust Statements.

Statement	**Signature**	**Explanation**
W_O	$A \times A \times X \rightarrow T$	Statement on a specific aspect of all objects (for arbitrary objects of O)
W_X	$A \times A \times O \rightarrow T$	Statement on all aspects of a specific object (for arbitrary aspects of X)
W_{OX}	$A \times A \rightarrow T$	Statement on all aspects of all objects (for arbitrary aspects of X and objects of O)

This could then be done with one W_X meta-trust statement instead of multiple W statements.

3.3 Domain Extension of the TS-ORS Model

While we use the aforementioned model in our implementation, in other scenarios it can be required to extend the model to provide additional functionality. In this section, we present a possible extension for the ontology rating use-case. Even though this example is tailored towards ontologies, which normally cover a domain of knowledge, the idea is generalizable to other applications which also allow a categorization of objects O.

For our extension we assume that the objects can be classified into a taxonomy. An example of such a classification can be found at online vendors, whose product catalogue usually is based on a taxonomy of categories into which products are classified (e.g., a doll can be classified as a toy). In the use case of ontology rating, we can assume that an ontology has an associated domain it covers, and that the domains are arranged in a taxonomy like DMOZ.[1]

3.3.1 Extended TS-ORS Model

We extend the TS-ORS model to cover domains, a taxonomy of domains, and a relation linking objects to domains. Apart from the components described in Section 3.1, we introduce:

- A set of concepts representing domains $C : \{c_1, c_2, c_3, \ldots\}$. In the case of ontology rating, the concepts represent possible domains of an ontology. The concepts should not be confused with concepts within the ontologies, but are elements of the taxonomy H_C.

[1]http://www.dmoz.org/, last checked on 24.11.2010

- A taxonomy $H_C \subset C \times C$, where H_C is a directed,[2] acyclic,[3] irreflexive,[4] transitive[5] relation. $(C_i, C_j) \in H_C$ means that C_i is a subconcept of C_j.

- A relation $L \subseteq O \times C$. This relation corresponds to an n-to-m mapping between objects and their domains, it can thus be read as a "has domain" relation. In the case of ontology rating, it is possible for an ontology to cover more than one domain, and also for a domain to be covered by more than one ontology. Let L^{-1} be the inverse relation of L and $H_C^{-1}(C_j)$ be the inverse relation of H_C. Then $L^{-1}(H_C^{-1}(C_j))$ provide a set of objects, which has as domain either C_j or any subconcept of C_j.

The information available in the taxonomy can be exploited in the ontology rating scenario both during a search for ontologies (to filter ontologies covering a specified domain), or also during the meta-trust assignment. A depiction of the extended TS-ORS model can be found in Figure 3.2.

3.3.2 Meta-trust Statements Using Domain Information

Apart from the already known meta-trust statements, in the extension we also permit meta-trust statements that make use of the fact that objects are mapped to domains via L. For that we introduce W_{O_C} and W_{X_C} as found in Table 3.2. An example would be that by trusting user A to rate cars (all objects classified under car), one would also trust A to rate BMW cars. In the case of ontology ratings, trusting a reviewer to rate ontologies with the domain "body part" would also imply trusting this reviewer to rate ontologies with domain "arm", given that "arm" is a subconcept of "body part". We assume that in the taxonomy, a subconcept relation between "arm" and "body part" would exist, either directly or through a chain of subconcept relations.

One could take the taxonomy idea even further and assume that the aspects X are themselves organized in a taxonomy (Lewen, 2005). In real applications the number of aspects is normally limited, and a taxonomy of aspects will not provide much benefit. If needed, however, the model could be simply extended by adding a taxonomy of aspects and adding meta-trust statements taking the taxonomy of aspects into account.

[2]Directed means that the relation is asymmetric, i.e., if A is a subconcept of B, B is not automatically a subconcept of A.

[3]Acyclic means that cycles like A is a subconcept of B, B is a subconcept of C and C is a subconcept of A are not allowed.

[4]Irreflexive means that an element cannot be in relation with itself. A concept cannot be a subconcept of itself.

[5]Transitive means that if A is a subconcept of B, and B is a subconcept of C, A is also a subconcept of C.

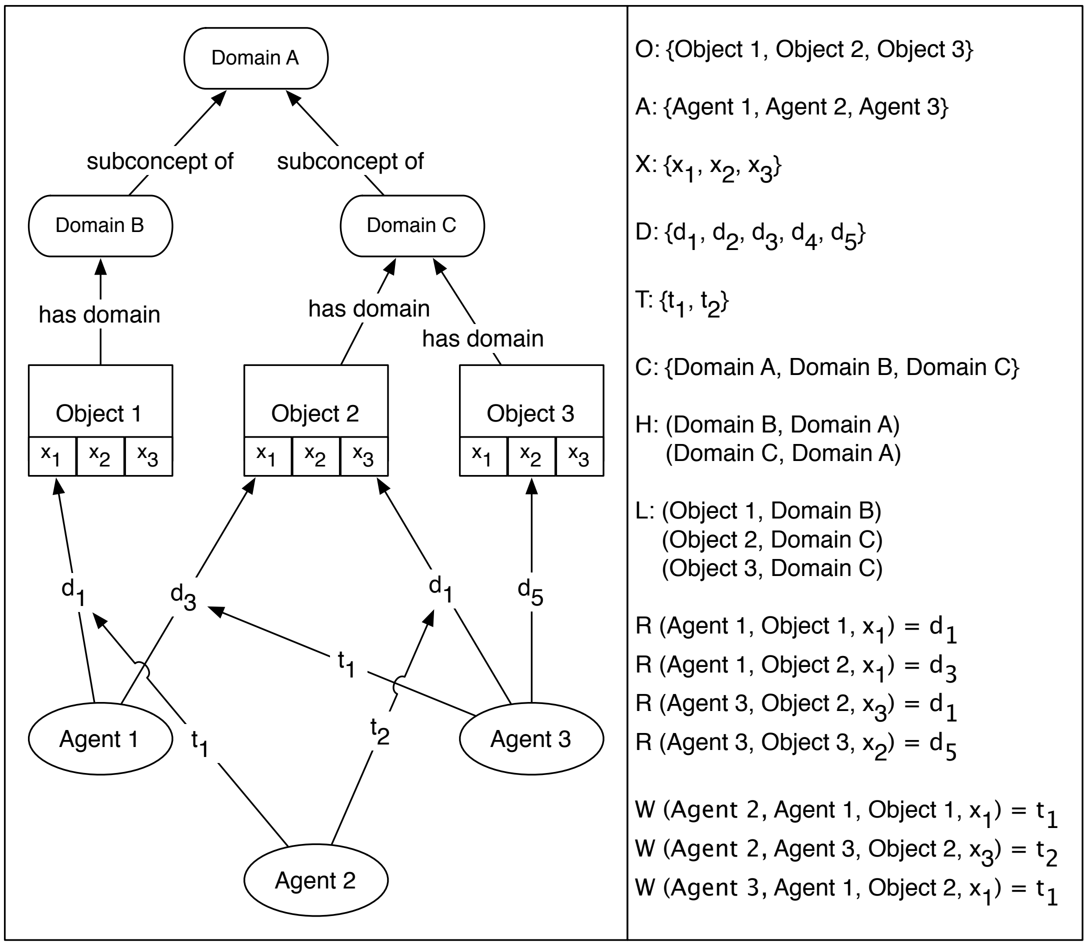

Domain A
subconcept of
subconcept of
Domain B
Domain C
has domain
has domain
has domain
Object 1
Object 2
Object 3
x1 x2 x3
d1
d3
d1
d5
t1
t1
t2
Agent 1
Agent 2
Agent 3
O: {Object 1, Object 2, Object 3}
A: {Agent 1, Agent 2, Agent 3}
X: {x1, x2, x3}
D: {d1, d2, d3, d4, d5}
T: {t1, t2}
C: {Domain A, Domain B, Domain C}
H: (Domain B, Domain A) (Domain C, Domain A)
L: (Object 1, Domain B) (Object 2, Domain C) (Object 3, Domain C)
R (Agent 1, Object 1, x1) = d1
R (Agent 1, Object 2, x1) = d3
R (Agent 3, Object 2, x3) = d1
R (Agent 3, Object 3, x2) = d5
W (Agent 2, Agent 1, Object 1, x1) = t1
W (Agent 2, Agent 3, Object 2, x3) = t2
W (Agent 3, Agent 1, Object 2, x1) = t1

Figure 3.2: A depiction of our TS-ORS model with an extension for domains.

Table 3.2: Allowed Meta-trust Statements in an Extended TS-ORS.

Statement	**Signature**	**Explanation**
W_{O_C}	$A \times A \times C \times X \rightarrow T$	Statement on a specific aspect of all objects belonging to a domain C
W_O	$A \times A \times X \rightarrow T$	Statement on a specific aspect of all objects (for arbitrary objects of O)
W_X	$A \times A \times O \rightarrow T$	Statement on all aspects of a specific object (for arbitrary aspects of X)
W_{X_C}	$A \times A \times C \rightarrow T$	Statement on all aspects of all objects belonging to a domain C (for arbitrary aspects of X)
W_{OX}	$A \times A \rightarrow T$	Statement on all aspects of all objects (for arbitrary aspects of X and objects of O)

3.4 Evaluation-Context Extension for the TS-ORS Model

As mentioned in Section 2.3, ontologies can be evaluated with regard to the task they are designed for. As there are several potential application scenarios for an ontology (see Section 2.1), an ontology can be great for one application, but unsuitable for another. An upper level ontology, for example, will not perform well in an application that needs specific domain knowledge. Therefore it is clear that all ontologies are evaluated according to requirements which are imposed by the context (e.g., application) in which the ontology is employed. While the context of an evaluation is important information, it is not modeled explicitly in our TS-ORS model. First, we assume that each ontology was built with a specific task in mind, and should be evaluated against its task. While it is possible to evaluate ontologies against tasks which they were not designed for, namely in cases where an ontology can be applied in more than one task, we assume that normally an ontology will be evaluated within a specific context.

In our TS-ORS model, each rating is justified with a review. Within the review text, information such as the context in which or the task against which an ontology was evaluated can be stated. We assume that users can read the reviews for an ontology, and then decide whether to trust the reviewer or not, even without explicit formalization of context. The context would be selected implicitly by users trusting reviewers that rate ontologies in the same context they intend to employ the ontology in. Nevertheless, the context of a rating can be formalized and made explicit in the TS-ORS model.

3.4.1 Extended TS-ORS Model Including Rating Context

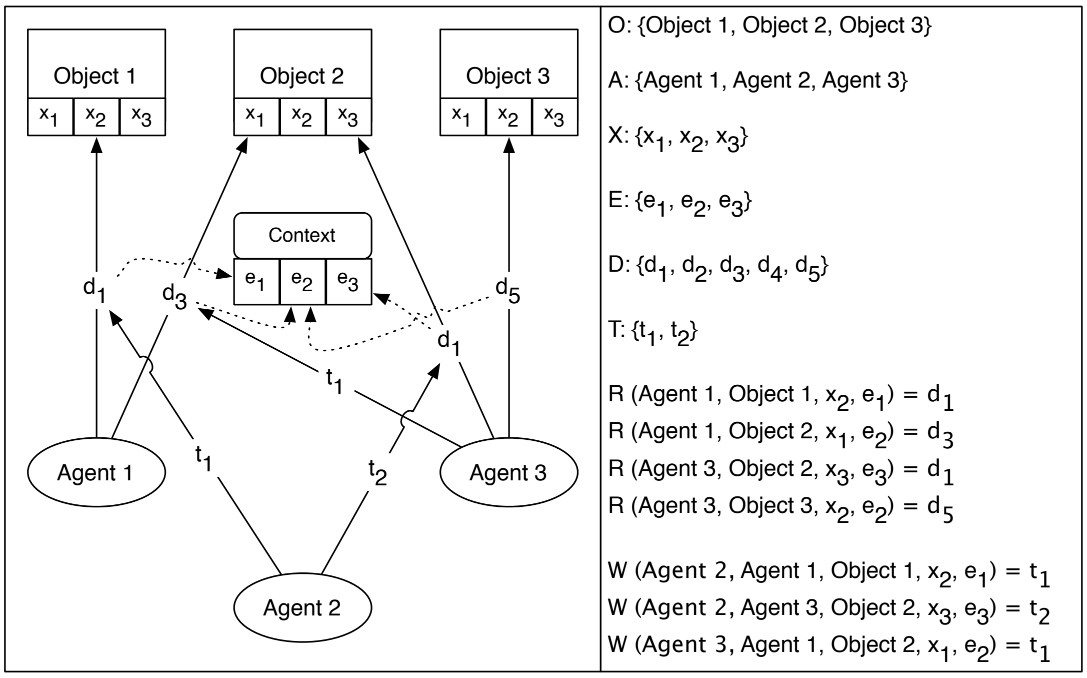
In order to formalize the evaluation context, our model can be extended and feature the following components:

- A set of objects $O : \{o_1, o_2, o_3, \ldots\}$ that can be rated.
- A set of agents $A : \{a_1, a_2, a_3, \ldots\}$ that can provide ratings and trust statements. Agents can be both human users or automatic agents.
- A set of object aspects $X : \{x_1, x_2, x_3, \ldots\}$.
- A set of evaluation contexts $E : \{e_1, e_2, e_3, \ldots\}$
- A set of possible values for ratings of objects $D : \{d_1, d_2, \ldots\}$.
- A set of possible values for trust ratings of agents $T : \{t_1, t_2, \ldots\}$.
- A partial function $R : A \times O \times X \times E \to D$. R corresponds to the rating of an agent on one certain aspect of an object in a specific evaluation context.
- A partial function $W : A \times A \times O \times X \times E \to T$, which corresponds to the trust of an agent in another agent for a specific aspect–object combination in a specific evaluation context.

A depiction of the extended TS-ORS model can be found in Figure 3.3.
If necessary, the domains of ontologies can be integrated into the model as presented in Section 3.3.1 by simply adding C, H_C, and L to the components found above.

3.4.2 Meta-trust Statements also Covering Evaluation Context

The meta-trust statements can be adopted by extending given meta-trust statements to also take evaluation contexts into account as shown in Table 3.3.



O: {Object 1, Object 2, Object 3}

A: {Agent 1, Agent 2, Agent 3}

X: $\{x_1, x_2, x_3\}$

E: $\{e_1, e_2, e_3\}$

D: $\{d_1, d_2, d_3, d_4, d_5\}$

T: $\{t_1, t_2\}$

R (Agent 1, Object 1, x_2, e_1) = d_1
R (Agent 1, Object 2, x_1, e_2) = d_3
R (Agent 3, Object 2, x_3, e_3) = d_1
R (Agent 3, Object 3, x_2, e_2) = d_5

W (Agent 2, Agent 1, Object 1, x_2, e_1) = t_1
W (Agent 2, Agent 3, Object 2, x_3, e_3) = t_2
W (Agent 3, Agent 1, Object 2, x_1, e_2) = t_1

Figure 3.3: A depiction of our TS-ORS model with an extension for including evaluation contexts.

Table 3.3: Allowed Meta-trust Statements Taking Into Account Evaluation Contexts.

Statement	**Signature**	**Explanation**
W_E	$A \times A \times O \times X \to T$	Statement on a specific aspect of a specific object for all evaluation contexts.
W_{O_E}	$A \times A \times X \times E \to T$	Statement on a specific aspect of all objects in a specific evaluation context (for arbitrary objects of O)
W_{X_E}	$A \times A \times O \times E \to T$	Statement on all aspects of a specific object in a specific evaluation context (for arbitrary aspects of X)
W_O	$A \times A \times X \to T$	Statement on a specific aspect of all objects for all evaluation contexts (for arbitrary objects of O)
W_X	$A \times A \times O \to T$	Statement on all aspects of a specific object for all evaluation contexts (for arbitrary aspects of X)
$W_{(OX)_E}$	$A \times A \times E \to T$	Statement on all aspects of all objects in a specific evaluation context (for arbitrary aspects of X and objects of O)
W_{OX}	$A \times A \to T$	Statement on all aspects of all objects in all evaluation contexts (for arbitrary aspects of X and objects of O)

Chapter 4

TS-ORS Algorithms and Trust Materialization

After presenting the TS-ORS model in Chapter 3, we now present the algorithms that make use of the data. In this chapter we show how the trust information in W and the meta-trust statements can be used to provide a ranking of reviews for a given aspect–object combination, and to compute an overall rating for objects (taking R into account).

The process starts with the materialization of meta-trust statements to normal trust statements described in Section 4.1. We then use the trust information to compute trust values for each user, which can be used to rank reviews (see Section 4.2). Based on the top ranked reviews (see Section 4.3), we can compute an overall rating of objects (see Section 4.4). An example of such computations with exemplary data is given in Chapter 5 Section 5.6. Possible extensions of the algorithms are presented in Sections 4.5, 4.6, and 4.7. Parts of this chapter are based on (Lewen and d'Aquin, 2010; Lewen *et al.*, 2006; Lewen, 2005; Sabou *et al.*, 2007).

4.1 Meta-trust Materialization

Since the meta-trust statements are not part of the model, we have to materialize them to single W statements before the trust computation. The materialization is based on our intuition that more specific trust statements (those covering a smaller scope) are more authoritative: $W \succ W_O \succ W_X \succ W_{OX}$ ("$\succ$" meaning more authoritative). The materialization is performed based on the above order, i.e., starting with all statements in the form W, then processing all statements in the form W_O, then W_X, and finally W_{OX}. For each of the meta-trust statements, their scope is checked and then the value of the statement is propagated to the object–aspect level. Existing trust information is not overwritten in this process. For example, if a meta-trust statement has been made for an object (W_X), it is checked which aspects of this object are not covered by trust statements yet, and the value of the meta-trust statement is used for these aspects. We refer to the final outcome of the materialization as W'. Using meta-trust statements, it is possible for example to distrust another user globally, but to trust him for a certain object o_n (since the statement on the object is more authoritative than the global statement, the user will be distrusted for all objects except for o_n).

4.1.1 Formal Meta-trust Materialization Algorithm

An operationalization of the meta-trust materialization can be found in Algorithm 1. We explain the algorithm in the following:

- First a partial function *trustprop* is created, which is initialized with the values from W.
- Then it is checked for which tuples (agent, agent, aspect) W_O is defined. For each tuple (agent, agent, aspect) for which W_O is defined, it is checked for all objects in the system, whether a trust statement in the form (agent, agent, object, aspect) is defined in the function *trustprop*. In case the tuple is defined in *trustprop* nothing is done. Otherwise, the value of the meta-trust statement for that (agent, agent, aspect) combination is taken, and *trustprop* (agent, agent, object, aspect) is set to the meta-trust value. This procedure is run until all tuples for which W_O is defined are processed, i.e., until the complete meta-trust for objects has been materialized.
- To propagate the meta-trust on properties, it is checked for which tuples (agent, agent, object) W_X is defined. For each tuple (agent, agent, object) for which W_X is defined, it is checked for all aspects in the system, whether a trust statement in the form (agent, agent, object, aspect) is defined in the function *trustprop*. In case the tuple is defined in *trustprop* nothing is done. Otherwise, the value of the meta-trust statement for that (agent, agent, object) combination is taken, and *trustprop* (agent, agent, object, aspect) is set to the meta-trust value. This procedure runs until all tuples for which W_X is defined are processed, i.e., until the complete meta-trust for aspects has been materialized.
- To propagate global meta-trust statements, it is checked for which tuples (agent, agent) W_{OX} is defined. For each tuple (agent, agent) for which W_{OX} is defined, it is checked for all objects-aspect combinations in the system, whether a trust statement in the form (agent, agent, object, aspect) is defined in the function *trustprop*. In case the tuple is defined in *trustprop* nothing is done. Otherwise, the value of the meta-trust statement for that (agent, agent) combination is taken, and *trustprop* (agent, agent, object, aspect) is set to the meta-trust value. This procedure is run until all tuples for which W_{OX} is defined are processed, i.e., until the complete global meta-trust has been materialized.

```
Input: $W, W_X, W_O, W_{OX}, B_R$
Output: $W'$
Create partial function $trustprop := W$
foreach $(a_i, a_j, x_k) \in A \times A \times X$ such that $W_O(a_i, a_j, x_k)$ is defined do
    foreach $o_n \in O$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_O(a_i, a_j, x_k)$
        end
    end
end
foreach $(a_i, a_j, o_n) \in A \times A \times O$ such that $W_X(a_i, a_j, o_n)$ is defined do
    foreach $x_k \in X$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_X(a_i, a_j, o_n)$
        end
    end
end
foreach $(a_i, a_j) \in A \times A$ such that $W_{OX}(a_i, a_j)$ is defined do
    foreach $o_n \in O, x_k \in X$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_{OX}(a_i, a_j)$
        end
    end
end
Create partial function $W' := trustprop$
```

Algorithm 1: Meta-trust Materialization

4.2 Computing Trust Values for Ranking

After the meta-trust has been materialized, all trust statements are in the form of W, and trust ranks can be computed. Note that in contrast to the basic ORS (see Section 2.6), we compute individual trust relationships for every aspect of every object (every $o_n x_k$ combination for $o_n \in O, x_k \in X$). In the following, whenever we use the subscript $o_n x_k$ for matrices or ranks, it means that they contain the data relevant to this $o_n x_k$ combination.

For each $o_n x_k$ combination, we define two matrices storing trust and distrust. The trust matrix $\mathbf{T}$ has entries $t_{ij} \in \{0, 1\}$. If $t_{ij} = 1$, user a_i trusts a_j according to W', otherwise no information is available. Analogously, the distrust matrix $\mathbf{D}$ has entries $d_{ij} \in \{0, 1\}$ capturing the distrust (we will explain how to work with continuous t_{ij}

and d_{ij} values in Section 4.5).

Given **T** and **D**, we can compute *GlobalTrustRank* (a relabeled Page-Rank (Page *et al.*, 1998), see Equation 4.1) and *GlobalDistrustRank* (from (Guha, 2004), see Equation 4.2). *GlobalTrustRank* provides a trust value for each user based on how many users trust him. *GlobalDistrustRank* provides a distrust value for each user, based on how many users distrust him. Using Algorithm 2, which is based on (Guha *et al.*, 2004), the local trust matrix **F** and the interpretation matrix **I** can be computed. **F** provides personalized user-to-user trust scores, and **I** provides, if possible, an interpretation of the scores in **F** as trust or distrust. We present the algorithms in the following, while examples can be found in Chapter 5

4.2.1 Global Trust

$$GlobalTrustRank_{i+1}(a_u) = (1-d) + d \cdot \left(\sum\nolimits_{v \in T_v} \frac{GlobalTrustRank_i(v)}{N_v} \right) \tag{4.1}$$

where a_u is the user whose *GlobalTrustRank* is computed, T_v is the set of users trusting a_u, $v \in T_v$ is a user from T_v trusting a_u, N_v is the total number of users user v trusts, d is a damping factor between 0 and 1,[1] and i is the number of iterations the algorithm has run. The *GlobalTrustRank*s can be initialized as 1. Intuitively speaking, *GlobalTrustRank* assigns trust to users based on how many other users trust them and how trusted the users trusting them are. The algorithm runs iteratively, normally until the ranks converge, i.e., until a new iteration does not change the computed *GlobalTrustRank*s anymore. Another possibility is to run the algorithm for a fixed number of iterations. The algorithm computes in each iteration the *GlobalTrustRank* of a user as the sum of the normalized *GlobalTrustRank*s of the people who trust that user. Normalized means that the *GlobalTrustRank* of a user is divided by the total number of trust statements he has made. Because of this, the importance of a trust statement is diminished every time another trust statement is made. Based on the *GlobalTrustRank*s, the *GlobalDistrustRank*s can be computed.

$$GlobalDistrustRank(a_u) = \sum\nolimits_{v \in B_v} \frac{GlobalTrustRank(v)}{N_v} \tag{4.2}$$

where a_u is the user whose *GlobalDistrustRank* is computed, B_v is the set of users distrusting a_u, $v \in B_v$ is a user from B_v distrusting a_u, N_v is the total number of users user v distrusts. *GlobalDistrustRank* is taking into account who distrusts a user and how high the *GlobalTrustRank* of the distrusting users are. The algorithm only runs one iteration, and is computing the *GlobalDistrustRank* of a user by summing up the normalized *GlobalTrustRank*s of users who distrust him. Normalized in this case again

[1]Based on (Page *et al.*, 1998), it is usually set to 0.85 for fast convergence of ranks.

means that the *GlobalTrustRank* of a user is divided by the total number of distrust statements he has made. The greater the number of distrust statements, the more the effect of each single distrust statement is diminished.

4.2.2 Local Trust

We base the computation of personalized user-to-user trust scores (see Algorithm 2) on the work from Guha et al (Guha *et al.*, 2004), who investigated trust and distrust propagation in a WOT based on real world data. The algorithm uses the 4 different kinds of atomic trust propagation shown in Table 4.1. It employs a combination of these trust propagation techniques to propagate trust within the WOT. Distrust is only propagated 1 step (statements from distrusted users are discounted), since the semantics of distrust propagation are not clear (Guha *et al.*, 2004) (see also Section 2.5.2).

The algorithm (see Algorithm 2) works by combining the 4 different atomic propagation steps to the propagation matrix **C** using a linear combination. Each atomic propagation step is weighted with a parameter β, which is used to shift importance between the different propagation steps.

The final local trust matrix **F** is computed by adding up the results of each propagation step until trust has been propagated K steps. Within each propagation step, first a propagation matrix for the correct number of propagation steps is computed. If trust is propagated 1 step, **C** remains unchanged. For 2 steps, it is squared, for 3 steps cubed, and so on. In general, $\mathbf{C}^{(k)}$ represents the propagation matrix for a k step propagation.

Then this propagation matrix is multiplied with the difference of the trust matrix **T** and the distrust matrix **D**. This multiplication represents the actual propagation of trust and distrust. The resulting matrix is then discounted with a parameter γ, which represents the decay of trust over propagation steps (see Equation 2.2). Once the matrix has been discounted with γ, it is added to the current **F** matrix. Since **F** is the result of several additions, the trust scores within **F** keep on growing, they are not normalized. It is not necessary for values f_{ij} to be between 0 and 1.

After **F** is computed, its values f_{ij} have to be interpreted as either trust or distrust (if possible). The idea of majority rounding (Guha *et al.*, 2004) is that values that result from the propagation process can be interpreted by looking at values for which the interpretation is known (the f_{ij} for which t_{ij} or d_{ij} is different from 0). To initialize the interpretation matrix **I**, the difference between **T** and **D** is computed. The resulting matrix **I** will have the entry 1 for all i_{ij} for which $t_{ij} = 1$, and $i_{ij} = -1$ for all entries for which $d_{ij} = 1$. In the interpretation matrix **I**, 1 is interpreted as trust, -1 as distrust, and 0 as unknown.

To interpret **F**, for each row (each row represents the trust scores from one user to all other users), a sequence is computed that maps all users to exactly one element in the sequence, and furthermore makes sure that the elements in the sequence are ordered ascending according to the value in **F**. Basically the sequence gives us a way to sort a row of **F** while keeping the correspondence to **I**. The interpretation as such is then realized by checking, for each unknown entry in the interpretation matrix, what the interpretation of the neighboring entries of f_{ij} are in the ordered sequence. In case they are both trust, the value is interpreted as trust. In case they are both distrust, the interpretation is also distrust. In case the interpretation of surrounding values differs, it is checked which value is closer to the value that is being interpreted, and the interpretation of that value is taken.

After all values in a row of **F** have been interpreted, the interpretations of entries where $f_{ij} = 0$ are set to unknown. A value of 0 in the final trust matrix means that the user-to-user combination could not be computed, so it should also not be interpreted.

4.3 Ranking Reviews at the Aspect Level of an Object

When a user requests the reviews for an object–aspect combination, the reviews for that combination have to be ranked by the TS-ORS to be provided in a user-specific order. If a user can be identified, we can base the user-specific ranking on the *local* trust information. This ranking is also influenced by a parameter $\alpha \in [0, 1]$ that the user can provide to combine global trust ($\mathit{GlobalTrustRank}(\mathit{GTR})$) and distrust ($\mathit{GlobalDistrustRank}(\mathit{GDR})$) to a $\mathit{GlobalCombinedRank}(\mathit{GCR})$. The higher α is, the more emphasis is put on the GTR. In case a user can be identified, we use Algorithm 3, in the other case Algorithm 4.

Algorithm 3 ranks reviews based on the information who is trusted and who is distrusted, and the local trust value from **F**. Given we want to get the ranking for user a_i, we first identify all existing reviews for the object-aspect combination which should be ranked. Then, for all reviews, we assign a triple consisting of the interpretation of the local trust score for the reviewer, the local trust score for the reviewer, and the *GlobalCombinedRank* of the reviewer. The *GlobalCombinedRank* is computed as a linear combination of $\mathit{GlobalTrustRank}(\mathit{GTR})$ and $\mathit{GlobalDistrustRank}(\mathit{GDR})$. Once all reviews are assigned triples, the reviews can be sorted. Based on the triples, reviews are sorted in descending lexicographic order (meaning columns are sorted descending, starting with i_{ij} and then considering f_{ij} to sort entries where the i_{ij} value is identical, and then considering $GCR(a_j)$ if both values for i_{ij} and f_{ij} are identical). The result of the ranking process is a ranking order which starts with the review from the reviewer with the highest local trust score which was interpreted as trust, and goes on with all reviewers whose local trust scores were identified as trust in a descending order. When all reviewers which have been interpreted as trusted in **I** have been ranked,

```
Input: Trust Matrix $\mathbf{T}$, Distrust Matrix $\mathbf{D}$
Output: Local Trust Matrix $\mathbf{F}$, Interpretation Matrix $\mathbf{I}$
$\mathbf{C} := \beta_1 \cdot \mathbf{T} + \beta_2 \cdot \mathbf{T}^\intercal\mathbf{T} + \beta_3 \cdot \mathbf{T}^\intercal + \beta_4 \cdot \mathbf{T}\mathbf{T}^\intercal$
$\mathbf{F} := \sum_{k=1}^{K} \gamma^k \cdot (\mathbf{C}^{(k)} \cdot (\mathbf{T} - \mathbf{D}))$
Initialize $\mathbf{I}$:=$\mathbf{T} - \mathbf{D}$
foreach $j \in A$ do
    Compute a sequence $a(1), a(2), \ldots, a(|A|)$ such that:
    – for all $a \in A$ there is exactly one $i \in \{1, \ldots, |A|\}$ such that $a(i) = a$
    – for all $i \in \{1, \ldots, |A| - 1\}$ we find that $f_{ja(i)} \leqslant f_{ja(i+1)}$
    To simplify notation, we use $i_{ja(0)}$ and $i_{ja(|A|+1)}$ to denote 0. Small letters
    with subscripts denote entries in the matrices.
    repeat
        foreach $m \in A$ do
            if $i_{ja(m)} = 0$ then
                $i_{ja(m)} := 1$
                if $\Big((i_{ja(m-1)} + i_{ja(m+1)} \leq -1) \vee \big((i_{ja(m-1)} = -1 \wedge i_{ja(m+1)} = 1)$
                $\wedge ((f_{ja(m+1)} - f_{ja(m)}) > (f_{ja(m)} - f_{ja(m-1)}))\big) \vee \big((i_{ja(m-1)} = 1$
                $\wedge i_{ja(m+1)} = -1) \wedge ((f_{ja(m+1)} - f_{ja(m)}) < (f_{ja(m)} - f_{ja(m-1)}))\big)\Big)$
                then
                    $i_{ja(m)} := -1$
                end
                if $i_{ja(m-1)} = i_{ja(m+1)} = 0$ then
                    $i_{ja(m)} := 0$
                end
            end
        end
    until no further changes occur in $\boldsymbol{I}$ ;
    foreach $m$ do
        if $f_{ja(m)} = t_{ja(m)} = d_{ja(m)} = 0$ then
            $i_{ja(m)} := 0$
        end
    end
end
```

Algorithm 2: Local Trust Computation

the following ranks are filled with reviewers for which no local trust score could be computed. Their reviews are sorted based on the *GlobalCombinedRank* in descending order. Ultimately, reviews from reviewers whose local trust score was interpreted as

Table 4.1: Atomic Trust Propagation according to Guha et al.

Propagation	**Operator**	**Description**
Direct Propagation	$\mathbf{T}$	If A trusts B, someone trusted by B should also be trusted by A
Co-Citation	$\mathbf{T}^{\intercal} \cdot \mathbf{T}$	If A trusts B and C, someone trusting C should also trust B
Transpose Trust	$\mathbf{T}^{\intercal}$	If A trusts B, someone trusting B should also trust A
Trust Coupling	$\mathbf{T} \cdot \mathbf{T}^{\intercal}$	If A and B trust C, someone trusting A should also trust B

```
Input: $o_n \in O, x_k \in X, GTR_{o_n x_k}, GDR_{o_n x_k}, \mathbf{F}_{o_n x_k}, \mathbf{I}_{o_n x_k}, \alpha_i \in [0,1], a_i \in A$
Output: Sorted Reviews
foreach $(a_j, o_n, x_k) \in B_R$ do
    $GCR(a_j) := \alpha_i \cdot GTR_{o_n x_k}(a_j) - ((1-\alpha_i) \cdot GDR_{o_n x_k}(a_j))$
    Assign to the review $(a_j, o_n, x_k) \in B_R$ a triple $(i_{ij}, f_{ij}, GCR(a_j))$
end
Based on these triples, reviews are sorted in descending lexicographic order
(meaning columns are sorted descending, starting with $i_{ij}$ and then considering
$f_{ij}$ to sort entries where the $i_{ij}$ value is identical, and then considering $GCR(a_j)$
if both values for $i_{ij}$ and $f_{ij}$ are identical)
```

Algorithm 3: Review Ranking if the User Can be Identified

distrust are sorted from the least distrusted reviewer (the one with the highest local trust score) to the most distrusted reviewer (the one with the lowest local trust score). Since the results take into account a user's WOT, they are personalized.

```
Input: $o_n \in O, x_k \in X, GTR_{o_n x_k}, GDR_{o_n x_k}, \alpha \in [0,1]$
Output: Sorted Reviews
foreach $(a_j, o_n, x_k) \in B_R$ do
    $GCR(a_j) := \alpha \cdot GTR_{o_n x_k}(a_j) - ((1-\alpha) \cdot GDR_{o_n x_k}, (a_j))$
    Assign to the review $(a_j, o_n, x_k) \in B_R$ a value $GCR(a_j)$
end
Based on these values, reviews are sorted starting with the highest value.
```

Algorithm 4: Review Ranking if the User Cannot be Identified

Algorithm 4 can only rank based on the GCR values, since the user is unknown.

Thus its ranking cannot be personalized. The algorithm works by first identifying all existing reviews for the object-aspect combination which should be ranked. Then, for all reviews, the *GlobalCombinedRank* value of its reviewer is assigned. The *GlobalCombinedRank* is computed as a linear combination of *GlobalTrustRank*(GTR) and *GlobalDistrustRank*(GDR). Once all reviews are assigned the values, the reviews can be sorted in a descending order according to the *GlobalCombinedRank* values. The result of the ranking process is a ranking of reviews according to the *GlobalCombinedRank* of the reviewers who wrote them. Depending on how the parameter α is chosen, this ranking can change. Since the user requesting the ranking is not know to the system, the results cannot be based on the WOT, and thus they are the same for all anonymous users, namely based on global trust.

4.4 Computing an Overall Rating of an Object

Each object in the TS-ORS has aspects $x_k \in X$ it can be reviewed on. The user can choose how to weigh each aspect for computing the overall rating of an object by specifying a weight $\mu_k \geq 0$ for each aspect $x_k \in X$. In case no rating exists for an aspect, we distribute its weight evenly to the remaining aspects. The user can also decide on how many top-ranked ratings per aspect the computation is based. This is done with the parameter $N \geq 1$. $N = 3$ means, for example, that the top-3-ranked ratings are considered. Another parameter $\nu \in (0, 1]$ can be specified that determines how the top-N-ranked ratings are combined. The closer ν is to 0, the more emphasis is put on the top-ranked ratings, if $\nu = 1$ a linear combination is computed.

We use Algorithm 5 to compute the overall rating. The ranking of reviews is determined by Algorithm 3 or 4, based on whether a user is identifiable or not. The algorithm first processes the aspects one after another. For each aspect, it is first made sure that parameter N is not greater than the number of ranked reviews. In case no reviews exist for an aspect, the rating for this aspect and the weight for the aspect are set to 0. In case ratings exist, the top N ratings are combined using a weighted combination based on parameters N and ν. Once the combined top-N-ratings exist for all aspects, the weights μ are normalized and, in the process, weights for aspects which do not have ratings are redistributed. In the final step, the overall rating of the object is computed as a weighted linear combination of the combined top-N-ratings for all aspects, taking into account the user's preferences as defined via the μ_k weights.

In contrast to Guha's ORS model, it is now possible to compose an overall rating using ratings on different aspects and based on topic-specific trust statements.

```
Input: $o_n \in O$, ranked sets of reviews
       $B_{o_n x_k} = \{(a_{j1}, o_n, x_k), \cdots, (a_{jm}, o_n, x_k)\} \subseteq B_R$ for each $x_k \in X$,
       $\nu \in (0,1]$, $N \geqslant 1$, $\mu_k$ for each $x_k \in X$
Output: Rating $D_{o_n}$
For a given $o_n x_k$ combination, we use the notation $(a_{ji}, o_n, x_k) \in B_{o_n x_k}$ to refer to
the $i$-th ranked result.
foreach $x_k \in X$ do
    $N := \min(N, |B_{o_n x_k}|)$
    if $N = 0$ then
        $D_{o_n x_k} := 0$
        $\mu_k := 0$
    else
        $D_{o_n x_k} := \sum_{i=1}^{N} \Big( (\nu^i / \sum_{s=1}^{N} \nu^s) \cdot R(a_{ji}, o_n, x_k) \Big)$
    end
end
foreach $k = 1, \cdots, |X|$ with $\mu_k \neq 0$ do
    $\mu_k := \mu_k / \sum_{l=1}^{|X|} \mu_l$
end
$D_{o_n} := \sum_{x_k \in X} \mu_k \cdot D_{o_n x_k}$
```

Algorithm 5: Computation of an Overall Rating

4.5 Extending the Algorithms for Continuous Trust Values

Even though many systems facilitate a user's trust decision to trust or distrust, it is possible to allow different trust values. The only important assumption is that trust and distrust can be distinguished. In this section we present how the algorithms can be extended to handle more precise trust and distrust statements. In this example, we assume that both trust and distrust can be expressed on a scale from 1 to 10.

Algorithm 1 can remain unchanged, since no assumptions are made on the trust scale. The trust or distrust value given is just used as such during the materialization process.

In contrast to the regular version, the trust matrix **T** now has entries $t_{ij} \in [0,1]$. We compute the entries t_{ij} by dividing the respective trust values by 10, because we have 10 values on the scale. As a result, the entries in **T** can now express the level of trust, and not only the existence. Analogously, the distrust matrix **D** has now entries $d_{ij} \in [0,1]$ capturing the distrust. We compute the entries d_{ij} by dividing the respective trust values by 10. The computation of *GlobalTrustRank* and *GlobalDistrustRank* have to be modified to make use of the more fine granular trust values. The adapted versions can be found in Equation 4.3 and Equation 4.4.

$$WeightedGTR_{i+1}(a_u) = (1-d) + d \cdot \left(\sum\nolimits_{v \in T_v} \frac{t_{vu} \cdot WeightedGTR_i(v)}{N_v} \right) \tag{4.3}$$

where a_u is the user whose *WeightedGTR* is computed, T_v is the set of users trusting a_u, $v \in T_v$ is a user from T_v trusting a_u, t_{vu} is the entry from $\mathbf{T}$ with the trust assigned by user v to user u, N_v is the sum of all of user v's trust statements (the sum of user v's row in $\mathbf{T}$), d is a damping factor between 0 and 1,[2] and i is the number of iterations the algorithm has run. The *WeightedGTR*s can be initialized as 1. By introducing the trust value into the computation, the strength of a trust statement is taken into account when computing the global trust values. Based on the *WeightedGTR*s, the *WeightedGlobalDistrustRank*s can be computed.

$$WeightedGlobalDistrustRank(a_u) = \sum\nolimits_{v \in B_v} \frac{d_{vu} \cdot WeightedGTR(v)}{N_v} \tag{4.4}$$

where a_u is the user whose *WeightedGlobalDistrustRank* is computed, B_v is the set of users distrusting a_u, $v \in B_v$ is a user from B_v distrusting a_u, d_{vu} is the entry from $\mathbf{D}$ with the distrust assigned by user v in user u, N_v is the sum of all of user v's distrust statements (the sum of user v's row in $\mathbf{D}$).

For Algorithm 2, the only change would be in the initialization of $\mathbf{I}$. Now, i_{ij} is set to 1 if $t_{ij} > 0$ or to -1 if $d_{ij} > 0$. In the rest of cases, i.e., if no trust or distrust for the i,j combination exists, the interpretation is kept as 0.

With these adaptations to the algorithms, the system can deal with more detailed trust and distrust statements.

4.6 Extending the Algorithms to Deal with Domains

The only algorithm that has to be adapted to enable use of domains as described in Section 3.3.1 is the meta-trust materialization. We have to include the additional meta-trust statements W_{O_C}, W_{X_C} into the meta-trust materialization algorithm. The order of authoritativeness of meta-trust statements is now $W \succ W_{O_C} \succ W_O \succ W_X \succ W_{X_C} \succ W_{OX}$. The adapted meta-trust materialization algorithm can be found in Algorithm 6. Once all meta-trust has been materialized, the rest of the computation steps is as in the standard case presented in this chapter.

[2] Based on (Page *et al.*, 1998), it is usually set to 0.85 for fast convergence of ranks.

4.7 Extending the Algorithms to Deal with Evaluation Contexts

The first algorithm that has to be adapted to enable the use of evaluation contexts as described in Section 3.4.1 is again the meta-trust materialization. We have to include the additional meta-trust statements $W_E, W_{X_E}, W_{O_E}, W_{(OX)_E}$ into the meta-trust materialization algorithm. The order of authoritativeness of meta-trust statements is now $W \succ W_E \succ W_{O_E} \succ W_{X_E} \succ W_O \succ W_X \succ W_{(OX)_E} \succ W_{OX}$. The adapted meta-trust materialization algorithm can be found in Algorithms 7 and 8.

For the rest of the computations, the algorithms can remain unchanged, but the trust computations have to be performed for each $o_n x_k$ combination for each evaluation context $e_m \in E$. During runtime, a user would then specify which evaluation context should be used, and only the relevant data is taken into account for the ranking and computation of overall ratings. It can be seen as selecting one value from E, and then working with a view on the data for that value.

```
Input: $W, W_X, W_O, W_{OX}, W_{O_C}, W_{X_C}, B_R, L, H_C$
Output: $W'$
Create partial function $trustprop := W$
foreach $(a_i, a_j, c_m, x_k) \in A \times A \times C \times X$ such that $W_{O_C}(a_i, a_j, c_m, x_k)$ is defined
do
    foreach $o_n \in L^{-1}(H_C^{-1}(c_m))$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_{O:C}(a_i, a_j, c_m, x_k)$
        end
    end
end
foreach $(a_i, a_j, x_k) \in A \times A \times X$ such that $W_O(a_i, a_j, x_k)$ is defined do
    foreach $o_n \in O$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_O(a_i, a_j, x_k)$
        end
    end
end
foreach $(a_i, a_j, o_n) \in A \times A \times O$ such that $W_X(a_i, a_j, o_n)$ is defined do
    foreach $x_k \in X$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_X(a_i, a_j, o_n)$
        end
    end
end
foreach $(a_i, a_j, c_m) \in A \times A \times C$ such that $W_{X_C}(a_i, a_j, c_m)$ is defined do
    foreach $o_n \in L^{-1}(H_C^{-1}(c_m)), x_k \in X$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_{X_C}(a_i, a_j, c_m)$
        end
    end
end
foreach $(a_i, a_j) \in A \times A$ such that $W_{OX}(a_i, a_j)$ is defined do
    foreach $o_n \in O, x_k \in X$ do
        if $trustprop(a_i, a_j, o_n, x_k)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k) := W_{OX}(a_i, a_j)$
        end
    end
end
Create partial function $W' := trustprop$
```

Algorithm 6: Meta-trust Materialization including Domains

```
Input: $W, W_E, W_{O_E}, W_{X_E}, W_O, W_X, W_{(OX)_E}, W_{OX}, B_R$
Output: $W'$
Create partial function $trustprop := W$
foreach $(a_i, a_j, o_n, x_k) \in A \times A \times O \times X$ such that $W_E(a_i, a_j, o_n, x_k)$ is defined do
    foreach $e_m \in E$ do
        if $trustprop(a_i, a_j, o_n, x_k, e_m)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k, e_n) := W_E(a_i, a_j, o_n, x_k)$
        end
    end
end
foreach $(a_i, a_j, x_k, e_m) \in A \times A \times X \times E$ such that $W_{O_E}(a_i, a_j, x_k, e_m)$ is defined
do
    foreach $o_n \in O$ do
        if $trustprop(a_i, a_j, o_n, x_k, e_m)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k, e_n) := W_{O_E}(a_i, a_j, x_k, e_m)$
        end
    end
end
foreach $(a_i, a_j, o_n, e_m) \in A \times A \times O \times E$ such that $W_{X_E}(a_i, a_j, o_n, e_m)$ is defined
do
    foreach $x_k \in X$ do
        if $trustprop(a_i, a_j, o_n, x_k, e_m)$ is undefined then
            $trustprop(a_i, a_j, o_n, x_k, e_n) := W_{X_E}(a_i, a_j, o_n, e_m)$
        end
    end
end
⋮
```

Algorithm 7: Meta-trust Materialization including Evaluation Contexts Part 1

```
⋮
foreach $(a_i, a_j, x_k) \in A \times A \times X$ such that $W_O(a_i, a_j, x_k)$ is defined do
    foreach $o_n \in O$ do
        if trustprop$(a_i, a_j, o_n, x_k)$ is undefined then
            trustprop$(a_i, a_j, o_n, x_k) := W_O(a_i, a_j, x_k)$
        end
    end
end
foreach $(a_i, a_j, o_n) \in A \times A \times O$ such that $W_X(a_i, a_j, o_n)$ is defined do
    foreach $x_k \in X$ do
        if trustprop$(a_i, a_j, o_n, x_k)$ is undefined then
            trustprop$(a_i, a_j, o_n, x_k) := W_X(a_i, a_j, o_n)$
        end
    end
end
foreach $(a_i, a_j, e_m) \in A \times A \times E$ such that $W_{(OX)_E}(a_i, a_j, e_m)$ is defined do
    foreach $o_n\ inO, x_k \in X, e_m \in E$ do
        if trustprop$(a_i, a_j, o_n, x_k, e_m)$ is undefined then
            trustprop$(a_i, a_j, o_n, x_k, e_n) := W_{(OX)_E}(a_i, a_j, e_m)$
        end
    end
end
foreach $(a_i, a_j) \in A \times A$ such that $W_{OX}(a_i, a_j)$ is defined do
    foreach $o_n \in O, x_k \in X$ do
        if trustprop$(a_i, a_j, o_n, x_k)$ is undefined then
            trustprop$(a_i, a_j, o_n, x_k) := W_{OX}(a_i, a_j)$
        end
    end
end
Create partial function $W' :=$ trustprop
```

Algorithm 8: Meta-trust Materialization including Evaluation Contexts Part 2

Chapter 5

Adaptation of TS-ORS for Ontology Reuse and Example Calculations

In this chapter, we start by explaining the history of using Open Rating Systems for ontology evaluation in Section 5.1. We then show how the TS-ORS model can be initialized for a use case in Section 5.2. Default parameters for the TS-ORS algorithms are provided in Section 5.3. Section 5.4 explains how the TS-ORS can be employed to facilitate ontology reuse. The incorporation of automatic evaluation techniques into the TS-ORS is discussed in Section 5.5. The chapter concludes with an exemplary computation of trust values and rankings in Section 5.6. Parts of this chapter are based on (Lewen and d'Aquin, 2010; Lewen *et al.*, 2006; Lewen, 2005; Sabou *et al.*, 2007).

5.1 History of Open Rating Systems for Ontology Evaluation

As already stated in Section 2.6, the idea of employing user ratings for ontology evaluation has first been proposed by Noy et al. (Noy *et al.*, 2005) in 2005. Guha's ORS was employed in the ontology portal Knowledge Zone (Supekar *et al.*, 2007) for the biomedical domain. Knowledge Zone has since been taken off the Web and been replaced by the NCBO's Bioportal, which also plans to include a user rating system of ontologies (Noy *et al.*, 2008). After our initial idea on how to extend an Open Rating System for enabling topic-specific trust has been published in (Lewen *et al.*, 2006), it has been picked up by Hartmann et al. and included in their proposed Generic Ontology Repository Framework architecture (Hartmann *et al.*, 2009). They suggest that every ontology repository should include the rating functionality offered by our Topic-Specific Trust Open Rating System. TS-ORS has been implemented and included inside the Cupboard system, which we will present in detail in Section 9.

5.2 Initialization of TS-ORS Model

Whenever the generic TS-ORS Model is employed in a certain context, the model has to be instantiated based on the requirements imposed by the desired use case. In the following, we discuss the adaptations needed for use of the TS-ORS for ontology rating and ranking. In the context of ontology evaluation and rating, the set of objects O can contain complete ontologies, parts of an ontology (ontology modules), or even URIs of classes. Internally, an implementation will likely just store an URI which can be used to access the object, and an ID for internal reference. So when a rating is made, we assume it covers all axioms that can be found under the URI of the rated object. While it might be possible to develop a user interface which allows the users to give a rating on only a selected part of the object, i.e., only some axioms of all axioms of the ontology, we follow a different approach. Whenever a user feels that only parts of the ontology should be rated, the user should extract these parts (for example using state of the art modularization techniques (Pathak *et al.*, 2009)) and upload them in whatever form deemed most useful to the system. Then the uploaded subset of the original ontology can be rated. This way, it is always clear that a displayed rating score is valid for the complete ontology, and not only for parts of it. Since the Linked Data Initiative is also interested in promoting the best URIs for certain entities, there is no restriction on how big an ontology in the system has to be. It is thus also possible to upload an ontology only consisting of one concept with its URI, and then rate it.

Initializing the aspects in X is a lot harder when the objects in the TS-ORS are heterogeneous in nature. Imagine a vendor with a huge variety of products. It is likely that there is no common set of aspects all products share. When focusing on ontology rating, all objects in the system are some kind of ontology. They all share the same aspects. How the aspects are instantiated is up to the provider of the TS-ORS. As mentioned in Section 2.3, there are different works identifying aspects of ontologies that can be evaluated. In our TS-ORS implementation within Cupboard, we instantiate the aspects X based on Gangemi's work on ontology evaluation (Gangemi *et al.*, 2006), using: reusability, correctness, complexity, domain coverage, and modeling.

The agents in A are the registered users of the system. Both, users participating actively in uploading and reviewing, and users only expressing trust in other users to receive a better personalized ontology ranking.

The possible rating values in D in Cupboard are 1 star, 2 stars, 3 stars, 4 stars and 5 stars, and we assume the distance between the star ratings is the same. That means, the perceived difference between 1 star and 2 stars is the same as the perceived distance between 4 stars and 5 stars. Since for the algorithms the number of elements in D does not matter, it is easy to use a different rating scale, for example a 7 point rating scale, or introduce half star ratings in the 5 star rating scale. We chose to use the 5 star rating scale, because it has become the most used rating scale on the Web, and people are likely familiar with it, for example from Amazon.com, iTunes, or

Epinions.[1]

For T, we chose $T = \{trust, distrust\}$, since most other rating sites also refrain from using more complicated trust scores. In case a more detailed trust scale is needed, the TS-ORS algorithms can be extended as detailed in Section 4.5.

The rating and trust functions R and W are realized as database tables which store the relevant data.

5.3 Initialization of the TS-ORS Algorithms with Default Parameters

The algorithms presented in Chapter 4 are highly customizable because they contain many parameters that can be changed to achieve different results. We are aware that providing highly customizable algorithms with many different parameters can confuse users, since knowing which parameter has which effect on the result is not trivial. It is important to note that many parameters have to be chosen only once when instantiating the system, and thereafter are normally not changed. Moreover, providing reasonable default values allows users to interact with the system without providing their own values, while still enabling expert users to take full advantage of the flexibility of the system.

In the following we explain the parameters and provide default values we use for Cupboard:

- Parameter d from Equation 4.1 defines how big the minimal *GlobalTrustRank* of each user is. The parameter was introduced in the original PageRank algorithm to represent the probability that a surfer would randomly jump to another page, not following the link structure. The parameter is important, since it ensures that trust sinks cannot acquire all the trust. Based on the best practices reported by (Page *et al.*, 1998), we also use $d = 0.85$ as default, since it ensures a fast convergence of ranks.

- Parameters β_1, β_2, β_3, β_4 from Algorithm 2 are used to compute the propagation matrix $\mathbf{C}$. Each parameter is the weight for one of the four basic propagation types (see Table 4.1). β_1 weighs direct propagation, β_2 co-citation, β_3 transpose trust, and β_4 trust coupling. Based on the results of the user study in (Guha *et al.*, 2004) we also use $\beta_1 = 0.4$, $\beta_2 = 0.4$, $\beta_3 = 0.1$, $\beta_4 = 0.1$ in Cupboard.

- Parameter γ from Algorithm 2 is the decay factor for trust. The lower γ is, the more trust is lost with each propagation step. Also based on (Guha *et al.*, 2004) we use $\gamma = 0.9$.

[1] `http://www.epinions.com`, last checked on 24.11.2010

- Parameter K from Algorithm 2 sets the number of propagation steps. The higher K is, the more steps trust is propagated in the Web of Trust. For K it is important to note that each increase of K leads to at least 2 more matrix multiplications, making it cost intensive to set K too high. If it is set too low, on the other hand, the trust matrix might still be sparse. Also the decay factor has to be considered when choosing K. For large K, the impact of the last propagation steps is diminished considerably by γ^k. In Cupboard, we decided to propagate trust 7 steps ($K = 7$ based on the idea of 6 degrees of separation (Milgram, 1967)).

- Parameter α from Algorithms 3 and 4 defines how *GlobalTrustRank* can be combined with *GlobalDistrustRank* to compute the *GlobalCombinedRank*. The bigger α is, the more emphasis is put on *GlobalTrustRank*. In case that $\alpha = 1$, *GlobalDistrustRank* is discarded completely, while for $\alpha = 0$, *GlobalTrustRank* is not taken into account. For users who do not specify this value, we use $\alpha = 0.7$ as default value.

- With parameter N from Algorithm 5 the user can choose on how many of the top-N-ranked reviews per aspect the overall rating should be based. In case there are few reviews in the system, or we know that there are only very few of good quality, it is advisable to set $N = 1$, that is, only taking the top review for each aspect into account. In Cupboard we set $N = 1$ if the user does not specify N.

- With parameter ν from Algorithm 5 the user can choose how the top-N-ranked ratings should be combined towards an overall rating. The closer ν is to 0, the more emphasis is put on the top-ranked rating, while $\nu = 1$ computes a linear combination. In Cupboard we set the value to $\nu = 0.80$ if not specified differently, even though the default parameter $N = 1$ would not require combining reviews. Nevertheless we provide a default value, since the user might provide a different N value which would then require a value for ν.

- With parameter μ_k from Algorithm 5, each ontology aspect can be given a weight. The weights for the different aspects are an important means by which the user can personalize the algorithm. Depending on the use case, a user might want to only focus on reusability, leaving the other aspects aside. This can be achieved by setting the respective $\mu_k = 1$. Also any other weighting of aspects can be realized by choosing adequate values for μ_k. In Cupboard, in case a user does not specify values for μ_k, we assume that all μ_k are equal, i.e., the user does not put more importance on a special ontology aspect. We thus set $\mu_k = 1/|X|$ as the default value.

5.4 Facilitating Ontology Reuse using TS-ORS

As laid out in Section 2.4, the common ontology reuse process consists of the steps ontology discovery, ontology selection, and ontology integration. Furthermore we stated that the ontology discovery and ontology integration phase are decently covered by ontology search engines and ontology engineering environments.

The main problem of current tools for ontology selection is that they often try to compute the quality or suitability for a task automatically, since this is the only way to scale up to millions of ontologies. The methods used can best be described as heuristics and metrics, but they can never replace the evaluation performed by a human user (apart from clearly objective measures like statistics on the ontology or correctness according to a reasoner). Furthermore, most techniques require much computation that cannot be performed at runtime. In the TS-ORS, we overcome these problems by relying on the evaluation of human users. Furthermore the data in the system is stored in databases, and ontologies are not processed at runtime. Since the more complex computations can be performed offline, the TS-ORS can provide fast ratings for ontologies at runtime (see Chapter 12). So the ranking of ontologies can be based on the rating they receive from the most trusted users. We have implemented this solution within the Cupboard Plugin, which we present in Chapter 9. By providing help during the most difficult part of the reuse process, the perceived difficulty of the selection task can be decreased (see Chapter 14). Of course this approach also has the problem that it relies on human evaluations, which are not always easy to get and do not scale up the same way automatic evolutions do. In the following section, we show how automatic evaluation techniques can be integrated into the TS-ORS framework.

5.5 Automatic Evaluation Techniques and the TS-ORS

A key requirement for integrating an automatic evaluation technique into the TS-ORS is the possibility to interpret the output of the automatic evaluation technique in terms of the 5 star rating scale. In theory one would expect that such a mapping is possible, since, in the end, the output of an automatic evaluation technique has to be interpretable to be useful. If the mapping to the 5 star scale can be provided, for example by the creator of the evaluation technique, the integration is very easy. One can create one user per automatic evaluation technique, and run the technique on all ontologies that qualify for this evaluation method. The output of the evaluation technique would then be added as the review (the textual explanation of the rating), and the interpreted output (based on the given mapping to a 5 star rating scale) is added as the ontology rating. Users can then choose to trust or distrust an evaluation technique, just as they would trust or distrust human users. For the TS-ORS algorithms, it does not matter if a rating comes from a human user or from an automatic evaluation technique. The rating of an automatic technique is treated the same as any other rating according to the trust issued by users. This way, the overall rating of an ontology could be partly based on reviews and ratings from human users, but also partly on ratings from automatic evaluation techniques. By allowing the integration of automatic techniques within the TS-ORS, we can ensure that the system can scale at least as well as the automatic evaluation techniques that are integrated. Because of the trust information the system has available on the ratings, the TS-ORS has an edge over regular evaluation techniques. Within the TS-ORS, the best review and rating can be chosen to compute an overall rating from a number of potential ratings and reviews.

Obviously, finding a mapping for the output of an automatic evaluation technique to the 5 star scale is the most difficult part of the integration. The easiest mapping can be established in case the output is a number which does not have any relation to other objects being evaluated. For approaches like (Burton-Jones *et al.*, 2005), where the input of the automatic evaluation technique is an ontology, and the output a value that can be normalized to a value between 0 and 1, one can find 5 intervals to represent the star ratings. One example could be mapping a value between 0 and 0.2 to 1 star, 0.21 to 0.4 to 2 stars and so on, until finally 0.81 to 1 maps to 5 stars. In the end the mapping has to be chosen in a way that ensures that the equidistance of the star ratings is preserved.

Another important point is that the results of the automatic evaluation technique have to be in the system before an actual query is posed to the system. That means, that the automatic evaluation techniques cannot be applied at runtime.

For automatic evaluation techniques that just provide a ranking of ontologies as an output, one approach is to determine a ranking of all ontologies, and group the ontologies into 5 buckets, according to their rank. The top 20% would receive a 5 stars rating, the next 20% a 4 stars rating and so on, until the bottom 20% would be rated 1 star.

Figure 5.1 provides a depiction of our proposed process. First it has to be determined which kind of aspects or ontologies can sensibly be evaluated with the automatic evaluation technique. Then, the technique can be run on all aspects and/or ontologies for which it can provide sensible results. The results of the automatic evaluation techniques alongside the mapped 5 star rating are then inserted into the TS-ORS. All data is entered under a user name which is unique to a specific evaluation technique. This enables a user to know directly which technique was used to provide the automated rating, and furthermore it facilitates trust management. A user might want to trust a given evaluation technique globally because it represents exactly the kind of evaluation needed for the task the ontology is intended for, or, in general, distrust an evaluation technique that does not provide the desired ratings.

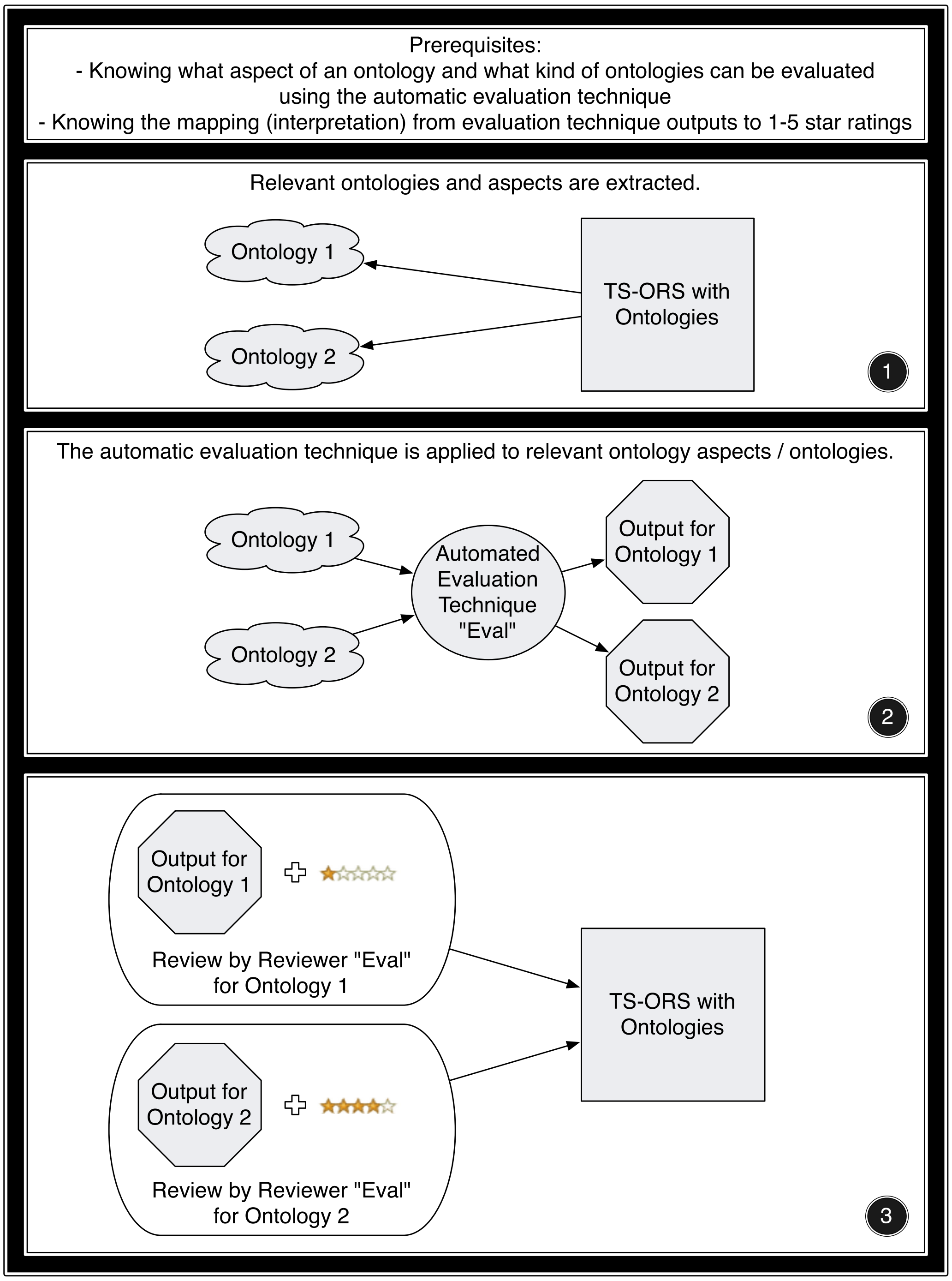

Prerequisites:
- Knowing what aspect of an ontology and what kind of ontologies can be evaluated using the automatic evaluation technique
- Knowing the mapping (interpretation) from evaluation technique outputs to 1-5 star ratings
Relevant ontologies and aspects are extracted.
Ontology 1
Ontology 2
TS-ORS with Ontologies
1
The automatic evaluation technique is applied to relevant ontology aspects / ontologies.
Ontology 1
Ontology 2
Automated Evaluation Technique "Eval"
Output for Ontology 1
Output for Ontology 2
2
Output for Ontology 1
Review by Reviewer "Eval" for Ontology 1
Output for Ontology 2
Review by Reviewer "Eval" for Ontology 2
TS-ORS with Ontologies
3

Figure 5.1: A depiction of how automatic evaluation techniques can be integrated within the TS-ORS.

5.6 Exemplary Computation

In order to provide a better understanding on how the algorithms presented in Chapter 4 work, we now present an exemplary calculation. We will give data needed for each algorithm in advance and then show the results of the computation as detailed as necessary. For the sake of space, we will not provide a giant example with all data needed, but will at some points rely on results from algorithms that we have already discussed, but that were run again on additional data. For example, the computation of the trust ranks and trust matrices has to be performed for each $o_n x_k$ combination, but we will only show the computation for one combination and then rely on the results from running the algorithms for the rest of the combinations.

5.6.1 Data used for Computations

For the following examples, let us assume we have the following trust matrix and distrust matrix extracted from our trust data for combination $o_1 x_1$:

$$\mathbf{T}_{o_1x_1} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \mathbf{D}_{o_1x_1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

5.6.2 Computation of GlobalTrustRank and GlobalDistrustRank

We start the computation of *GlobalTrustRank*(GTR) defined in Equation 4.1 with $d = 0.85$. All ranks are initialized with 1.

$$GTR_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$GTR_1 = \begin{pmatrix} (1-0.85)+0.85*(1/3+1/2+1/2) \\ (1-0.85)+0.85*(1/3+1/2) \\ (1-0.85)+0.85*(1/2+1/2+1/2) \\ (1-0.85)+0.85*(1/2) \\ (1-0.85)+0.85*(1/3+1/2) \end{pmatrix} = \begin{pmatrix} 1.283 \\ 0.858 \\ 1.425 \\ 0.575 \\ 0.858 \end{pmatrix}$$

$$GTR_2 = \begin{pmatrix} (1-0.85)+0.85*(1.283/3+1.425/2+0.575/2) \\ (1-0.85)+0.85*(1.283/3+0.858/2) \\ (1-0.85)+0.85*(0.858/2+1.425/2+0.858/2) \\ (1-0.85)+0.85*(0.575/2) \\ (1-0.85)+0.85*(1.283/3+0.858/2) \end{pmatrix} = \begin{pmatrix} 1.364 \\ 0.878 \\ 1.485 \\ 0.394 \\ 0.878 \end{pmatrix}$$

$$GTR_3 = \begin{pmatrix} (1-0.85)+0.85*(1.364/3+1.485+0.394/2) \\ (1-0.85)+0.85*(1.364/3+0.878/2) \\ (1-0.85)+0.85*(0.878/2+1.485/2+0.878/2) \\ (1-0.85)+0.85*(0.394/2) \\ (1-0.85)+0.85*(1.364/3+0.878/2) \end{pmatrix} = \begin{pmatrix} 1.335 \\ 0.910 \\ 1.528 \\ 0.318 \\ 0.910 \end{pmatrix}$$

$$\vdots$$

$$GTR_{10} = \begin{pmatrix} (1-0.85)+0.85*(1.316/3+1.604/2+0.261/2) \\ (1-0.85)+0.85*(1.316/3+0.909/2) \\ (1-0.85)+0.85*(0.909/2+1.604/2+0.909/2) \\ (1-0.85)+0.85*(0.261/2) \\ (1-0.85)+0.85*(1.316/3+0.909/2) \end{pmatrix} = \begin{pmatrix} 1.316 \\ 0.909 \\ 1.605 \\ 0.261 \\ 0.909 \end{pmatrix}$$

$$GTR_{11} = \begin{pmatrix} (1-0.85)+0.85*(1.316/3+1.605/2+0.261/2) \\ (1-0.85)+0.85*(1.316/3+0.909/2) \\ (1-0.85)+0.85*(0.909/2+1.605/2+0.909/2) \\ (1-0.85)+0.85*(0.261/2) \\ (1-0.85)+0.85*(1.316/3+0.909/2) \end{pmatrix} = \begin{pmatrix} 1.316 \\ 0.909 \\ 1.605 \\ 0.261 \\ 0.909 \end{pmatrix}$$

After 10 iterations in our example, the *GlobalTrustRank*s have converged and are stable, i.e., they do not change with additional iterations. When this state is achieved, the algorithm can terminate.

Next *GlobalDistrustRank*(GDR) as defined in Equation 4.2 is computed.

$$GDR = \begin{pmatrix} 0.909/1 \\ 0 \\ 0 \\ 0 \\ 0.261/1 \end{pmatrix} = \begin{pmatrix} 0.909 \\ 0 \\ 0 \\ 0 \\ 0.261 \end{pmatrix}$$

5.6.3 Computation of Trust Matrix F and Interpretation Matrix I

Then **F** and **I** are computed based on Algorithm 2, with $\beta_1 = 0.4$, $\beta_2 = 0.4$, $\beta_3 = 0.1$, $\beta_4 = 0.1$, $\gamma = 0.9$, $K = 7$

$$\mathbf{T}_{o_1x_1} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \mathbf{T}^{\intercal}_{o_1x_1} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{T}^{\intercal}\mathbf{T}_{o_1x_1} = \begin{pmatrix} 3 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 0 & 1 \\ 1 & 1 & 3 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 2 \end{pmatrix} \mathbf{T}\mathbf{T}^{\intercal}_{o_1x_1} = \begin{pmatrix} 3 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 0 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 0 & 1 & 2 & 0 \\ 1 & 1 & 1 & 0 & 2 \end{pmatrix}$$

$$(\mathbf{T}-\mathbf{D})_{o_1x_1} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ -1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{C}_{o_1x_1} = \begin{pmatrix} (0.4*1+0.4*3+0.1*1+0.1*3)=2 & \cdots = 0.9 & 0.6 & 0.6 & 0.9 \\ (0.4*1+0.4*1+0.1*1+0.1*1)=0.6 & \cdots = 1.5 & 0.9 & 0 & 0.5 \\ (0.4*1+0.4*1+0.1*0+0.1*1)=0.9 & \cdots = 0.6 & 1.9 & 0.1 & 0.6 \\ (0.4*1+0.4*1+0.1*0+0.1*1)=0.9 & \cdots = 0 & 0.1 & 1.1 & 0 \\ (0.4*0+0.4*1+0.1*1+0.1*1)=0.6 & \cdots = 0.5 & 0.9 & 0 & 1.5 \end{pmatrix}$$

$$(\mathbf{C}\cdot(\mathbf{T}-\mathbf{D}))_{o_1x_1} = \begin{pmatrix} 2.300 & 2.900 & 2.400 & 0.600 & 2.300 \\ 0.000 & 2.100 & 2.900 & 0.000 & 1.100 \\ 2.300 & 1.500 & 3.100 & 0.100 & 1.400 \\ 2.100 & 0.900 & 0.100 & 1.100 & -0.200 \\ 1.000 & 1.100 & 2.900 & 0.000 & 2.100 \end{pmatrix}$$

$$\mathbf{C}^2_{o_1x_1} = \begin{pmatrix} 6.160 & 3.960 & 4.020 & 1.920 & 3.960 \\ 3.210 & 3.580 & 3.870 & 0.450 & 2.580 \\ 4.320 & 3.150 & 5.240 & 0.840 & 3.150 \\ 2.880 & 0.870 & 0.840 & 1.760 & 0.870 \\ 3.210 & 2.580 & 3.870 & 0.450 & 3.580 \end{pmatrix}$$

$$(\mathbf{C}^2\cdot(\mathbf{T}-\mathbf{D}))_{o_1x_1} = \begin{pmatrix} 8.140 & 10.120 & 11.94 & 1.92 & 8.200 \\ 3.950 & 6.790 & 10.03 & 0.45 & 5.340 \\ 7.250 & 7.470 & 11.54 & 0.84 & 6.630 \\ 4.610 & 3.750 & 2.58 & 1.76 & 1.990 \\ 4.950 & 5.790 & 10.03 & 0.45 & 6.340 \end{pmatrix}$$

$$\mathbf{C}^3_{o_1x_1} = \begin{pmatrix} 22.418 & 15.876 & 18.654 & 6.210 & 15.876 \\ 14.004 & 11.871 & 14.868 & 2.808 & 10.871 \\ 17.892 & 13.332 & 18.302 & 4.040 & 13.332 \\ 9.144 & 4.836 & 5.066 & 3.748 & 4.836 \\ 14.004 & 10.871 & 14.868 & 2.808 & 11.871 \end{pmatrix}$$

$$(\mathbf{C}^3\cdot(\mathbf{T}-\mathbf{D}))_{o_1x_1} = \begin{pmatrix} 31.406 & 38.294 & 50.406 & 6.210 & 32.084 \\ 19.809 & 25.875 & 37.610 & 2.808 & 22.067 \\ 26.902 & 31.224 & 44.966 & 4.040 & 27.184 \\ 13.122 & 13.980 & 14.738 & 3.748 & 10.232 \\ 20.809 & 24.875 & 37.610 & 2.808 & 23.067 \end{pmatrix}$$

$$\mathbf{C}^4_{o_1x_1} = \begin{pmatrix} 86.265 & 63.121 & 78.091 & 22.147 & 63.121 \\ 57.562 & 44.766 & 57.400 & 12.978 & 43.766 \\ 71.890 & 53.748 & 69.911 & 17.009 & 53.748 \\ 32.024 & 20.941 & 24.191 & 10.116 & 20.941 \\ 57.562 & 43.766 & 57.400 & 12.978 & 44.766 \end{pmatrix}$$

$$(\mathbf{C}^4\cdot(\mathbf{T}-\mathbf{D}))_{o_1x_1} = \begin{pmatrix} 123.383 & 149.385 & 204.3324 & 22.1472 & 127.238 \\ 83.173 & 102.328 & 145.933 & 12.978 & 88.350 \\ 105.062 & 125.638 & 177.4066 & 17.0094 & 108.629 \\ 45.390 & 52.965 & 66.0738 & 10.1158 & 42.849 \\ 84.173 & 101.328 & 145.933 & 12.978 & 89.350 \end{pmatrix}$$

$$\mathbf{C}^5_{o_1x_1} = \begin{pmatrix} 338.489 & 250.734 & 315.964 & 83.930 & 250.734 \\ 231.583 & 175.278 & 224.575 & 54.553 & 174.278 \\ 286.506 & 214.144 & 274.412 & 68.836 & 214.144 \\ 120.054 & 85.219 & 103.884 & 32.761 & 85.219 \\ 231.583 & 174.278 & 224.575 & 54.553 & 175.278 \end{pmatrix}$$

$$(\mathbf{C}^5\cdot(\mathbf{T}-\mathbf{D}))_{o_1x_1} = \begin{pmatrix} 487.649 & 589.223 & 817.432 & 83.930 & 505.293 \\ 335.432 & 406.862 & 574.131 & 54.553 & 351.309 \\ 415.610 & 500.650 & 702.699 & 68.836 & 431.814 \\ 171.479 & 205.272 & 274.321 & 32.761 & 172.511 \\ 336.432 & 405.862 & 574.131 & 54.553 & 352.309 \end{pmatrix}$$

$$\mathbf{C}^6_{o_1x_1} = \begin{pmatrix} 1337.763 & 995.687 & 1263.139 & 327.013 & 995.687 \\ 924.115 & 693.226 & 885.698 & 221.415 & 692.226 \\ 1138.907 & 850.789 & 1085.628 & 275.064 & 850.789 \\ 465.349 & 340.816 & 426.081 & 118.457 & 340.816 \\ 924.115 & 692.226 & 885.698 & 221.415 & 693.226 \end{pmatrix}$$

$$(\mathbf{C}^6\cdot(\mathbf{T}-\mathbf{D}))_{o_1x_1} = \begin{pmatrix} 1932.229 & 2333.450 & 3254.513172 & 327.012636 & 2006.438 \\ 1338.002 & 1617.342 & 2271.151036 & 221.41548 & 1394.926 \\ 1648.809 & 1989.696 & 2787.206444 & 275.063832 & 1714.632 \\ 669.072 & 806.165 & 1107.712164 & 118.45736 & 687.708 \\ 1339.002 & 1616.342 & 2271.151036 & 221.41548 & 1395.926 \end{pmatrix}$$

$$\mathbf{C}^7_{o_1x_1} = \begin{pmatrix} 5301.488 & 3953.244 & 5027.561 & 1288.686 & 3953.244 \\ 3675.904 & 2749.076 & 3506.345 & 886.596 & 2748.076 \\ 4523.383 & 3377.971 & 4304.964 & 1094.477 & 3377.971 \\ 1829.762 & 1356.094 & 1714.077 & 452.121 & 1356.094 \\ 3675.904 & 2748.076 & 3506.345 & 886.596 & 2749.076 \end{pmatrix}$$

$$(\mathbf{C}^7 \cdot (\mathbf{T} - \mathbf{D}))_{o_1x_1} = \begin{pmatrix} 7664.490 & 9254.732 & 12934.049 & 1288.686 & 7966.046 \\ 5319.770 & 6424.980 & 9003.496 & 886.596 & 5537.384 \\ 6544.852 & 7901.354 & 11060.907 & 1094.477 & 6806.877 \\ 2639.866 & 3185.856 & 4426.266 & 452.121 & 2733.736 \\ 5320.770 & 6423.980 & 9003.496 & 886.596 & 5538.384 \end{pmatrix}$$

Values f_{ij} are then computed by using the decay factor and the results of the previous propagation steps. For example, $f_{11} = 0.9 * 2.3 + 0.81 * 8.14 + 0.729 * 31.406 + 0.656 * 123.383 + 0.590 * 487.649 + 0.531 * 1932.229 + 0.478 * 7664.490 = 5093.229$.

$$\mathbf{F}_{o_1x_1} = \begin{pmatrix} 5093.229 & 6151.266 & 8581.223 & 860.875 & 5290.391 \\ 3525.778 & 4266.208 & 5986.244 & 584.865 & 3676.648 \\ 4348.526 & 5244.822 & 7347.884 & 725.187 & 4519.635 \\ 1764.439 & 2122.215 & 2924.012 & 310.331 & 1811.884 \\ 3530.474 & 4261.512 & 5986.244 & 584.865 & 3681.343 \end{pmatrix}$$

For the interpretation of the computed values in **F** the matrix **I** is initialized with the difference of **T** and **D**. We explain the so called majority rounding for the first user. The process starts by computing a sequence. Let us refer to the users in A as 1, 2, 3, 4, 5, according to their position in the matrices. In our example, the sequence for user 1 is $a(1) = 4$, $a(2) = 1$, $a(3) = 5$, $a(4) = 2$, $a(5) = 3$. So far, the interpretation of $a(2)$, $a(3)$, $a(4)$ are known to be trust. The interpretation sequence is thus $?, +, +, +, ?$. Because the neighboring entries are trust, the unknown values are also interpreted as trust. The interpretation then has to be written at the correct position in the interpretation matrix.

For user 2 the sequence is $a(1) = 4$, $a(2) = 1$, $a(3) = 5$, $a(4) = 2$, $a(5) = 3$. Originally, the interpretation of $a(2)$ is known to be distrust, and $a(4)$, $a(5)$ are known to be trust. The interpretation sequence thus is $?, -, ?, +, +$. Because the closest neighbor of $a(1)$ is known to be distrust, the interpretation for $a(1)$ is also distrust. For $a(3)$, there are two neighbors in the sequence, one interpreted as distrust and one as trust. Since the f_{ij} value of $a(3)$ is closer to $a(2)$ which is known to be distrust, $a(3)$ is also interpreted as distrust.

$$\mathbf{I}_{o_1x_1} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

5.6.4 Sorting the Ratings

For Algorithms 3 and 4 we assume that user 1 has rated o_1x_1 with 1 star, user 2 with 2 stars, user 3 with 3 stars, user 4 with 4 stars, and user 5 with 5 stars. For the computation, we use $\alpha = 0.7$. For both algorithms, we first need to compute the $GlobalCombinedRank(GCR)$.

$$GlobalCombinedRank(GCR) = \begin{pmatrix} 0.7 * 1.316 - 0.3 * 0.909 \\ 0.7 * 0.909 - 0.3 * 0 \\ 0.7 * 1.605 - 0.3 * 0 \\ 0.7 * 0.261 - 0.3 * 0 \\ 0.7 * 0.909 - 0.3 * 0.261 \end{pmatrix} = \begin{pmatrix} 0.6485 \\ 0.6363 \\ 1.1235 \\ 0.1827 \\ 0.558 \end{pmatrix}$$

With that, we first compute the sorted ratings using Algorithm 3 for user 2. For sake of presentation, we will present the triples alongside the rating as a vector with entries $(i_{ij}, f_{ij}, GCR(a_j), R(a_j, o_n, x_k))$.

$$\text{Unsorted list for user 2:} \begin{pmatrix} -1, & 3525.778, & 0.6485, & 1 \\ 1, & 4266.208, & 0.6363, & 2 \\ 1, & 5986.244, & 1.1235, & 3 \\ -1, & 584.865, & 0.1827, & 4 \\ -1, & 3676.648, & 0.558, & 5 \end{pmatrix}$$

$$\text{Sorted list for user 2:} \begin{pmatrix} 1, & 5986.244, & 1.1235, & 3 \\ 1, & 4266.208, & 0.6363, & 2 \\ -1, & 3676.648, & 0.558, & 5 \\ -1, & 3525.778, & 0.6485, & 1 \\ -1, & 584.865, & 0.1827, & 4 \end{pmatrix}$$

The resulting ranking of ratings is $3, 2, 5, 1, 4$. When computing the sorted ratings using Algorithm 4, the order changes:

$$\text{Unsorted list:} \begin{pmatrix} 0.6485, & 1 \\ 0.6363, & 2 \\ 1.1235, & 3 \\ 0.1827, & 4 \\ 0.558, & 5 \end{pmatrix}$$

$$\text{Sorted list:} \begin{pmatrix} 1.1235, & 3 \\ 0.6485, & 1 \\ 0.6363, & 2 \\ 0.558, & 5 \\ 0.1827, & 4 \end{pmatrix}$$

If only global trust can be used, the resulting ranking of ratings is $3, 1, 2, 5, 4$.

5.6.5 Computation of an Overall Rating

For the computation of the overall rating, we assume that we compute the result for user 2, and the ranking of ratings for o_1x_1 is $3, 2, 5, 1, 4$ (as shown above), for o_1x_2 is $2, 2, 3, 4, 5$, for o_1x_3 is $2, 3, 4, 4, 1$, and for o_1x_4 is $5, 5, 3, 1, 4$. We furthermore assume that for o_1x_5 no ratings are available, and that the following user parameters are used: $\nu = 0.85$, $\mu_1 = 2$, $\mu_2 = 3$, $\mu_3 = 1$, $\mu_4 = 0$, $\mu_5 = 3$, $N = 2$. The algorithm then iterates through all aspects from X and computes the rating for each aspect:

$D_{o_1x_1} := (0.54 * 3 + 0.46 * 2) = 2.54$
$D_{o_1x_2} := (0.54 * 2 + 0.46 * 2) = 2$
$D_{o_1x_3} := (0.54 * 2 + 0.46 * 3) = 2.46$
$D_{o_1x_4} := (0.54 * 5 + 0.46 * 5) = 5$
$D_{o_1x_5} := 0$ (because there were no ratings, the value was automatically set to 0)

After the ratings for the different aspects have been computed, the weights μ have to be computed.

$\mu_1 := 2/(2 + 3 + 1 + 0) = 0.33$
$\mu_2 := 3/(2 + 3 + 1 + 0) = 0.5$
$\mu_3 := 1/(2 + 3 + 1 + 0) = 0.17$
$\mu_4 := 0$
$\mu_5 := 0$

The overall rating for o_1 for user 2 can then be computed as:

$D_{o_1} := 0.33 * 2.54 + 0.5 * 2 + 0.17 * 2.46 + 0 * 5 + 0 * 0 = 2.256$

In this example we have demonstrated how the trust information together with the rating data can be used to compute a personalized overall rating of an object. In order to compare multiple objects, the overall rating for all objects is computed, and the results are then ordered in descending order.

Chapter 6

Evolution of Ratings

A very important and difficult question is how to deal with ratings that have been made on an object that has since changed, and whether users should be allowed to change their rating if they change their mind. We discuss how ratings can be changed or corrected in Section 6.1. Section 6.2 discusses how ratings for evolving objects should be treated.

6.1 Changing Ratings and Correcting Reviews

Everybody can make mistakes, for example have a typo in the review or click on 4 stars instead of 5 stars when selecting the rating. In case this mishap is discovered after hitting the send button, it is important that it still can be corrected. Internet forums normally have some minutes grace period in which a submitted post can be edited without this edit showing in the system. This feature was included exactly for the case of correcting typos. An alternative strategy is to allow changes while nobody has seen the post.

When looking at reviews and ratings in the TS-ORS, some potential problems come to mind when thinking about the possibility to change made reviews or ratings. Imagine one could, without limits, edit reviews or change ratings, whenever desired. This would open the door for attacks, where malicious users first gain trust by providing high quality reviews, and then, later, change the ratings once they gained sufficient trust statements. The result would be that good objects could be demoted, or bad objects could be promoted. While it still should be possible to correct typos, changing the rating, or even the review text too much could have a bad impact on the system. There are several possible solutions to the problem:

- One allows any change for 2 minutes after the review was originally posted. In case a reviewer spots a typo in his review, or any other error, this error can be corrected right away.
- One allows the review text to be changed at any time, but the rating itself not. In the end, what influences the rating results and computed scores provided by the algorithms is the rating, and not the review. The review is primarily provided as

an explanation of the rating to be understood by human users. While it might be the case that a user would not have trusted a reviewer for an aspect–object combination if the review explaining the rating would have been different, the rating value does not change, and therefore the effect of the change is limited. One could also consider to have all changes to a review text reviewed by dedicated administrators of the TS-ORS.

- One could think about allowing a reviewer to also update his rating at a later point, because the perception of what rating is deserved might change. In this case, users who are somehow connected to the changed rating, either by trust for the specific aspect–object combination or by meta-trust, should receive a notification which informs them about the change. Ideally, this notification should also provide the possibility to change one's trust towards the changed rating. If there is a dedicated trust statement for this aspect–object connection, and not only a meta-trust statement, this trust statement should not be taken into account until the user has confirmed the change.
- Another possibility would be to only allow deletion of a review, but no change of the review as such. Also trust statements made for this aspect–object combination would then have to be deleted. In case a new review is made by the same reviewer covering the same aspect–object combination, each user who is connected to the reviewer by meta-trust should be notified about the added review. This way, users can check whether they agree with the new review, or whether they want to distrust the reviewer from now on.

No matter which of the proposed solutions is implemented by the administrator of a TS-ORS, it is important that the integrity of the TS-ORS is preserved, i.e., that it is not possible for malicious users to attack the system by first gaining trust and then using the trust to manipulate rankings and scores by afterwards changing their ratings.

6.2 Ratings for Objects which Evolve

Depending on the objects in a TS-ORS it might be difficult to see that an object has changed. Imagine a good restaurant changing the cook. The address and location stay the same, but the taste and possibly selection of food will likely change. In case the owner of the restaurant does not publish this change, and notify the provider of a TS-ORS, the knowledge that the reviewed object has evolved is not represented in the system. Reviews and ratings made after a change happened might improve or worsen, but the reviews that were made before the change still exist and are not marked as such. Especially if the ratings change after an object has evolved, this is problematic.

6.2.1 Identifying when Objects Change

Luckily, for many objects changes can be easily identified, for example when a book is published in a different revision, or the identification of an object somehow differs after the evolution. In our use case of ontology rating, it is very easy to distinguish different versions of ontologies, for example using hashes of ontologies to see whether something has changed. Many ontologies come with information on the current version number, which can be used to identify different versions of the same ontology. To check whether a change occurred, a crawler can periodically check the URIs of ontologies in the system and check whether a new version has been made available. For objects where it cannot be automatically checked whether the object that was rated has evolved, the users should be able to somehow contact the administrator of the TS-ORS to inform about a potential change concerning the reviewed object. In our example with the restaurant changing the chef, in case users inform the administrators of the TS-ORS about a change, the administrators could create two versions of the object, one before the change, and one after. When searching for the object, the user can be presented with a disambiguation page, similar to the ones in Wikipedia,[1] showing all versions of an object. In the case of the restaurant, the date of the personal change could be used for disambiguation, for example titling one object “Restaurant until 11.05.2010”, and another “Restaurant after 11.05.2010”. In case there are multiple changes, the dates can be stated as “from” and “until” in the title.

6.2.2 Managing Reviews and Ratings for Different Versions of an Object

In case an evolution of the rated object was identified, and several versions of an object exist in the system, the question is whether reviews for the former version of an object are also valid for the new version. In theory, an object could change in many different ways, potentially leaving certain aspects unchanged. Let us take the restaurant example again. Let us assume, that the object aspects of a restaurant would include location, food, pricing. In case the cook of the restaurant changes, this does not influence the location of the restaurant. In theory, ratings that were made on the location before the change of the cook, should remain valid.

In practice, judging whether some aspects of an object are not affected by the evolution is very difficult. While it might be possible for TS-ORS administrators to manually decide which of the former reviews should remain valid for the new version, this approach does not work in case frequent updates occur. We thus promote a different approach, which also works for ontology rating. In general, ratings should always stay with the version of the object that was reviewed. That means, that there is no automatic propagation of old reviews to the updated object. What should be

[1] `http://en.wikipedia.org/wiki/Wikipedia:Disambiguation`, last checked on 24.11.2010

done, though, is first providing a link to the old reviews, in case users want to decide for themselves whether an old review can still provide a benefit. Then, all users who provided reviews for that evolved object are notified by the system, informing them that there is a new version of an object they reviewed available, and allowing them to quickly either copy the old review to the current version, or enter an edited version of the old review for the current version of the object. In case a users confirms that his rating and review are still valid for the current version of the object, the review is copied and linked immediately. From that moment on, other users can see it, and it can be used in computations. This way, we avoid showing reviews to the users that are not valid for a newer version of an object and also allow reviewers to copy or adapt their review with as little effort as possible.

Especially for complicated objects like ontologies, there is no other way. While it might be possible to determine for some aspects automatically if an old review still holds, the potential for error is too big, so we rely on the reviewers to keep control over their reviews. Since reviewers who have reviewed the object once have shown that they are willing to invest some time to publicize their opinion about the object, it is not unlikely that they will cooperate and update their review, also to stay visible for other users.

Chapter 7

Initialization of the TS-ORS for Other Applications

We have shown how the TS-ORS can be adapted and initialized for ontology rating and reuse in Chapter 5. Since the model and algorithms we describe in Chapters 3 and 4 are generic in nature, the system can be used in many different contexts and with a variety of objects. The system can be employed whenever a personalized ranking of reviews or objects is needed. In this chapter, we first show how a legacy ORS can be updated to the TS-ORS in Section 7.1, and then show which steps have to be performed when initializing and installing the TS-ORS for a specific use case in Section 7.2. The chapter concludes with a discussion on how users can be motivated to provide ratings in Section 7.3.

7.1 Upgrading an Existing ORS to a TS-ORS

When an ORS exists in a given system, and the plan is to extend it to a TS-ORS, some design decisions have to be taken. First, it has to be checked whether the objects in the ORS are of homogeneous or heterogeneous nature. In case the objects are homogeneous, the case is simpler, because they all share the same aspects. Finding the right aspects in X also requires consideration, since the aspects will likely not change afterwards. As a rule of thumb, the most important requirement is that aspects have to be chosen that are reviewable. Also, the user will have to put a weight on each of the aspects when computing an overall rating. It is thus important, that the user can somehow link the different aspects to certain characteristics of an object that are needed for a given purpose. For example, in order to find a domain ontology, the user will put much importance and thus a high weight on the aspect domain coverage. Once the aspects have been chosen, it has to be decided how to handle existing ratings which cover the complete object. One possibility is to have a special aspect that only contains legacy ratings on the complete objects, and that users cannot edit or write to. After it is possible to review aspects of an object, users should not make ratings on the complete object anymore. Even if users wanted to give an overall rating, it is preferable that instead they try to think how the complete rating can be decomposed to the different aspects. The main benefit from this approach is that while reviewers have a certain weighting of the different aspects in their head when they give an overall rating, users might have a different use case requiring a different weighting. If only an overall rating is available, this is not possible. Still, keeping the legacy overall ratings is better than starting without any reviews, because until ratings on aspects are available, the system can provide a ranking based on the legacy ratings.

Apart from that, existing trust connections between users have to changed to overall meta-trust statements. Once this change has been made, the TS-ORS is ready for use. In the simplest case, which is keeping the overall ratings as one object aspect, and converting all trust statements to overall trust statements, the ranking and rating output of the algorithms would not change. This means the ranking results are still at least as good as in the ORS. But once the TS-ORS has been introduced, the users can provide more detailed reviews and more detailed trust statements, enabling the system to improve its ranking and rating computation with each additional rating or trust statement made.

Of course, the users also have to be informed about the changes and new possibilities in the system, ideally with examples and a detailed description on how to use the new features, and what changes in the user interface have occurred.

The situation is a little bit more complicated if the objects are heterogeneous in nature, and common aspects they all share cannot be found. In this case, a user cannot simply search over all objects and specify a weighting for the search. Ideally, objects can be clustered according to their aspects, meaning that objects that share the same aspects can later be searched together. In a system, there has to a mechanism which can keep track which aspects belong to which objects, and during the execution of the algorithms, only $o_n x_k$ combinations for which x_k is an aspect of o_n have to be taken into account.

7.2 Initializing a TS-ORS from scratch

When choosing to use a TS-ORS for one's application, it is also important to think about the kinds of objects that will be rated in the system, and the best object aspects they can be reviewed on. Once this crucial decision has been made, the user interface has to ensure that all functionality of the system is easily accessible and usable. We present an example of such a user interface in Chapter 9 which covers the Cupboard system, which uses the TS-ORS for ontology rating. Once the TS-ORS has been integrated into the application, it is time to fill the system with objects. Depending on the type of object that is going to be rated in the TS-ORS, either catalogues or lists exist that can be used for reference, or one can start by filling in some objects and letting users add the rest. In general it is a good idea to allow users to add objects to the system. With respect to the reviews, if possible it is a good idea to write some reviews, and ask other people to also review objects, so the system gets filled with some data. If possible and permitted, data from other rating systems can be imported. Since the reviews and ratings are a crucial part in any rating systems, it has to be made sure that incentives exist for user to write reviews and use the system. We provide some possibilities in the next section.

7.3 Incentives to Review and Rate Objects in the TS-ORS

Given that many rating systems give no monetary or material benefit for contributing to their system by means of reviews and ratings, some might find it surprising to still find many reviews, for example for products on Amazon.com. The survey provided in (Jøsang *et al.*, 2007) presents a detailed overview of different websites soliciting user ratings and feedback, and also explains which incentives users have to participate. The reasons differ based on the nature of the systems. In systems like Ebay, where it pays off to have a good reputation as a seller, the incentive is to maintain that reputation by continuing to provide good service and also providing ratings on buyers, which in turn will rate the seller. On e-commerce sites like Amazon.com, it pays to be in the top list of reviewers, because these are interesting for the industry and might be provided with free product samples from time to time. Some do not need financial or material rewards, but are helpful in nature and like to contribute to the community, for example by providing ratings which help the other users. Some like the competition, and want to become the number one reviewer, or be the most trusted reviewer. While motivations for users to contribute can differ and are still not fully understood, the key message is that in general user reviewing works; people do provide ratings in online rating systems.

Concrete incentives for providing ratings within the TS-ORS can be:

- Monetary or material. Especially in the beginning, when ratings are needed to get the momentum of a website going, it might be wise to invest part of the budget to either hire professionals to review the objects, or develop an incentive mechanism where users get paid if other users found their reviews helpful and trust them. In case there is real business value in having the ratings available, companies might be willing to sponsor or pay money to keep the incentives alive and the ratings coming in.

- Reputation or immaterial. One can have statistics in the system, for example who provided the most reviews, who provided the most helpful reviews, who is the most trusted reviewer and so on. One can also give out a title like “reviewer of the month”. Smith proposes to count ontology reviews as journal publications, thus providing an incentive for academics to invest their time to evaluate and review ontologies (Smith, 2008). Having the reputation to be a good reviewer can also lead to good opportunities in the industry, like being offered a job as ontology engineer or product tester.

- Entertainment. Luis von Ahn describes ways to turn tasks into games, thus making them more interesting for the participants (von Ahn, 2006). Depending on the type of object that should be rated, adapting the idea of product rating as a game is possible to some extent, even though this is not one of the typical tasks to be turned into a game. For some people it might even become a free time activity to review objects as a way to enjoy themselves and relax.
- Altruism. Some reviewers contribute with ratings because they want to help other users in making the same decisions. Also, once a user is able to provide a review of an object, the review only has to be written. The harder part of evaluating the object is usually done at this point.

Which kind of incentive works for a TS-ORS will always depend on the kind of object that is reviewed and the kind of resources available to promote the system. For ontology rating, the number of people available that are qualified to provide reviews is smaller than for reviewing microwave food. Also the number of people interested in ontologies is much smaller than for other domains. But, the advantage of having ontology ratings available is high in the industrial context. In case users are not willing to provide reviews, a combination of reputation or material incentives can be employed.

Part III

Implementation and Application

Chapter 8

Implementation of the Topic-Specific Trust Open Rating System

In this chapter, we first lay out the design decisions made in the process of implementing the TS-ORS in Section 8.1. We then present the architecture in Section 8.2 and the UML-Diagrams of our system in Section 8.3. The chapter concludes with a Section on optimization techniques we applied to improve performance (see Section 8.4). Parts of this chapter are based on (Lewen, 2009b; Lewen, 2009c; Sabou *et al.*, 2009; Angeletou *et al.*, 2010).

8.1 Design Decisions

We implemented the TS-ORS for integration with the Cupboard system (see Chapter 9). The required functionality was storing and serving TS-ORS related data (such as ratings, trust, or meta-trust) and providing review rankings or overall ratings. Our implementation is an instantiation of the model presented in Chapter 3 and the algorithms presented in Chapter 4. We did not implement the possible extensions, since they were not needed within Cupboard.

When interacting with external applications, the TS-ORS takes care of mediating between external and internal identifiers. It takes as input URIs as identifiers for ontologies and is flexible in terms of user identification. It has its own MySQL database to store reviews, users, ontologies and trust information. The application is implemented in JAVA, and servlets are used for interacting within Cupboard and with other programs. At the moment, REST services are offered that provide the results either as JSON, XML or HTML, based on content negotiation using the HTTP header.

8.2 Architecture

The architecture of the TS-ORS consists of a java application which connects to a MySQL database. We have optimized both the database schemata and the program code to achieve good performance at runtime (see Chapter 12 Section 12.2).

8.2.1 Database Schema

One of the foremost concerns when using databases in an application is developing a database schema that ensures data integrity but does not sacrifice performance. In order to optimize performance, enough memory has to be allocated for the database to allow for fast processing and also caching results. In our case we decided to store the most basic information, like users, ontology aspects, ontologies, ratings, trust between users, and meta-trust in dedicated tables (see Fig. 8.1). We use MySQL as a database with MyISAM storage engine for better performance. The different tables used to store and access the data are:

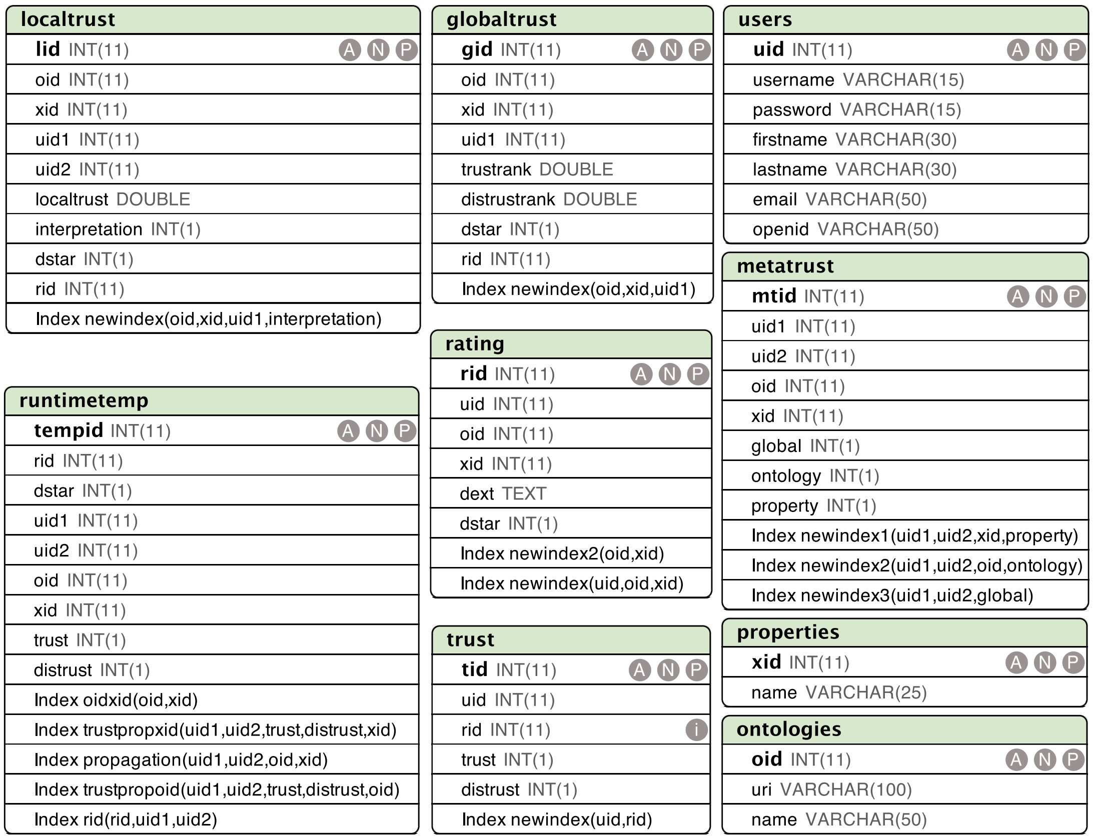


Figure 8.1: The TS-ORS Database Schema.

- **Users**: This table stores the user data. Each user can be uniquely identified with the user ID (*uid*).

- **Ontologies**: This table stores the names and the URIs of ontologies in the system. Each ontology can be uniquely identified with the ontology ID (*oid*).
- **Properties**: This table stores aspects (also called properties) of ontologies. Each aspect can be uniquely identified with the aspect ID (*xid*).
- **Rating**: This table stores the rating data from R. For each rating, the user, ontology, aspect are stored together with the textual review (*dext*) and the rating (*dstar*). Each rating is uniquely identified by the rating ID (*rid*).
- **Trust**: This table stores the trust information from W. Each trust statement is uniquely identified by the trust ID (*tid*). Trust and distrust are stored in different columns.
- **Metatrust**: This table stores all meta-trust statements. Each meta-trust statement is uniquely identified by the meta-trust ID (*mtid*). We allow the three meta-trust statements W_X, W_O, W_{OX} within our TS-ORS implementation (see Table 3.1).
- **Runtimetemp**: This table is only available after the trust computations have been performed at least one time, since it is the result of renaming a table used during the computation. During the initial computation of the *TrustRank* and *DistrustRank* (which are stored in *globaltrust*), and the local trust matrix **F** and interpretation matrix **I** (which are stored in *localtrust*), a table called *temp* is created. The table *temp* is the result of joining the *trust* and *rating* tables and adding database indices. The table is used to store the materialized meta-trust and constantly queried during the trust computations. After the computation has been completed, the *temp* table is renamed to *runtimetemp*. When a new round of computations is triggered, the system can continue to work with the *runtimetemp* table while running the computations on the *temp* table.
- **Globaltrust**: This table stores the *TrustRank* and *DistrustRank* of users for a given *oid*, *xid* combination. To improve runtime-performance, also the rating *dstar* and rating ID *rid* of a review are stored in this table. During the initial computation, a table named *tempglobaltrust* with an identical database schema is used, which is renamed to *globaltrust* once the computations are complete. The distinguishing between table that is used at runtime and a temporary table that is used during computations is necessary to ensure that the system can continue to work while the trust is re-computed.
- **Localtrust**: This tables stores the local trust matrix **F** and interpretation matrix **I**, which are the result of the local trust computation. To improve runtime-performance, also the rating *dstar* and rating ID *rid* of a review are stored in

this table. During the initial computation, a table named *templocaltrust* with an identical database schema is used, which is renamed to *localtrust* once the computations are complete.

8.3 UML-Diagram

We partitioned the functionality into methods belonging to one of several classes. UML class diagrams of the classes and methods can be found in Figures 8.2, 8.3, 8.4, 8.5. We now describe the functionality of each class and important methods.

- **Metatrust**: This class contains methods that are needed to set and retrieve meta-trust statements. The method *setEigentrust* is used to ensure that every reviewer trusts himself. The *propagateMetaTrustandDistrust* method is responsible for materializing the meta-trust statements. A complete UML class diagram can be found in Figure 8.2.

- **Computations**: This class contains the methods that compute *TrustRank*, *DistrustRank*, and the local trust matrix **F** with its interpretation matrix **I**. It furthermore contains methods responsible for retrieving the ranking of reviews, and for computing the overall ratings of an ontology. Methods that contain the string “global” are used when the user is unknown, otherwise the “local” methods are used. Methods that carry the string “update” in their name work on the tables that are used at runtime, while the normal computations are performed on the temporary tables. A complete UML class diagram can be found in Figure 8.2.

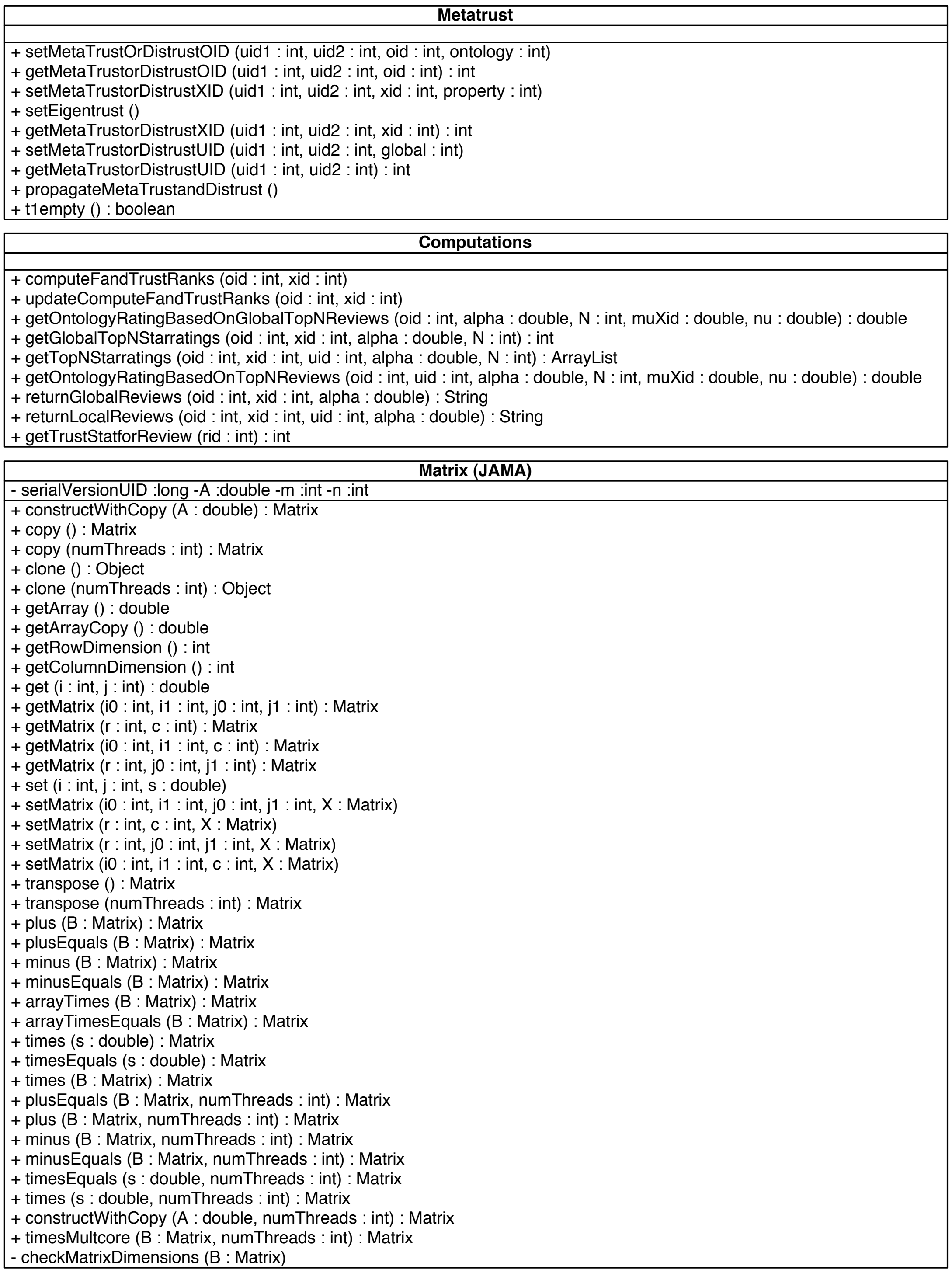
- **Matrix (JAMA)**: This class is an extension of the matrix class from the JAMA matrix package.[1] We have extended the code to allow a multi-threaded execution of the operations. Methods that have the integer *numThreads* as parameter have been extended and can be run on multiple threads. The number of threads can be specified using the *numThread* parameter. A complete UML class diagram can be found in Figure 8.2.

- **DBConnection**: This class contains methods used to establish a database connection from within the java program. For improved performance, we use the nanopool[2] connection pool. The connection itself is created using the MySQL Java connector. A complete UML class diagram can be found in Figure 8.3.

- **DBQuery**: This class contains the method used to pose a query against the database and retrieve the results. A complete UML class diagram can be found in Figure 8.3.

[1]http://math.nist.gov/javanumerics/jama/, last checked on 24.11.2010
[2]http://www.ohloh.net/p/nanopool, last checked on 24.11.2010

- **DBManipulateData**: This class contains methods used to insert data into the database, or update existing data within the database. A complete UML class diagram can be found in Figure 8.3.
- **DBInteraction**: This class contains methods that provide access to the database for methods from other classes. Most methods encapsulate SQL queries or update statements and simply fill in their parameters into SQL statements. Among other tasks, the class contains the methods used to read the trust and distrust matrices from the database, and also to store results. A complete UML class diagram can be found in Figure 8.3.
- **Setup**: This class contains methods that can be used to create and delete database tables and indices. A complete UML class diagram can be found in Figure 8.4.
- **Settings**: This class contains variables that are read by different methods. It is the main place to change settings of the application. A complete UML class diagram can be found in Figure 8.4.
- **CachedObjects**: This class contains objects that cache information from the database, so that database interactions can be minimized. A complete UML class diagram can be found in Figure 8.5.
- **Caching**: This class contains methods that are responsible for caching data that is likely retrieved at runtime. If a user logs in, for example, data he might request can be cached to improve performance. A complete UML class diagram can be found in Figure 8.5.
- **Updates**: This class contains methods that are used to alter and refresh data in the system. The method *recomputeEverything* is the main method that refreshes runtime data based on the information of the database tables. Since not all changes can be directly computed at runtime, it is important to call this method at fixed points in time (when exactly can be based on size of the system and frequency of changes). For example, while a regular trust statement only affects one ontology–aspect combination, for which the trust data can be quickly recomputed at runtime, a global meta-trust statement affects all ontology–aspect combination. In case such a meta-trust statement is made, it is stored in the database, but the change is not visible in the application until the runtime data has been recomputed. A complete UML class diagram can be found in Figure 8.5.



Metatrust

+ setMetaTrustOrDistrustOID (uid1 : int, uid2 : int, oid : int, ontology : int)
+ getMetaTrustorDistrustOID (uid1 : int, uid2 : int, oid : int) : int
+ setMetaTrustorDistrustXID (uid1 : int, uid2 : int, xid : int, property : int)
+ setEigentrust ()
+ getMetaTrustorDistrustXID (uid1 : int, uid2 : int, xid : int) : int
+ setMetaTrustorDistrustUID (uid1 : int, uid2 : int, global : int)
+ getMetaTrustorDistrustUID (uid1 : int, uid2 : int) : int
+ propagateMetaTrustandDistrust ()
+ t1empty () : boolean

Computations

+ computeFandTrustRanks (oid : int, xid : int)
+ updateComputeFandTrustRanks (oid : int, xid : int)
+ getOntologyRatingBasedOnGlobalTopNReviews (oid : int, alpha : double, N : int, muXid : double, nu : double) : double
+ getGlobalTopNStarratings (oid : int, xid : int, alpha : double, N : int) : int
+ getTopNStarratings (oid : int, xid : int, uid : int, alpha : double, N : int) : ArrayList
+ getOntologyRatingBasedOnTopNReviews (oid : int, uid : int, alpha : double, N : int, muXid : double, nu : double) : double
+ returnGlobalReviews (oid : int, xid : int, alpha : double) : String
+ returnLocalReviews (oid : int, xid : int, uid : int, alpha : double) : String
+ getTrustStatforReview (rid : int) : int

Matrix (JAMA)

- serialVersionUID :long -A :double -m :int -n :int

+ constructWithCopy (A : double) : Matrix
+ copy () : Matrix
+ copy (numThreads : int) : Matrix
+ clone () : Object
+ clone (numThreads : int) : Object
+ getArray () : double
+ getArrayCopy () : double
+ getRowDimension () : int
+ getColumnDimension () : int
+ get (i : int, j : int) : double
+ getMatrix (i0 : int, i1 : int, j0 : int, j1 : int) : Matrix
+ getMatrix (r : int, c : int) : Matrix
+ getMatrix (i0 : int, i1 : int, c : int) : Matrix
+ getMatrix (r : int, j0 : int, j1 : int) : Matrix
+ set (i : int, j : int, s : double)
+ setMatrix (i0 : int, i1 : int, j0 : int, j1 : int, X : Matrix)
+ setMatrix (r : int, c : int, X : Matrix)
+ setMatrix (r : int, j0 : int, j1 : int, X : Matrix)
+ setMatrix (i0 : int, i1 : int, c : int, X : Matrix)
+ transpose () : Matrix
+ transpose (numThreads : int) : Matrix
+ plus (B : Matrix) : Matrix
+ plusEquals (B : Matrix) : Matrix
+ minus (B : Matrix) : Matrix
+ minusEquals (B : Matrix) : Matrix
+ arrayTimes (B : Matrix) : Matrix
+ arrayTimesEquals (B : Matrix) : Matrix
+ times (s : double) : Matrix
+ timesEquals (s : double) : Matrix
+ times (B : Matrix) : Matrix
+ plusEquals (B : Matrix, numThreads : int) : Matrix
+ plus (B : Matrix, numThreads : int) : Matrix
+ minus (B : Matrix, numThreads : int) : Matrix
+ minusEquals (B : Matrix, numThreads : int) : Matrix
+ timesEquals (s : double, numThreads : int) : Matrix
+ times (s : double, numThreads : int) : Matrix
+ constructWithCopy (A : double, numThreads : int) : Matrix
+ timesMultcore (B : Matrix, numThreads : int) : Matrix
- checkMatrixDimensions (B : Matrix)

Figure 8.2: UML class diagram of classes needed for computation and meta-trust propagation.

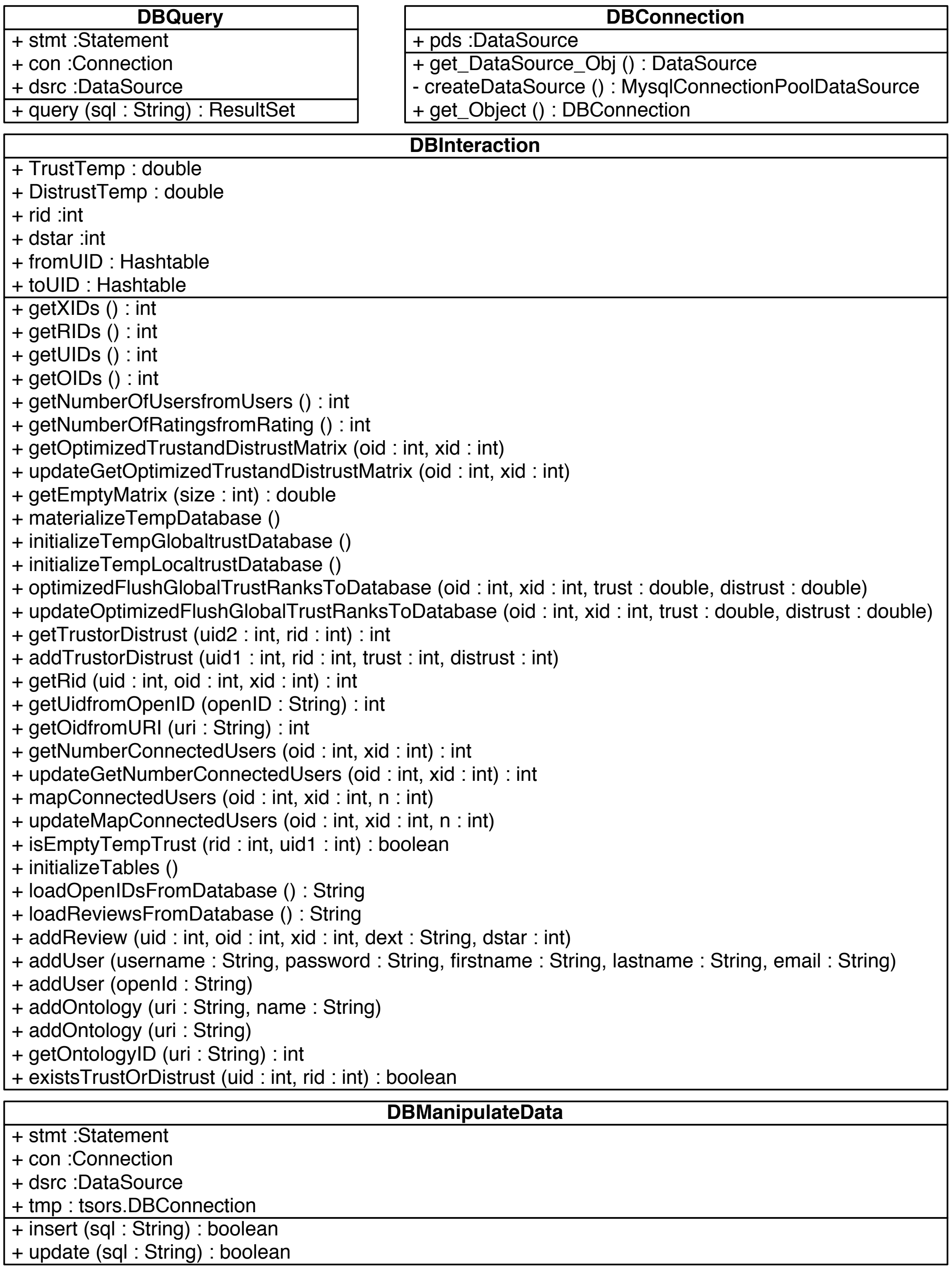

DBQuery
+ stmt :Statement
+ con :Connection
+ dsrc :DataSource
+ query (sql : String) : ResultSet
DBConnection
+ pds :DataSource
+ get_DataSource_Obj () : DataSource
- createDataSource () : MysqlConnectionPoolDataSource
+ get_Object () : DBConnection
DBInteraction
+ TrustTemp : double
+ DistrustTemp : double
+ rid :int
+ dstar :int
+ fromUID : Hashtable
+ toUID : Hashtable
+ getXIDs () : int
+ getRIDs () : int
+ getUIDs () : int
+ getOIDs () : int
+ getNumberOfUsersfromUsers () : int
+ getNumberOfRatingsfromRating () : int
+ getOptimizedTrustandDistrustMatrix (oid : int, xid : int)
+ updateGetOptimizedTrustandDistrustMatrix (oid : int, xid : int)
+ getEmptyMatrix (size : int) : double
+ materializeTempDatabase ()
+ initializeTempGlobaltrustDatabase ()
+ initializeTempLocaltrustDatabase ()
+ optimizedFlushGlobalTrustRanksToDatabase (oid : int, xid : int, trust : double, distrust : double)
+ updateOptimizedFlushGlobalTrustRanksToDatabase (oid : int, xid : int, trust : double, distrust : double)
+ getTrustorDistrust (uid2 : int, rid : int) : int
+ addTrustorDistrust (uid1 : int, rid : int, trust : int, distrust : int)
+ getRid (uid : int, oid : int, xid : int) : int
+ getUidfromOpenID (openID : String) : int
+ getOidfromURI (uri : String) : int
+ getNumberConnectedUsers (oid : int, xid : int) : int
+ updateGetNumberConnectedUsers (oid : int, xid : int) : int
+ mapConnectedUsers (oid : int, xid : int, n : int)
+ updateMapConnectedUsers (oid : int, xid : int, n : int)
+ isEmptyTempTrust (rid : int, uid1 : int) : boolean
+ initializeTables ()
+ loadOpenIDsFromDatabase () : String
+ loadReviewsFromDatabase () : String
+ addReview (uid : int, oid : int, xid : int, dext : String, dstar : int)
+ addUser (username : String, password : String, firstname : String, lastname : String, email : String)
+ addUser (openId : String)
+ addOntology (uri : String, name : String)
+ addOntology (uri : String)
+ getOntologyID (uri : String) : int
+ existsTrustOrDistrust (uid : int, rid : int) : boolean
DBManipulateData
+ stmt :Statement
+ con :Connection
+ dsrc :DataSource
+ tmp : tsors.DBConnection
+ insert (sql : String) : boolean
+ update (sql : String) : boolean

Figure 8.3: UML class diagram of classes needed for database interaction.

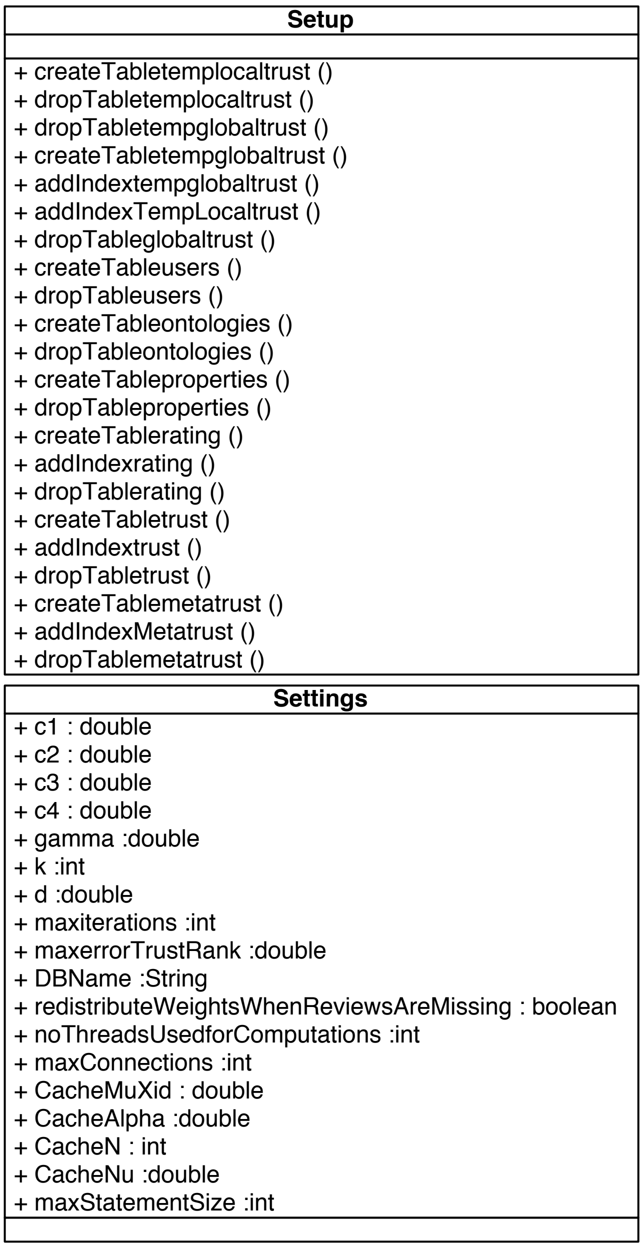
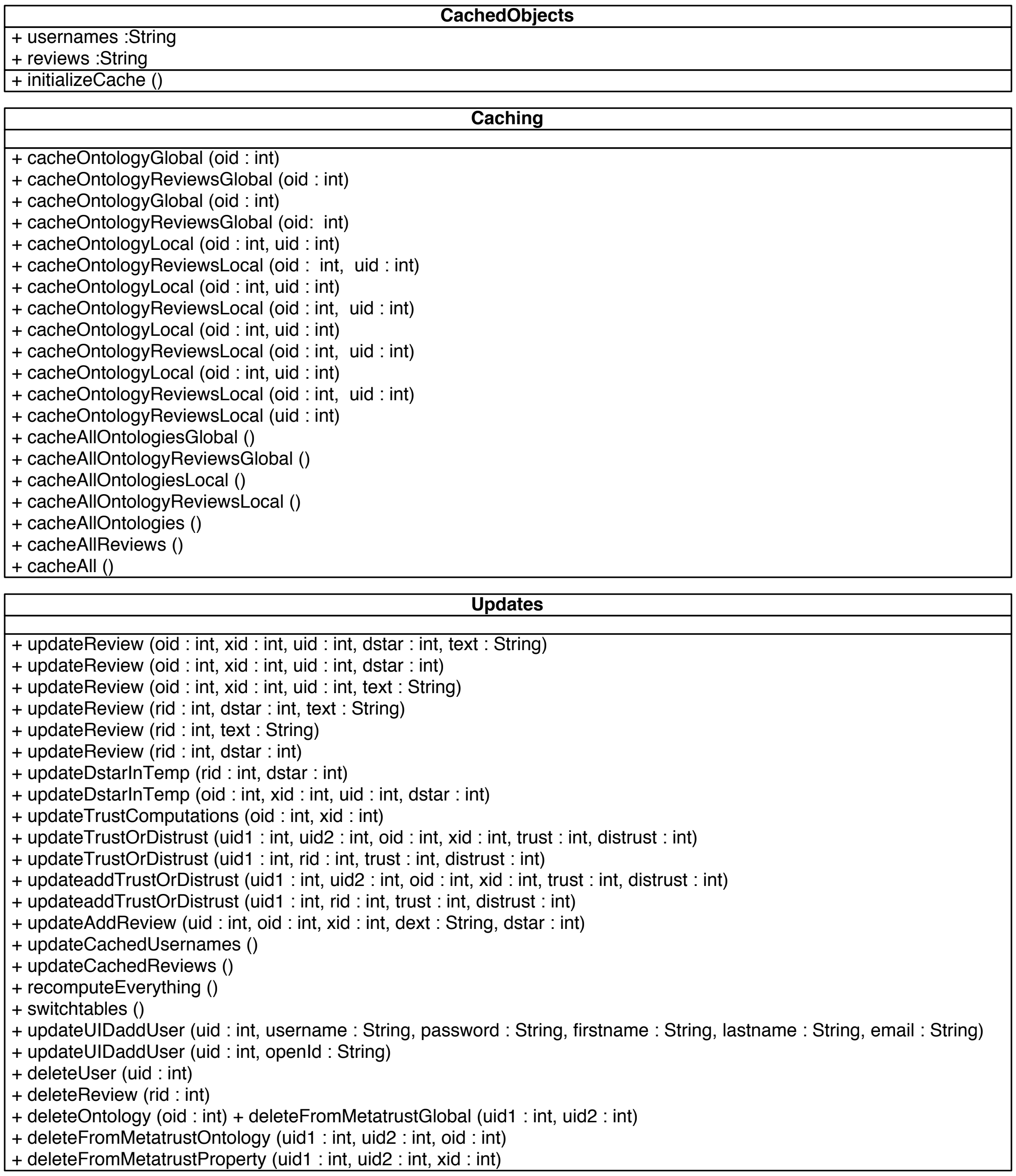


Figure 8.4: UML class diagram of classes needed for initializing the TS-ORS and storing settings.



```
CachedObjects
+ usernames :String
+ reviews :String
+ initializeCache ()

Caching

+ cacheOntologyGlobal (oid : int)
+ cacheOntologyReviewsGlobal (oid : int)
+ cacheOntologyGlobal (oid : int)
+ cacheOntologyReviewsGlobal (oid:  int)
+ cacheOntologyLocal (oid : int, uid : int)
+ cacheOntologyReviewsLocal (oid :  int,  uid : int)
+ cacheOntologyLocal (oid : int, uid : int)
+ cacheOntologyReviewsLocal (oid : int,  uid : int)
+ cacheOntologyLocal (oid : int, uid : int)
+ cacheOntologyReviewsLocal (oid : int,  uid : int)
+ cacheOntologyLocal (oid : int, uid : int)
+ cacheOntologyReviewsLocal (oid : int,  uid : int)
+ cacheOntologyReviewsLocal (uid : int)
+ cacheAllOntologiesGlobal ()
+ cacheAllOntologyReviewsGlobal ()
+ cacheAllOntologiesLocal ()
+ cacheAllOntologyReviewsLocal ()
+ cacheAllOntologies ()
+ cacheAllReviews ()
+ cacheAll ()

Updates

+ updateReview (oid : int, xid : int, uid : int, dstar : int, text : String)
+ updateReview (oid : int, xid : int, uid : int, dstar : int)
+ updateReview (oid : int, xid : int, uid : int, text : String)
+ updateReview (rid : int, dstar : int, text : String)
+ updateReview (rid : int, text : String)
+ updateReview (rid : int, dstar : int)
+ updateDstarInTemp (rid : int, dstar : int)
+ updateDstarInTemp (oid : int, xid : int, uid : int, dstar : int)
+ updateTrustComputations (oid : int, xid : int)
+ updateTrustOrDistrust (uid1 : int, uid2 : int, oid : int, xid : int, trust : int, distrust : int)
+ updateTrustOrDistrust (uid1 : int, rid : int, trust : int, distrust : int)
+ updateaddTrustOrDistrust (uid1 : int, uid2 : int, oid : int, xid : int, trust : int, distrust : int)
+ updateaddTrustOrDistrust (uid1 : int, rid : int, trust : int, distrust : int)
+ updateAddReview (uid : int, oid : int, xid : int, dext : String, dstar : int)
+ updateCachedUsernames ()
+ updateCachedReviews ()
+ recomputeEverything ()
+ switchtables ()
+ updateUIDaddUser (uid : int, username : String, password : String, firstname : String, lastname : String, email : String)
+ updateUIDaddUser (uid : int, openId : String)
+ deleteUser (uid : int)
+ deleteReview (rid : int)
+ deleteOntology (oid : int) + deleteFromMetatrustGlobal (uid1 : int, uid2 : int)
+ deleteFromMetatrustOntology (uid1 : int, uid2 : int, oid : int)
+ deleteFromMetatrustProperty (uid1 : int, uid2 : int, xid : int)
```

Figure 8.5: UML class diagram of classes needed for caching results and updating data in the system.

8.4 Employed Optimization Techniques

Over the years, we have run benchmarks to identify bottlenecks in the application and optimize it further.

8.4.1 Minimizing Database Interaction

One of the main bottlenecks of the application is its database interaction. Since database access requires hard-disk access (unless the query results are cached), it is much slower than computations done within the application, which use data from the main memory. One important technique to minimize the overhead of database interactions, is not initializing a connection each time access is needed, but to keep a couple of connections ready at all time. This is achieved using a connection pool. It is also preferable to collect data that should be written to the database and then make a large insert, instead of inserting parts of the data regularly. For this reason, whenever possible we keep data in main memory until enough data is gathered to actually write to the database. For example, during the computation of the local trust matrix **F**, the results are interpreted as trust or distrust row-wise. Instead of writing the result to the database row-by-row, we wait until all rows have been interpreted before writing the result to the database. In case the resulting database insert statement would be too big to be handled correctly by the database, we break it into smaller insert statements. The same way, when querying data, we try to bundle queries when possible. We prefer storing the result of a query that will be needed for later computations in main memory over accessing the database multiple times. Another crucial point is using database indices to speed up the response time of the database. Finding out which indices are needed can be achieved by analyzing each SQL statement, for example using the *explain* construct of the MySQL language. When the result of the *explain* query indicates that no index can be used for a query, it should be checked whether an index is needed to speed up the results.

Our database optimizations can be summarized as minimizing access to the database, and maximizing the data transferred on each access, while ensuring that indices are used whenever possible.

8.4.2 Only Processing Relevant Data

We first analyzed which of the data generated by the computations is really needed in the database. For example, trust scores for users who have not provided a rating for an ontology–aspect combination do not have to be stored, since they are never used for further computation at runtime. We thus check whether a rating exist before we write the local trust score and its interpretation to the database. We furthermore only store results that we cannot deduct quickly by another query. This way, we limit the size of the data.

Another observation we made is that users that are not connected to the WOT, because they have not made a rating for an ontology-aspect combination, or have not made a trust statement, do not influence the computation of trust scores for the rest of the users. We thus run the computation only for users for which we can later deduct new results. In a naive approach, for $100,000$ users, one might initialize the trust matrix as a $100,000^2$ matrix. However, this would not be possible in most systems due to the memory consumption. Instead, one would look for all users who are connected to the WOT and then initialize a smaller matrix. Of course, this only works if there are unconnected users.

8.4.3 Parallelizing Computations

Many of the algorithms we use can be run in a parallelized manner. Java allows to write applications using multi-threading, enabling the different threads to be distributed among CPU cores. We have implemented the algorithms to use multi-threading whenever possible to achieve a speedup. A performance improvement can be seen in systems with multi-core processors (see Chapter 12).

Since the computations for each ontology–aspect combination are independent of each other, they can also be distributed to different machines, thus parallelizing the computations even further.

8.4.4 Using Fast Hardware

Because our implementation relies on a database, a fast hard-disk can reduce the runtime of the different algorithms. This, together with a fast multi-core CPU and sufficient main memory provides a significant speedup compared to less powerful systems, like laptops. Furthermore, a cluster of servers can be used if extremely fast computation of trust scores is needed, since then the computation of trust scores can be performed in parallel on many machines. This, however, will likely only be needed for big e-commerce sites.

8.4.5 Reviewing the Source Code

Often, there are many ways to implement an algorithm. In order to minimize execution time, it is advisable to check for edge cases in the code, which might lead to a shortcut in computation. For example, if the weight for an ontology aspect is set to 0 during the computation of an overall rating, it is not necessary to retrieve the top-ranked ratings for that aspect. Finding these shortcuts can help to further improve the runtime of the application.

Chapter 9

Cupboard and TS-ORS Integration

We build the Cupboard system[1] (d'Aquin and Lewen, 2009; d'Aquin *et al.*, 2009) to tackle the ontology reuse problem. As motivated before, ontology search engines rarely provide help selecting ontologies from their result set. We argue that one factor inhibiting ontology reuse is the lack of support for ontology practitioners to find, assess, and exploit existing ontologies. Furthermore, it is difficult for ontology engineers to effectively deploy and expose their ontologies for use and reuse. Most ontology search engines lack mechanisms for assessing ontologies, for providing rich ontology metadata, or providing alignments between ontologies.

To tackle the problem of providing help for the selection step of ontology reuse, we combine Watson[2] (d'Aquin *et al.*, 2008d) technology with the TS-ORS, allowing users to review and rate ontologies and the system to use this data for result ranking. We furthermore include Oyster (Palma *et al.*, 2006), a peer-to-peer ontology sharing system which relies on rich metadata for ontologies, using the OMV format (Hartmann *et al.*, 2005). The alignment server (Euzenat, 2004) is integrated into Cupboard to allow the creation, management and evaluation of ontology alignments from within Cupboard. The emphasis of the Cupboard system lies on ontology publishing, sharing and reuse. Because for many ontology engineers, setting up their own server infrastructure, generating search indices and setting up SPARQL-endpoints is a cumbersome task, Cupboard provides a simple solution to these problems. Cupboard does not only allow users to add their ontologies in a personal ontology space, but actually indexes them (using the Watson engine), provides a mechanism to link them (using the Alignment Server), hosts them and exposes them through APIs and SPARQL. It enables the ontology engineer to deploy ontologies with minimal effort. Moreover, Cupboard is designed to be a community tool, helping ontology users and practitioners in finding and reusing ontologies, through the use of rich ontology metadata (thanks to Oyster and OMV), and to advanced ontology review mechanisms (using the TS-ORS system).

More detailed information on Cupboard and a user manual can be found in (Lewen *et al.*, 2009; Lewen *et al.*, 2010).

The chapter starts with an overview on Cupboard in Section 9.1. We then present

[1]`http://cupboard.open.ac.uk/`, last checked on 24.11.2010
[2]`http://watson.kmi.open.ac.uk/`, last checked on 24.11.2010

the Cupboard Architecture in Section 9.2. Section 9.3 explains how different components can be integrated with Cupboard, while versioning is covered in Section 9.4. Parts of this section are based on (Lewen *et al.*, 2009; Lewen *et al.*, 2010; d'Aquin and Lewen, 2009; d'Aquin *et al.*, 2009).

9.1 Overview

Cupboard differs from classical ontology repository systems since it does not provide a single space in which ontologies are exposed. On the contrary, it allows each user to create his own ontology space, containing added or selected ontologies. The main advantage in relying on the idea of ontology spaces is that each ontology space virtually implements a complete infrastructure for building semantic applications.

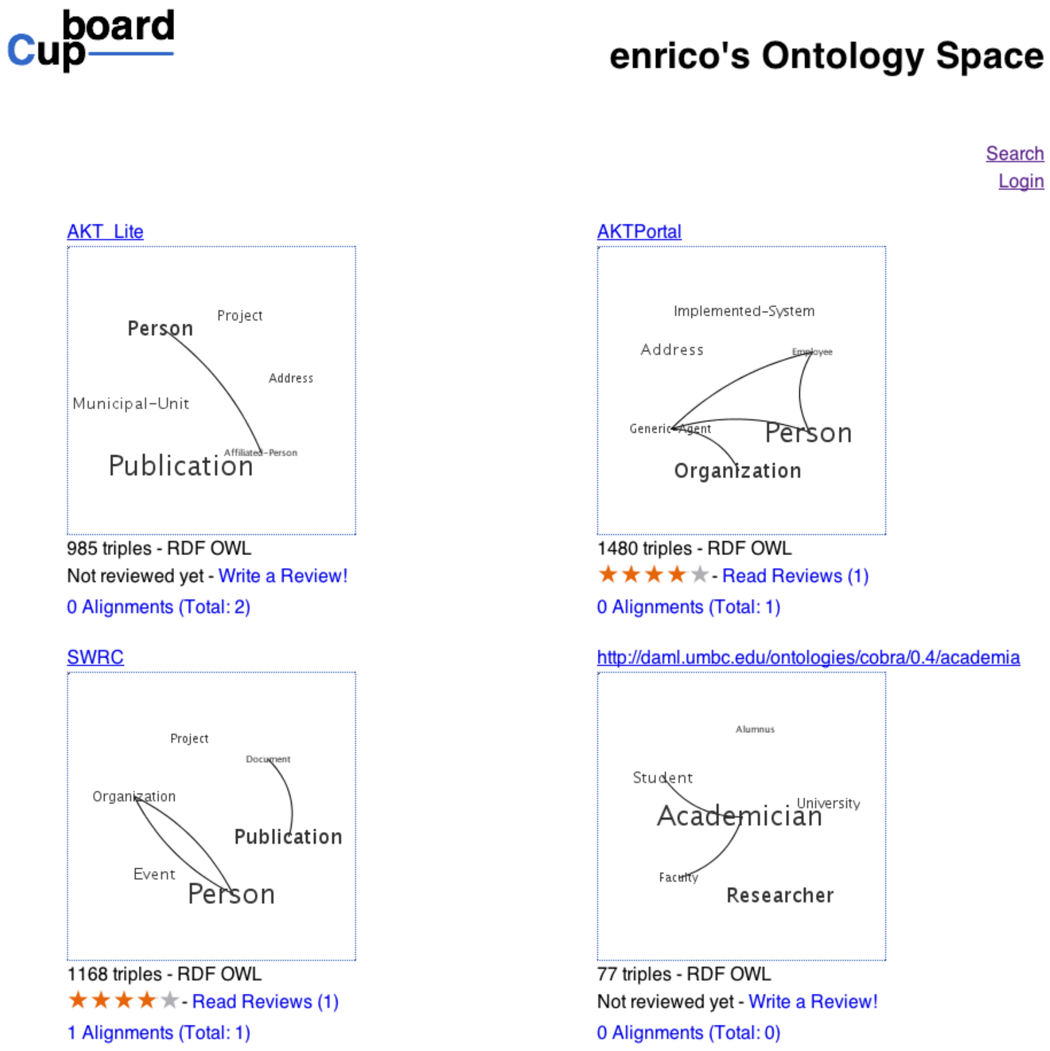
The uploaded ontologies are automatically indexed using the Watson engine. Furthermore, using the Watson APIs or SPARQL, also access to the ontology is provided using Watson technology. In Cupboard, each ontology space is like a little Watson, which enables developers to easily build semantic applications using the exposed ontologies from Cupboard.

Within an ontology space, the user can link ontologies with alignments. Alignments can be either downloaded from the Alignment server, if available, or computed locally.

The metadata of ontologies is stored in the OMV format and managed in the Oyster system. Users can enter metadata themselves, or rely on metadata found online in the Oyster peer-to-peer system. The metadata can be used for ontology search, as they provide an additional way to filter the results of a query. Let us assume that a user wanted to find all ontologies build using a certain methodology. Since the ontology itself does not carry this information, the only way to find the matching ontologies is to query the metadata.

Most importantly, Cupboard contains a TS-ORS to handle ontology ratings and ranking functionality. All users can check the different ontology spaces, and review ontologies in the spaces. In case an ontology is reviewed in one space, but exists in more ontology spaces, the review for this ontology becomes visible in all ontology spaces, regardless of the ontology space in which it was reviewed. Users can also state trust or distrust towards other users, enabling the TS-ORS to provide them a personalized ranking of reviews and computation of overall ratings.

A screenshot of an ontology space can be found in Figure 9.1. Figure 9.2 shows the ratings and reviews for an ontology within Cupboard.



AKT_Lite

985 triples - RDF OWL
Not reviewed yet - Write a Review!
0 Alignments (Total: 2)

AKTPortal

1480 triples - RDF OWL
★★★★☆ - Read Reviews (1)
0 Alignments (Total: 1)

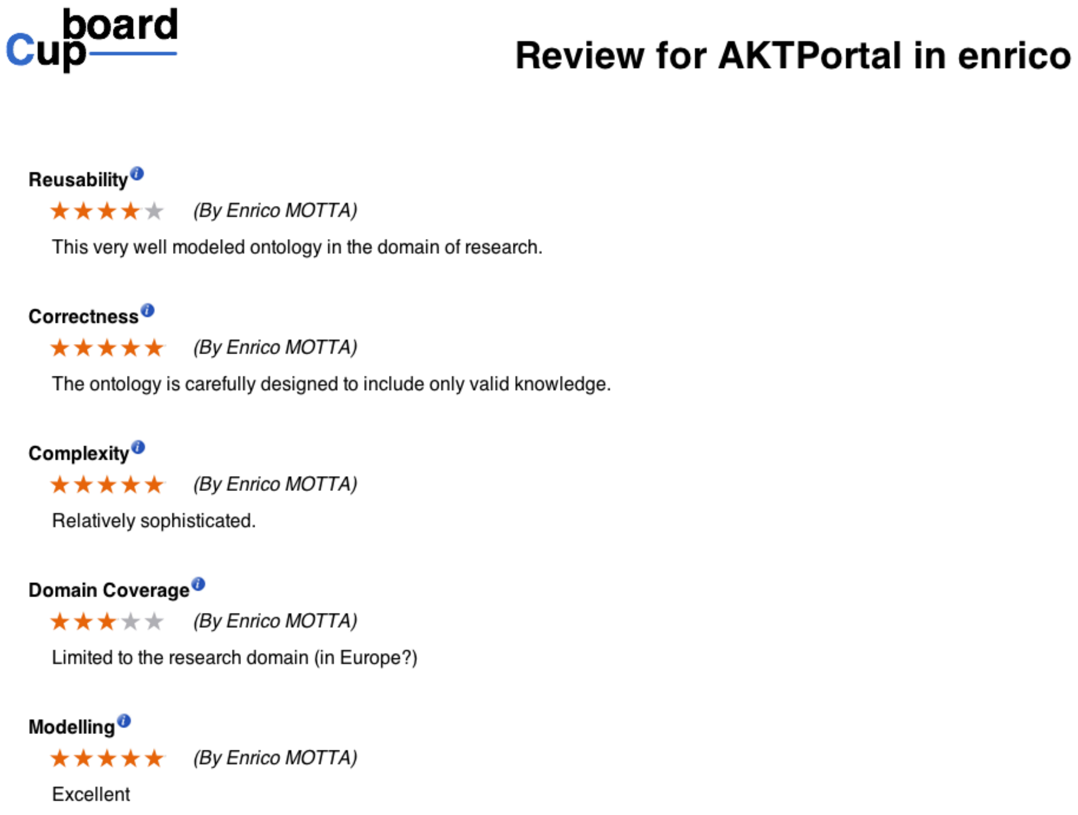
SWRC

1168 triples - RDF OWL
★★★★☆ - Read Reviews (1)
1 Alignments (Total: 1)

http://daml.umbc.edu/ontologies/cobra/0.4/academia

77 triples - RDF OWL
Not reviewed yet - Write a Review!
0 Alignments (Total: 0)

Figure 9.1: A screenshot of an ontology space in Cupboard. Ontologies with some statistical information, the key concept visualization and rating information can be seen.



Review for AKTPortal in enrico

Reusability
★★★★☆ *(By Enrico MOTTA)*
This very well modeled ontology in the domain of research.

Correctness
★★★★★ *(By Enrico MOTTA)*
The ontology is carefully designed to include only valid knowledge.

Complexity
★★★★★ *(By Enrico MOTTA)*
Relatively sophisticated.

Domain Coverage
★★★☆☆ *(By Enrico MOTTA)*
Limited to the research domain (in Europe?)

Modelling
★★★★★ *(By Enrico MOTTA)*
Excellent

Figure 9.2: Rating information within Cupboard for an ontology.

9.2 Architecture

While the different technologies we combined in Cupboard have been described and implemented before, this is the first application that combines them and exposes this broader and more comprehensive set of functionalities to ontology engineers and users alike. The main challenges of such an integration are defining suitable interfaces between the components, optimizing the performance and generating an intuitive Web interface. The user experience with the technologies within Cupboard is thus completely different from using the technologies separately.

The Cupboard architecture can be found in Figure 9.3. The central component of Cupboard is called the *Cupboard Core*, which is in charge of orchestrating the interaction between all components, external applications and users. It relies heavily on the Watson engine to provide basic functionalities such as storing and indexing the ontologies, validating them and exposing them to applications using APIs. The TS-ORS, Oyster, and the Alignment Server each provide specific functionalities to the system. These functionalities are federated and exposed in a homogeneous way to the Cupboard Web interface and APIs, through the Cupboard Core.

9.3 Implementation Details and Integration of Components

As visible in Figure 9.3, the architecture of Cupboard is centered around the “core” component, to which all the other components are connected. In the current implementation, the components that have been integrated are: Watson, Oyster, the TS-ORS, Key-Concept Visualization, and the Alignment Server.

9.3.1 The Cupboard Core

The role of the Cupboard Core component is twofold:

- It provides the interfaces to integrate components in the system and to federate them, in order to offer a complete set of functionalities.
- It provides the interfaces to the external world, in particular user interfaces, to access the functionalities offered by the other components in a homogeneous way.

As such, the Cupboard Core is the only part of the development of Cupboard which is entirely new and which does not rely on the reuse of previously existing components. In the following, whenever we talk about integration of components, this relates to their implementation within the Cupboard Core, which enables the interaction with the integrated components. Concretely, the Cupboard Core is implemented as a Java Web

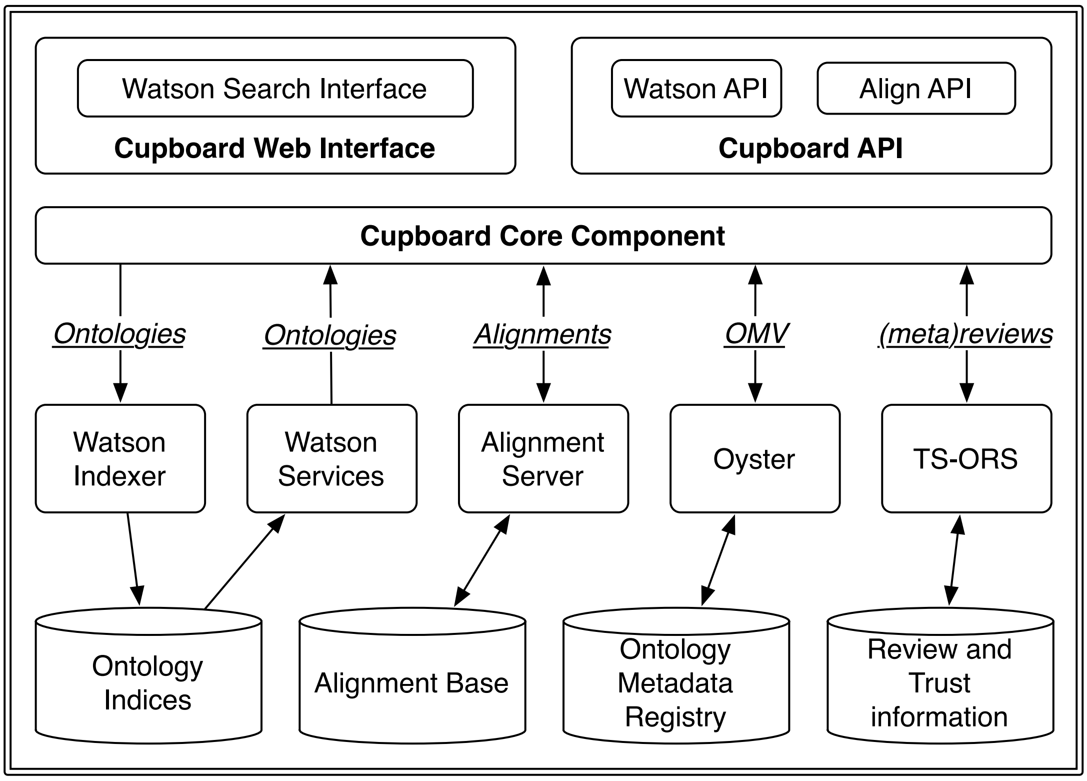


Figure 9.3: The Cupboard Architecture.

application (i.e., a set of servlets, enhanced with Javascript based interface elements). It provides a set of Graphical User Interfaces (GUIs) for the functions provided by Cupboard and also the back-end operations. It also provides a complete API to build applications on top of exposed ontology spaces.

9.3.2 Integration of Watson

The Watson system can be seen as a search engine for the Semantic Web. It supports users and applications in finding, selecting and (re)using ontologies that are available online, through search and exploration mechanisms. Its role in Cupboard is to index the ontologies that are uploaded, in order to provide the same search and exploration mechanisms available in Watson. Only this time ontologies are not crawled automatically from the Web, but only when uploaded to an ontology space within Cupboard. As such, the integration is realized by deploying a dedicated Watson engine within the Cupboard server, comprising both the Watson indexer and the Watson services. Concretely, this means that Cupboard relies on its own local indices for uploaded ontologies, which are produced, accessed and exploited through a local (i.e., on the Cupboard server) instantiation of the Watson engine.

In practice, the Watson engine is then used to provide the following functionalities:

- Retrieving all the ontologies contained in a given ontology space (one of the core functionalities of Cupboard).
- Analyzing the content of ontologies to automatically extract basic metadata (e.g., label, comment, URI, imported ontologies, etc.), which are then passed on to Oyster.
- Exploring the content of ontologies, by providing navigation functionality to investigate the relation between ontological entities.
- Searching for semantic content exposed through Cupboard, to find ontologies or ontological entities to be reused.

In addition, elements of the Watson Web interface are also reused within Cupboard (e.g., the search interface), and simply re-branded to appear as a homogeneous part of the Cupboard graphical user interface (GUI).

9.3.3 Integration of Oyster

Oyster is a distributed registry that exploits semantic web techniques in order to provide a solution for exchanging and re-using ontologies and related entities. As an ontology registry, it provides services for storage, cataloging, discovery, management, and retrieval of ontology metadata definitions. To achieve these goals, Oyster implements the OMV[3] metadata standard to describe ontologies and related entities, supporting advanced searches of the registered objects and providing services to support the management and evolution of ontologies in distributed environments.

The role of Oyster in Cupboard is the management of OMV metadata for the ontologies exposed within ontology spaces in order to facilitate their discovery by supporting metadata-based searches. Hence, the integration is realized by deploying an Oyster peer within the Cupboard server that is populated by metadata about every ontology uploaded into the system. Concretely, a local Oyster peer running in server mode is accessed by the Cupboard system via its API. Consequently, ontologies exposed within Cupboard also become available to the whole Oyster network.

In practice, it provides the following functionalities:

- Storing OMV metadata for ontologies exposed within ontology spaces.
- Retrieving OMV metadata for displaying ontology details.
- Searching for ontologies using metadata-based restrictions.
- Extracting metadata information from uploaded ontologies that is then complemented with metadata extracted by Watson.

9.3.4 Integration of the TS-ORS

The TS-ORS provides users the possibility to review ontologies within Cupboard. For each of the aspects of the ontologies, users can provide a 5 star rating and a textual explanation of their rating.

Users can express their trust or distrust towards other users. Based on the web of trust, the reviews can be ranked in a user-specific way and also overall ratings of the ontologies can be computed according to a users needs and preferences.

The role of TS-ORS in Cupboard is to add user-based evaluation to the ontologies. Without quality information, it is hard for users to judge which ontology is good enough to be reused in an application or another ontology. Because evaluating the ontologies can take some time, user like to rely on existing evaluations. The quality information provided by the community can then be used to order reviews of the ontologies, and also the ontologies themselves based on user-specific trust and parameters.

The TS-ORS's functionality is exposed through servlets, which are called by the Cupboard core. It can be seen as a REST-like API. The results are provided as HTML, XML or JSON, based on content negotiation between the Cupboard core and the TS-ORS (based on the HTTP-request header).

[3]http://sourceforge.net/projects/omv2/, last checked on 24.11.2010

In practice, the TS-ORS provides the following functionalities:

- Storing user reviews.
- Exposing user reviews in a personalized (if the user is identifiable) order.
- Calculating overall ratings for ontologies based on their ratings taking into account user-specific parameters and inter-user trust.
- Managing trust and meta-trust between users.

9.3.5 Integration of the Key Concept Visualization

Using the approach for identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures (Peroni *et al.*, 2008), a method was developed to visualize found key concepts. The method takes as input the ontology and renders the results of the key concept analysis as an image which is then displayed in the ontology space. The component is integrated via the Cupboard core.

9.3.6 Integration of the Alignment Server

The users can choose to add and load alignments for ontologies in their Cupboard space. They can furthermore set alignments as selected. This functionality is for use with the Cupboard APIs, it does not affect the ontology space as such. The integration itself is realized via the Cupboard core.

9.4 Versioning In Cupboard

Versioning is an important aspect to be considered in a system like Cupboard, especially when dealing with changing content that has been reviewed. First, each ontology in Cupboard is identified internally through a hash-code that ensures that the same ontology is not indexed more than once. This also ensures that all reviews entered in one of the ontology spaces can be seen in the rest of the ontology spaces. In addition, users have the possibility to submit new versions of ontologies already uploaded in their ontology spaces. The existence of a previous version of an ontology is detected by comparing the URIs, base namespaces, initial locations and, ultimately, the content of the ontologies. Once a user has validated the versioning link between the two considered ontologies, the new version is indexed in the same way as the original version was, also keeping track of the link between them within the metadata of the ontologies.

This allows the system to always display or provide the latest version of an ontology, while keeping a complete history of its evolution accessible to the user. In relation with this versioning mechanism, an important design decision in the system concerns the propagation of reviews between versions of the same ontologies. Indeed, user reviews provided for one version of an ontology might not be valid for a later version, but at the same time, the effort deployed by the users in reviewing ontologies should not need to be repeated if the changes introduced in the new version do not affect the quality of the ontology. Therefore, users can chose to be alerted of the appearance of new versions of the ontologies they have reviewed. They are then shown the differences between the versions and can decide to propagate their reviews for the new version or to modify them to take these changes into account.

Chapter 10

The Cupboard NeOn Toolkit Plugin

The Cupboard Plugin for the NeOn Toolkit connects the NeOn Toolkit ontology engineering environment with the Cupboard system. The plugin is build on the code basis of the Watson Plugin for the Neon Toolkit.

We start the chapter by presenting the Watson plugin for the NeOn Toolkit in Section 10.1. The Cupboard plugin is presented in Section 10.2. We then show how the functionality of the Cupboard plugin was incorporated in an updated version of the Watson plugin in Section 10.3.

10.1 Watson Plugin

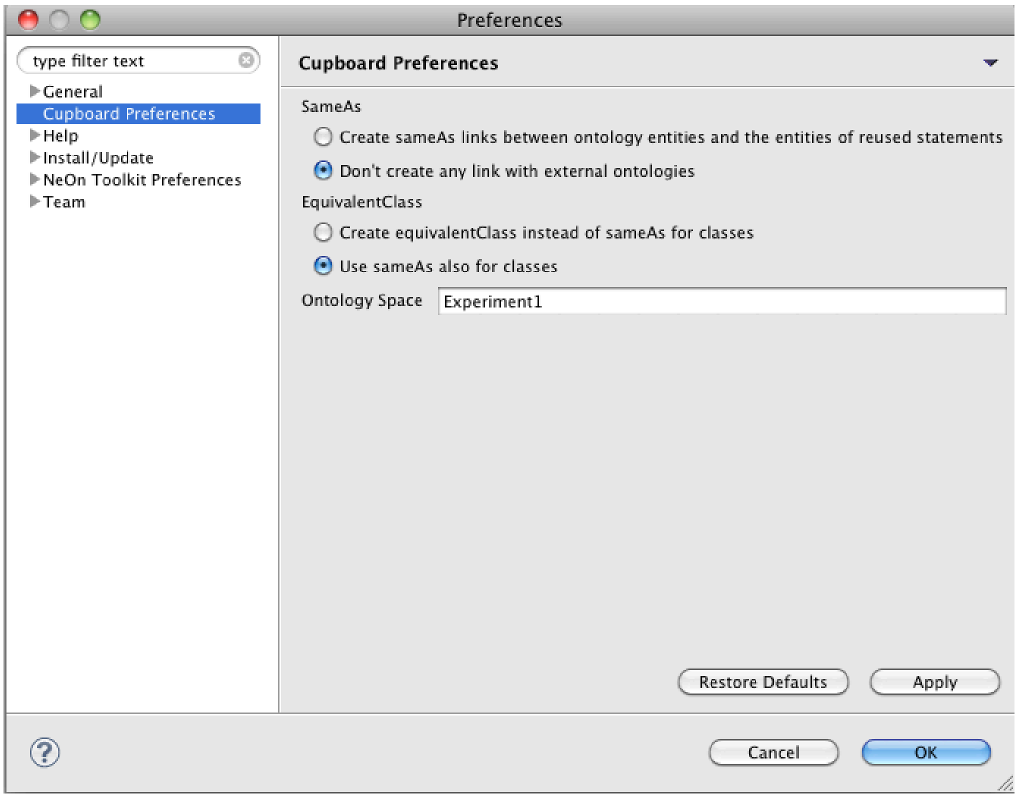
The Watson plugin for the NeOn Toolkit[1] (d'Aquin *et al.*, 2008c) uses the Watson client API to connect to Watson and enable interaction with Watson from within the ontology engineering environment. As mentioned in Section 2.4.1, with few steps a user can search ontologies from within the NeOn Toolkit (see Figure 2.3), and later browse the results (see Figures 2.4 and 2.5) and integrate them with one click (see Figure 2.6). As mentioned before, the major shortcoming of the Watson Plugin is its result ranking, which is based on Lucene, and thus does not mirror the quality of the ontologies. For that reason, we extended the Watson Plugin to integrate with Cupboard, displaying ontology ratings alongside the results, and basing the ranking upon overall rating scores for the ontologies, and not Lucene.

10.2 Cupboard Plugin

The Cupboard plugin for the NeOn Toolkit has a configuration page, where the ontology space to which the plugin should connect can be specified (see Figure 10.1). The plugin itself uses first Lucene to check which ontologies match the query, and then retrieves the overall ratings for these ontologies from the TS-ORS. The ontologies are then ranked in the result list according to the overall ratings. The overall rating can be seen in brackets, while we also display its value in terms of stars. A sample result

[1]`http://watson.kmi.open.ac.uk/editor_plugins.html`, last checked on 24.11.2010

list with the stars and overall ratings can be seen in Figure 10.2. The initial version of the Cupboard Plugin, which was also used in our user experiment (see Chapter 14) also provided an additional feature when adding axioms from ontologies in Cupboard. It allowed the user to import multiple axioms at once by choosing either to add all super-classes of a class, or to add the whole sub-branch of a class. We thought this shortcut can save time because some users otherwise keep on navigating the hierarchy with the plugin to add the remaining super-classes or subclasses one by one.



Cupboard Preferences

SameAs

Create sameAs links between ontology entities and the entities of reused statements [✓ Don't create any link with external ontologies]

EquivalentClass

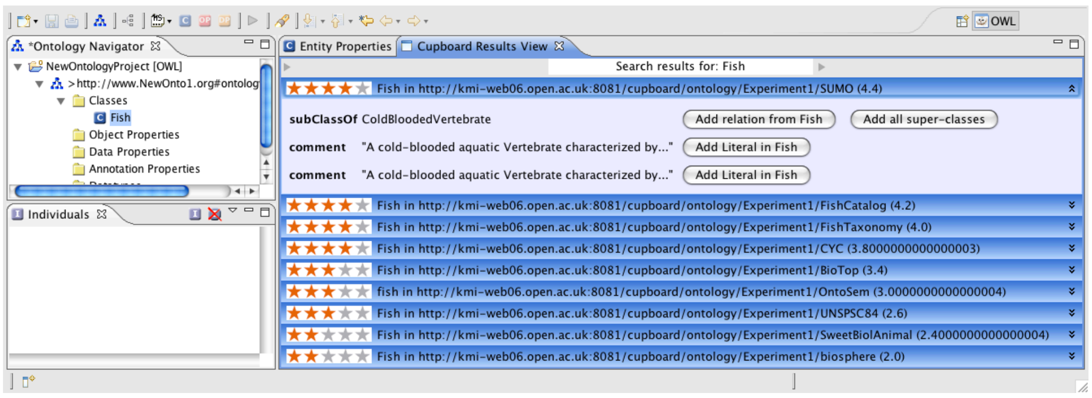
Create equivalentClass instead of sameAs for classes [✓ Use sameAs also for classes]

Ontology Space: Experiment1

Figure 10.1: The Preference Panel of the Cupboard NeOn Toolkit Plugin.

10.3 Updated Watson Plugin

The latest version of the Watson Plugin is build as a hybrid plugin, which can either connect to Watson or to Cupboard. In the preference panel (see Figure 10.3) the user



Search results for: Fish

★★★★☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/SUMO (4.4)

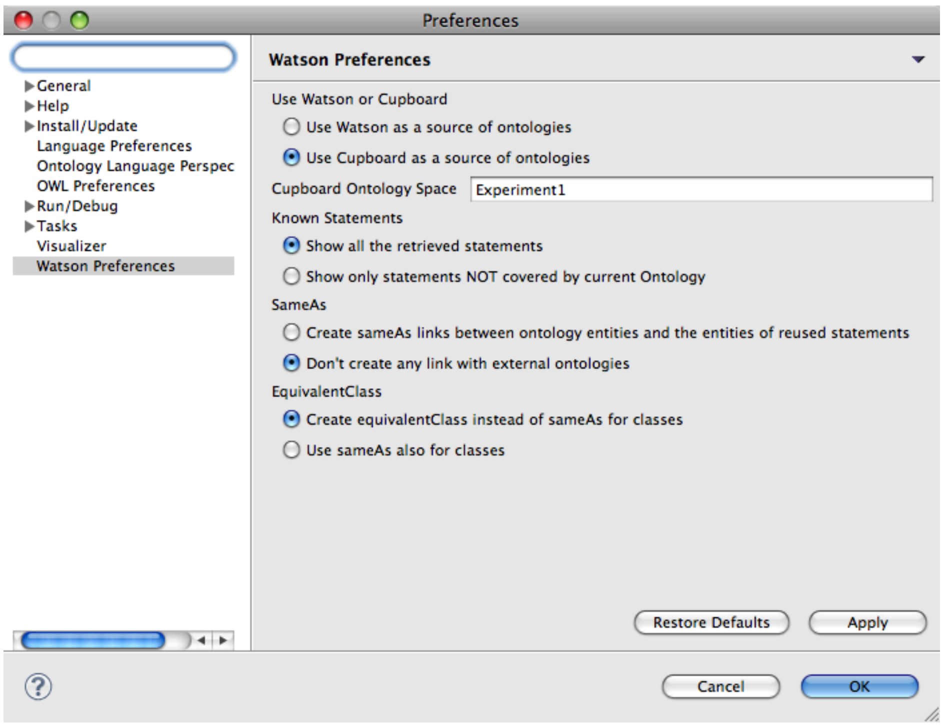
subClassOf ColdBloodedVertebrate
comment "A cold-blooded aquatic Vertebrate characterized by..."
comment "A cold-blooded aquatic Vertebrate characterized by..."

★★★★☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/FishCatalog (4.2)
★★★★☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/FishTaxonomy (4.0)
★★★★☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/CYC (3.8000000000000003)
★★★☆☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/BioTop (3.4)
★★★☆☆ fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/OntoSem (3.0000000000000004)
★★★☆☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/UNSPSC84 (2.6)
★★☆☆☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/SweetBiolAnimal (2.4000000000000004)
★★☆☆☆ Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/biosphere (2.0)

Figure 10.2: The Result View of the Cupboard NeOn Toolkit Plugin.

can choose whether to use Watson or Cupboard as a source of ontologies. In case Cupboard is chosen, the user can further specify whether results should be restricted to a special ontology space, or all ontology spaces should be searched (which is the default behavior if no ontology space is provided).

The result view of the updated Watson Plugin (see Figure 10.4) is similar to the Cupboard Plugin, but does not allow to add more than one axiom at a time. We are currently still investigating whether this feature should be included in future versions of the plugin or not. We are working on an extended version allowing bidirectional communication with Cupboard, meaning ontologies can also be added or reviewed from within the NeOn Toolkit.



Watson Preferences

Use Watson or Cupboard

( ) Use Watson as a source of ontologies

[✓ Use Cupboard as a source of ontologies]

Cupboard Ontology Space: Experiment1

Known Statements

[✓ Show all the retrieved statements]

( ) Show only statements NOT covered by current Ontology

SameAs

( ) Create sameAs links between ontology entities and the entities of reused statements

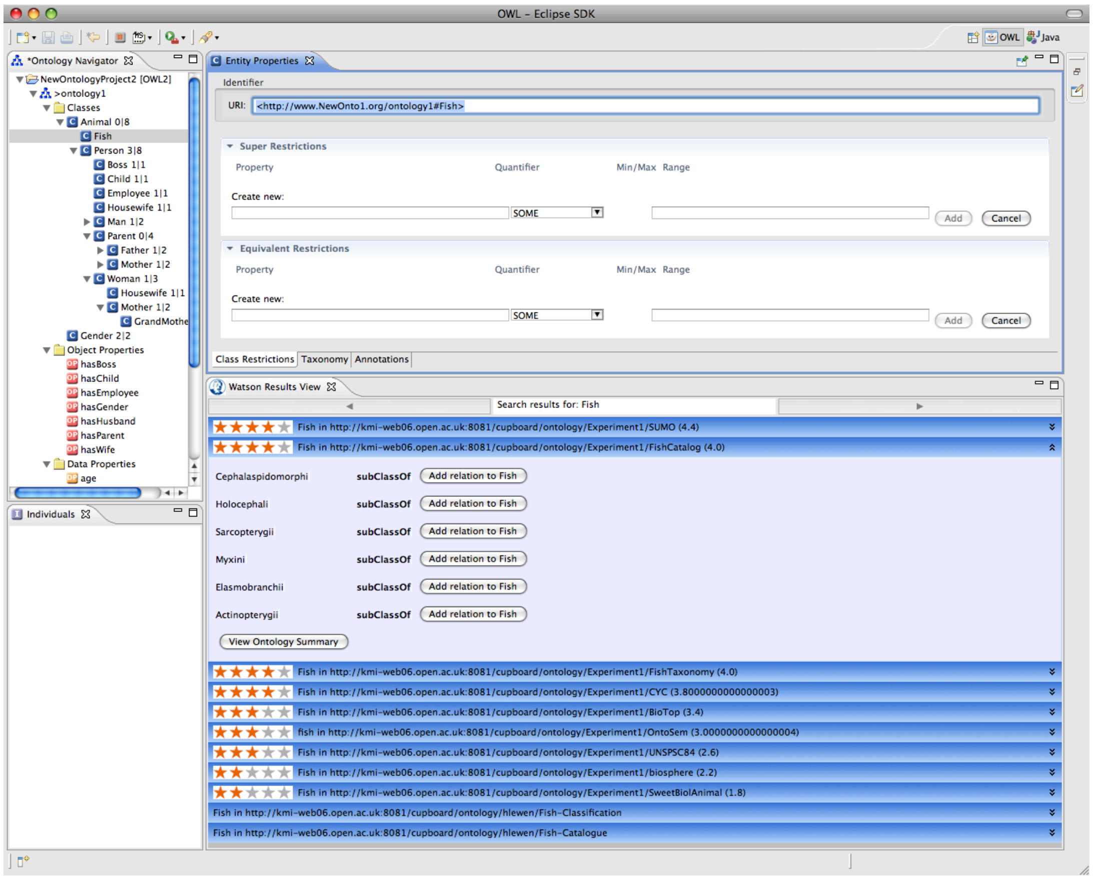
[✓ Don't create any link with external ontologies]

EquivalentClass

[✓ Create equivalentClass instead of sameAs for classes]

( ) Use sameAs also for classes

Figure 10.3: The Watson Plugin's Preference Panel.



URI: <http://www.NewOnto1.org/ontology1#Fish>

Search results for: Fish

Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/SUMO (4.4)
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/FishCatalog (4.0)

Cephalaspidomorphi	subClassOf	Add relation to Fish
Holocephali	subClassOf	Add relation to Fish
Sarcopterygii	subClassOf	Add relation to Fish
Myxini	subClassOf	Add relation to Fish
Elasmobranchii	subClassOf	Add relation to Fish
Actinopterygii	subClassOf	Add relation to Fish

Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/FishTaxonomy (4.0)
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/CYC (3.8000000000000003)
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/BioTop (3.4)
fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/OntoSem (3.0000000000000004)
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/UNSPSC84 (2.6)
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/biosphere (2.2)
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/Experiment1/SweetBiolAnimal (1.8)
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/hlewen/Fish-Classification
Fish in http://kmi-web06.open.ac.uk:8081/cupboard/ontology/hlewen/Fish-Catalogue

Figure 10.4: A result list within the Watson Plugin's.

Chapter 11

Exporting Ratings and Trust Information

Since reviews are public in the TS-ORS, they can be crawled by other repositories. To make crawling unnecessary and to facilitate collaboration between repositories, it is important that data from the repositories can be exchanged. In order for collaboration to work it is important that repositories can agree on a format for data exchange. In this chapter we want to discuss possible ways of exchanging rating and trust information between ontology repositories. We start by discussing the importance of user privacy in Section 11.1. The requirements for the export of data are defined in Section 11.2. The chapter concludes with an overview of ontologies for exporting ratings and trust information in Section 11.3.

11.1 User Privacy

An important aspect to take into account when offering data to other applications is user privacy. Some of the data in the system is not visible and has been entered only for internal use. One example is the email address of a user, which normally is not displayed to avoid email spam. Trust information is also private. Also users should only see trust statements they made, and not trust statements other users have made. That means, that if user A trusts user B, this trust statement is visible to user A, but whether user B trusts another user or A should not be disclosed to user A. User A should not gain information on who trusts or distrusts him.

Within a system, a verified email address or an openID[1] are possible ways to identify a user. Also, an email address or openID can be the same among multiple repositories, while the user name could already be taken in another repository. Therefore, when exchanging review information, the email address or openID can be used to identify a user. Given that the email address can be considered private information, we suggest instead using a secure-hash (e.g., SHA-512 (NIST, 2002)) of the email address for data exchange. While the email address cannot be inferred from the hash, a system

[1] http://openid.net/developers/specs/, last checked on 24.11.2010

also having the email address in the database can compare the hash against hashes of email addresses in the system. FOAF (Brickley and Miller, 09 August 2010) allows to specify agents by the sha1sum of their email address, or by openID.

In terms of trust, a logged in user should be allowed to export the trust information he entered into the system for use within another system. Within the export, users can be identified using a secure hash of either their email address or openID.

11.2 Requirements for Data Export

If a rating should be exchanged, the minimal data needed is the URI of the ontology, the name of the aspect of the ontology, the identity of the user who has made the rating, the rating value and the review explaining the rating.

For the trust export, the identity of the two users affected by the trust statement have to be known, and the scope of the trust statement (in case of a regular trust statement the URI of the ontology and the name of the aspect covered by the trust statement). In case a meta-trust statement is exported, the type of the meta-trust statement and its scope have to be contained.

11.3 Ontologies for Exporting Ratings and Trust Information

The rating ontology[2] (Longo and Sciuto, 2007) provides a good basis for formalizing ratings. It provides enough expressivity to serve as a container for reviews from the old ORS-Model. The rating ontology reuses the FOAF definition of an agent, it is thus possible to protect user privacy by exclusively using sha1sum hashes of their email address.

The ontology currently does not allow to add reviews for certain aspects of a resource, only for the complete resource. This functionality can easily be added by extending the rating ontology by adding a class *Aspect* which has the different aspects as individuals. It is important to then also link the review to the aspect as well, this can be done by adding an object property *coversAspect* with domain *Review* and range *Aspect* .

This small change is sufficient to express TS-ORS ratings using the rating ontology.

[2]http://purl.org/stuff/rev, last checked on 24.11.2010

11.3.1 Example Review

Let's assume we want to export the reviews of Enrico on the AKTPortal ontology from Cupboard[3] (see Figure 9.2) using the extended rating ontology. First, we would create an individual of type *foaf:person* with data property assertion *accountName* Enrico and an object property assertion *mboxsha1sum*[4] with the sha1 hash of the email address. Then we would create an instance of *http://www.w3.org/2000/01/rdf-schema#Resource* with the URI of the ontology covered by the rating, in our case http://kmi-web06.open.ac.uk:8081/cupboard/ontology/enrico/AKTPortal. Then we could create five instances of http://purl.org/stuff/rev#Review, one for each review (all five aspects are reviewed). Then the reviews have to be filled with data. For that, for each review an object property assertion *reviewer* and *coversAspect* have to be added. The rating value is then added to the review using a data property assertion *rating* and the review text can be added using the data property assertion *text*. Additionally, the created reviews have to be linked to the ontology. Therefore, for the AKTPortal ontology, 5 object property assertions *has review* are created linking to the reviews.

11.3.2 Trust Exchange

Jennifer Golbeck has created a small trust ontology[5] which provides an example of how trust can be modeled. The information we have to represent in a trust ontology are the two agents (which we can identify as foaf:person with the mboxsha1sum property assertion), the rating value and the scope of the trust statement. For that, the type of trust statement, i.e., W, W_O, W_X, and W_{XO} have to be linked with a trust statement. Depending on the signature of the meta-trust statements, the trust assertions also have to be linked to the URI of the ontology and the aspect they cover. The trust exchange can be realized as a separate ontology or integrated into the rating ontology.

As we said, a user should have the possibility to retrieve personal trust statements from the repository after being logged in, for use in other repositories. The export should not contain information not entered into the system by the user.

11.3.3 Exchange Between Ontology Repositories

While it is not necessary to use ontologies to exchange data, they can provide a schema that repositories can agree upon. When relying on the schema, data can be exchanged between ontology repositories.

[3]http://kmi-web06.open.ac.uk:8081/cupboard/fullreviews?ontouri=http://kmi-web06.open.ac.uk:8081/cupboard/ontology/enrico/AKTPortal, last checked on 24.11.2010

[4]The correct value here would be 28a0f82609671f47d811e6bee865afb23abfb8db

[5]http://trust.mindswap.org/trustOnt.shtml, last checked on 24.11.2010

Often, when exchanging data, the receiving repository will not be able to map all users or ontologies to data in their system. In these cases the data can be stored for future reference. For example, let us assume the sha1 hash of an email address cannot be found in the database of an importing repository. Then, the provided *accountName* can be displayed, and the hash used for internal reference. In case a reviewer with an email address matching the hash later registers, the reviews can automatically be associated correctly.

Ontology repositories should also contain the possibility to import trust data that has been exported from other repositories. In case the system can identify the trusted users using the sha1 hashes of their email address, a trust relationship can be added to the system. Otherwise, the trust statement can be added when the trusted user registers with the repository.

Part IV

Evaluation

Chapter 12

Complexity Analysis and Benchmark

In this chapter we first present a complexity analysis of our TS-ORS in Section 12.1, followed by a performance benchmark in Section 12.2. Parts of this chapter are based on (Lewen, 2009b; Lewen, 2009c; Sabou *et al.*, 2009; Angeletou *et al.*, 2010).

12.1 Complexity Analysis

In order to understand how the runtime of an algorithm changes with increasing input data, it is important to analyze the algorithm's complexity. The Big-O notation (Bachmann, 1894; Knuth, 1976) has become a standard way to express the complexity of algorithms. In the Big-O notation, the complexity class is determined by the term with the highest growth rate. Constants and factors are omitted. For example, let us assume that $f(x) = 4 * x^4 + x^3 + 4$. Then $O(f) = x^4$. In case an algorithm employs several types of computations algorithms, the one with the highest complexity will determine the overall complexity of the algorithm. For our algorithms, we can therefore analyze each operation of the algorithms separately and then see in which complexity class the combined results lie. Since constants are irrelevant for the complexity analysis, we do not try to provide the exact number of times an operation is performed, but concentrate on the complexity of the operation performed. Usually one input parameter is chosen for the complexity analysis, since the result is easier to understand than trying to combine all variables.

In our system, we have a number of variables. The number of ontologies, the number of aspects, the number of users, the number of reviews, and the number of trust statements. We will consider the number of users as our input parameter, since it is the one having the largest impact on the complexity.

We distinguish between so called offline computations, i.e., computations that are not required at runtime (they do not depend on user input), and can be triggered at fixed points in time, and runtime computations, which have to be constantly executed at runtime and depend on user parameters.

12.1.1 Offline Computation Complexity

It is important to understand that the offline computations (computation of *TrustRank*, *DistrustRank* and the local trust matrix **F** with its interpretation matrix **I**) have to be performed for each ontology–aspect combination, that means they are performed exactly $\#ontologies * \#aspects$ times. The symbol # stands for "number of".

During the offline computations, the following operations are used:

- Matrix addition (in $O(n^2)$)
- Matrix subtraction (in $O(n^2)$)
- Matrix transposition (in $O(n^2)$)
- Matrix multiplication (in $O(n^3)$)[1]
- Matrix scalar multiplication (in $O(n^2)$)
- The majority rounding (in $O(n^3)$) which consists of n times the following:
  - Sorting (in $O(n * \log n)$)
  - Interpreting Values from **F** (in $O(n^2)$)
- *TrustRank* computation (in $O(n^2)$)
- *DistrustRank* computation (in $O(n^2)$)

Since the performed database operations all have a complexity below $O(n^3)$, the overall complexity of the computations is in $O(n^3)$. As long as there are not too many users, there is no problem computing the trust scores for every ontology–aspect combination separately. If the time taken to compute the trust scores is too long for the needs of the application, the number of times the computations are performed can be reduced by grouping the available trust information and only performing the computations where they provide the most benefit. Possible groupings could for example be by aspect or by ontologies of a special area.

The optimization technique we propose involves only computing the trust scores for users who are connected to the WOT for a given ontology–aspect combination. In most cases, the number of users should be only the upper bound for n, with real values being much lower. Imagine for example Amazon.com with its millions of users. For a given product, the number of people having reviewed or trusting reviewers are way smaller than the total number of users. The computations would thus not include the majority of users who will not affect the computation outcome, but focus on the users who said whether they trust or distrust a reviewer, or, in the case of Amazon.com, who said whether for them a review was helpful or not.

[1]We are aware that other methods for matrix multiplication exist that have a lower complexity, but their overhead is too big to employ them in our application.

12.1.2 Runtime Complexity

For the runtime complexity we have to distinguish two tasks the TS-ORS performs at runtime: Ranking the reviews for a given ontology–aspect combination, and computing an overall rating for an ontology. We also have to distinguish the cases that the user is identified (and local trust can be used), and the case where the user is unknown (and global trust has to be used).

Ranking of Reviews

For this task the system is given as input parameters the ontology and aspect for which the reviews have to be ranked, and the user who is requesting the information (if available). Also the combination of trust and distrust can be influenced by a parameter. For the result based on the global reviews, the method basically just performs one database query and then provides the ordered reviews as output. Its runtime is dependent on the number of reviews that exist for this ontology-aspect combination. Within the database query, the results are ordered. So with n being the number of reviews, the complexity of the retrieval is in $O(n * \log n)$ which is mainly due to the sorting needed. In case the user is known, more database queries are performed, but the complexity stays in $O(n * \log n)$ with n being the number of existing reviews. Since there cannot be more than 1 review per user for each ontology–aspect combination, the number of users is an upper bound for the number of reviews. Most of the times, the number of reviews will be far smaller than the number of users.

Overall Rating of Ontologies

In case the overall rating of an ontology has to be computed, the complexity is the same as for the ranking of the reviews, since for all aspects of an ontology, the top-n reviews have to be retrieved. Since during this process the results are sorted in the database, the complexity is $O(n * \log n)$ for both retrieval based on global and on local trust, with n being the number of reviews. Since the number of aspects is a constant factor in any installation of the system, it does not influence the complexity. Again, the number of users is an upper bound for n.

12.2 System Benchmarking

Because the TS-ORS delivers overall ratings for ontologies to Cupboard which are then used to rank ontologies in a result set, time is crucial. Users nowadays are accustomed to receive search results within a fraction of a second, thanks to Google. In order to see how quickly our system can deliver the needed results, we have decided to run a benchmark with synthetic data. Within the benchmark, we did not only check how the system handles an increasing number of users, reviews and trust statement, but also how parallelizing the offline computations affects the computation time. For all of the following benchmarks, we set the number of ontologies to 4 and the number of aspects to 5. Therefore, all results reported below are not normalized to represent the execution time for one ontology or one ontology-aspect combination, but are reported as the combined time for all 20 ontology–aspect combinations. We chose to use 5 aspects because in Cupboard we also use 5 aspects. As for the ontologies we could have used just one ontology, since the number of ontologies affects the execution time linearly, but we chose to use 4 to get a more balanced results.

12.2.1 Setup

We created a java application that automatically runs all the benchmarks, and writes the results into text files. The results of the benchmark can be replicated using the code we used to run the benchmark.[2]

The application contains apart from the classes covered in Chapter 8 a method called Evaluation, which is responsible for coordinating the benchmarking. The method works by first filling the *user*, *ontologies*, *ratings*, *trust*, and *metatrust* tables in the database with randomly generated data, based on passed-on parameters. We adapt parameters to generate different scenarios. Adapted parameters include the number of users, ontologies, reviews per ontology–aspect combination, trust and meta-trust statements as well as the percentage of trust statement versus distrust statement. It is ensured that no duplicate/contradictory data is generated during the process.

After the data is created, the offline computations are run to materialize the meta-trust, compute the *TrustRank*, *DistrustRank* and the local trust matrix **F** with its interpretation matrix **I** for all 20 ontology–aspect combinations. These computations also have to be performed from time to time in running applications to ensure that the application is working with the most current data for the ranking and overall rating computation. When running the offline computations, the following steps are executed:

1. Materialization of the *temp* database to allow faster access.
2. Creating global reflexive meta-trust statements for every user (thus ensuring every reviewer trusts his own review).
3. Materialization of meta-trust and meta-distrust statements based on Algorithm 1.
4. Computation of *TrustRank*, *DistrustRank* and the local trust matrix **F** with its interpretation matrix **I** for all ontology–aspect combinations in the database

[2]The files needed can be found at: `http://people.aifb.kit.edu/hle/thesis/benchmark/`, last checked on 24.11.2010.

(since these tasks rely more on CPU power than on database interaction, the computations are implemented to allow making use of multi-threading, which is beneficial in case the CPU has more than one core).

5. Indices are added to the *tempglobaltrust* and *templocaltrust* tables.
6. The tables are renamed: *temp* becomes *runtimetemp*, *tempglobaltrust* becomes *globaltrust*, *templocaltrust* becomes *localtrust*. This way, the system can stay operational while the trust scores are re-computed.

Once this data is computed, the runtime tests are performed. We measure how long it takes to retrieve an overall rating for all 4 ontologies, both using global (based on *tempglobaltrust* and *templocaltrust*) and local (primarily based on **F** and **I**) trust. Each overall rating is retrieved 500 times based on a varying number of reviews (using only the top review, the top 3 reviews, and all reviews). We measure the duration of each execution and report the minimum, maximum and average duration. It is important to see how database caches can be exploited to minimize the latency at runtime. The maximum time can be usually measured when the rating is first retrieved and no result are cached, while for the rest of the runs the results of the database queries should still be cached. We can exploit the query result cache for short response time during runtime by querying all ontologies in the system once after re-computation using the default parameters, so that results are cached and runtime database access can be minimized.

We also measure how long it takes to retrieve all reviews for the 20 ontology–aspect combinations based on both global trust ranking and local trust ranking.

12.2.2 Execution

We have performed the benchmark on a 2.40 GHz Intel Core 2 Quad Q6600 processor with 8GB 800 MHz DDR2 SDRAM running Ubuntu 10.04.1 LTS with Ubuntu Kernel 2.6.32-24 x86_64. The hard disk used is a Samsung HD103UJ (1TB, 7200 RPM, 32MB Cache, avg. seek time 8.9). The java version used was OpenJDK Runtime Environment (IcedTea6 1.8.1) 6b18-1.8.1-0ubuntu11. We used the MySQL 5.1.49-linux x86_64-GLIBC23 database and the MySQL J-Connector version 5.1.13.

In order to receive a detailled result, we did not only vary the number of users (100, 250, 500, 1000, 2500), but also the amount of reviews and trust statements available, resulting in 15 different benchmark configurations. During the generation of the test data, for each different user group size, we had one setting which had 10% of the users review each ontology–aspect combination and 10% of the users then trusting or distrusting the reviewer for these reviews, one with a 50%–50% distribution and a worst case scenario (100%–100%). Worst case means that every users reviews every ontology–aspect combination, and also every user then states their trust or distrust on every other user for every ontology–aspect combination. In a realistic setting, the distribution is likely way less than 10%–10%. We furthermore assumed that the trust to distrust ration was 70% to 30%, meaning 70% of the trust data in the system are trust statements, and 30% are distrust statements. For the meta-trust generation, we have fixed the percentage of W_{OX}, W_O and W_X meta-trust statements to 20% per user for all runs, that means each users makes meta-trust statements covering 20% of the other users. The test data was regenerated between all runs, to prevent caching effects in between runs.

To get an idea of the size of the data handled during the benchmark, we explain our distribution idea once again, this time with an example. Let us look at the 2500 user scenario with 100% rating and trust coverage. For our 20 ontology–aspect combinations, the *rating* table produced has $\#users * \#ontologies * \#aspects = |U| * |O| * |X| = 2500 * 4 * 5 = 50,000$ entries. Since every user states trust or distrust to all other users for all ontology–aspect combination, the *trust* table produced has $\#users * \#users * \#ontologies * \#aspects = |U| * |U| * |O| * |X| = 2500 * 2500 * 4 * 5 = 125,000,000$ entries (we assume users trust themselves for their reviews). The resulting *runtimetemp* table has a data length of 2.44 GB and an index length of 4.44 GB.

In general, one can compute the number of entries created as such: Let m be the percentage of users rating an ontology, and n be the percentage of users then trusting the reviewers for their respective reviews. Then the number of entries in *rating* can be computed as $\#users*m*\#ontologies*\#aspects = |U|*m*|O|*|X|$. For 100 users and 10% ratings, the number of entries thus would be $100*0.1*4*5 = 200$. The number of entries in *trust* can be computed as $\#users * m * \#ontologies * \#aspects * \#users * n = |U| * m * |O| * |X| * |U| * n$. For 100 users, 10% ratings, and 10% trust, the number of entries in *trust* would be $100 * 0.1 * 4 * 5 * 100 * 0.1 = 2000$.

For the meta-trust statement, we always use 20%, so the resulting number of meta-trust statements can be computed as $\#users * \#users * 0.2 * 3 = |U| * |U| * 0.2 * 3$. The factor 3 represents the meta-trust statements of form W_O, W_X, and W_{OX}. For each of the meta-trust statements, we take each user and then randomly select 20% of the other users to trust or distrust. In case the statement covers ontologies or aspects, the ontology or aspect covered is randomly chosen from O or X.

12.2.3 Results

The results of the meta-trust materialization benchmark can be found in Figure 12.1. One the x-axis, the three different benchmark scenarios can be found ((10%/50%/100%) of ontologies rated by each user, (10%/50%/100%) of reviewers trusted by each user, and 20% of other users meta-trusted by each user), each with 100, 250, 500, 1000, and 2500 users. The total time taken for each meta-trust materialization is broken down into the time for materializing W_O, W_X, and W_{OX}. It is important to note that the y-axis has a logarithmic scale, that means that the relation between the duration of each of the 3 steps is easily misinterpreted when looking at the graph. For that reason we show the absolute execution time inside the graph as well.

The results of the trust computation benchmarks can be found in Figures 12.2 (for the 10%10%20% scenario), 12.3 (for the 50%50%20% scenario), and 12.4 (for the 100%100%20% scenario). The total time taken for the computation is broken down into duration of the *TrustRank* computation, the *DistrustRank* computation, the computation of **F** and **I** and the database input/output (IO). The y-axis also has a logarithmic scale. In order to get a better feeling for how the time is distributed among the different parts of the computation, Figure 12.5 provides the distribution of time in percent for the results also found in Figure 12.4.

The result of the overall rating benchmarks can be found in Figures 12.6 (based on the top review), 12.7 (based on the top 3 reviews), and 12.8 (based on all reviews). The y-axis has a logarithmic scale, the time is shown in milliseconds for these graphs. Each figure contains one graph for the overall rating benchmark based on global trust, and one graph for the benchmark based on local trust. For each configuration, the maximum duration, the minimum duration and the average duration over the 500 runs are shown. The results are the combined time for retrieving the overall ratings for 4 ontologies.

The result of the review ranking benchmark can be found in Figure 12.9. The y-axis has a logarithmic scale and the time is shown in milliseconds. There are two figures, one for the benchmark results based on global trust, and one for the benchmark results based on local trust. For each configuration, the maximum duration, the minimum duration and the average duration over the 500 runs are shown. The results are the combined time for retrieving the review ranking for all 20 ontology–aspect combinations.

12.2.4 Result Analysis

We will now analyze the results of the benchmarks and try to explain the system behavior.

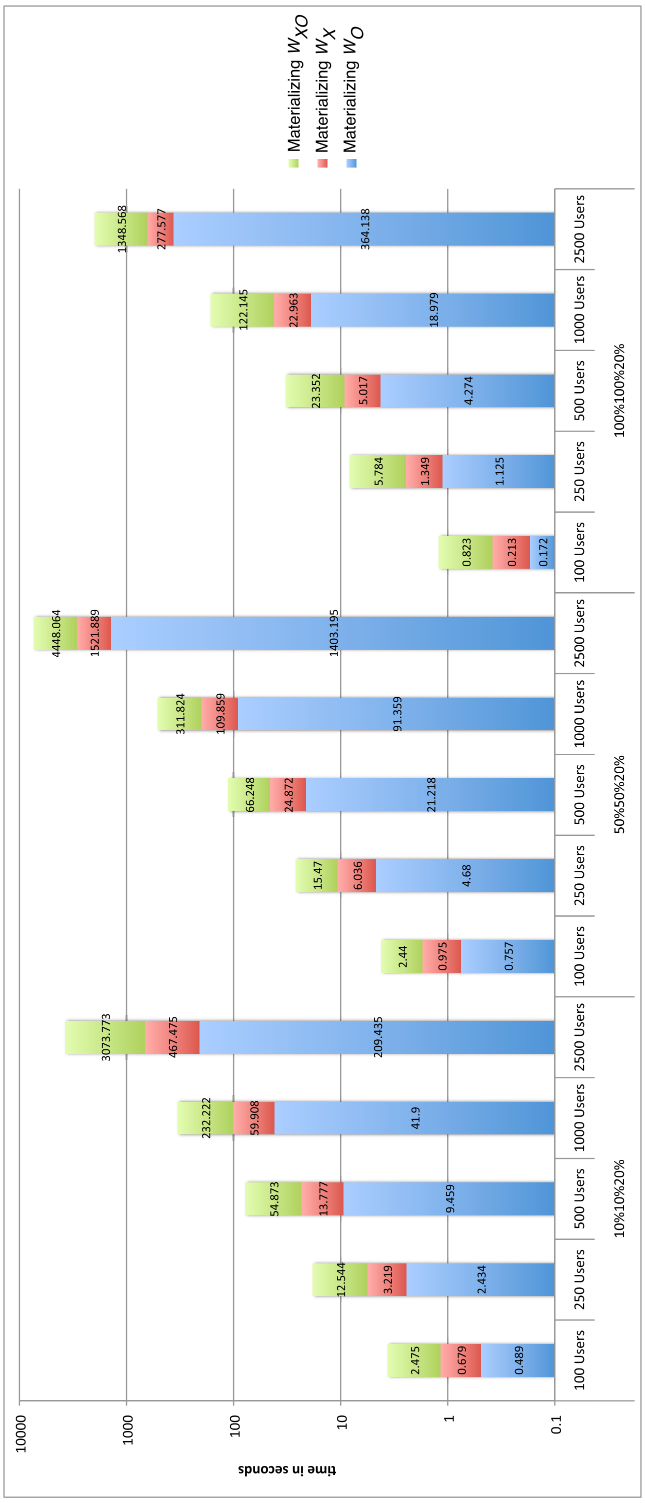


Figure 12.1: Result of Meta-trust Materialization Benchmark.

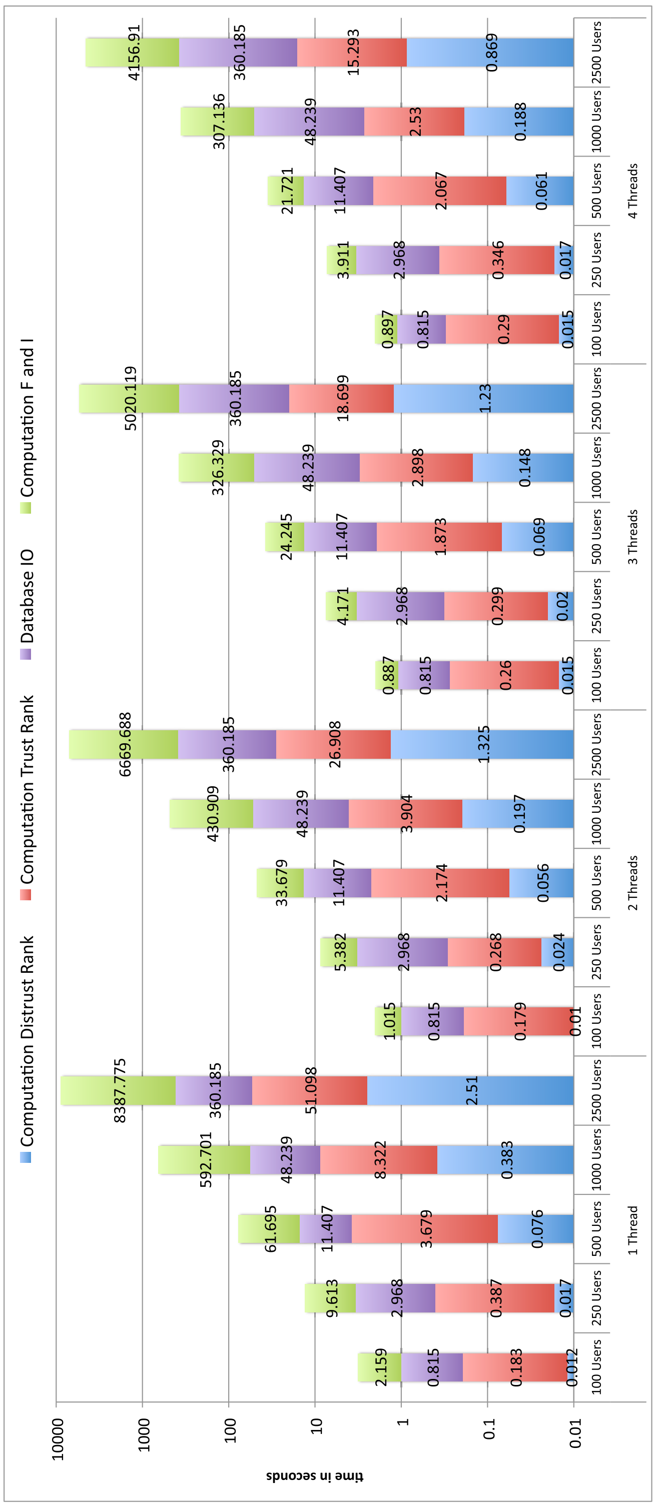


Figure 12.2: Result of Trust Computation for the 10%10%20% Scenario.

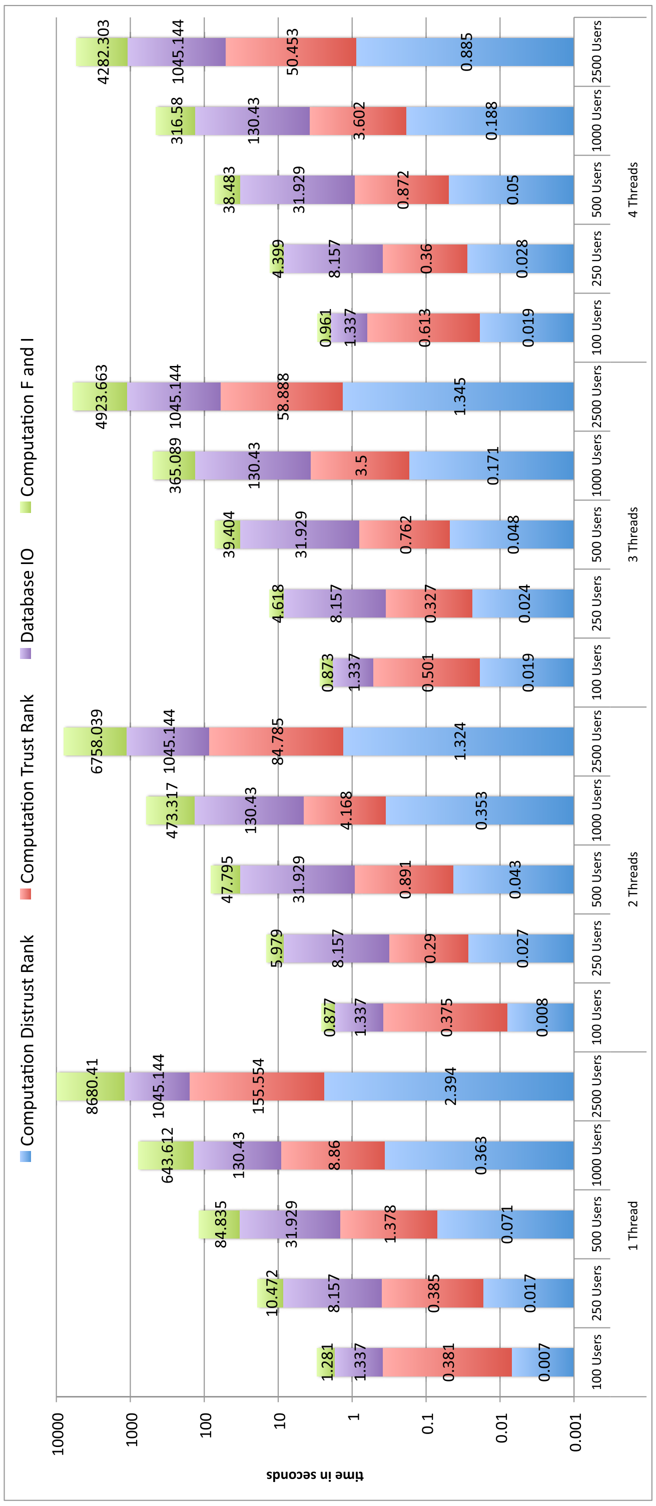


Figure 12.3: Result of Trust Computation for the 50%50%20% Scenario.

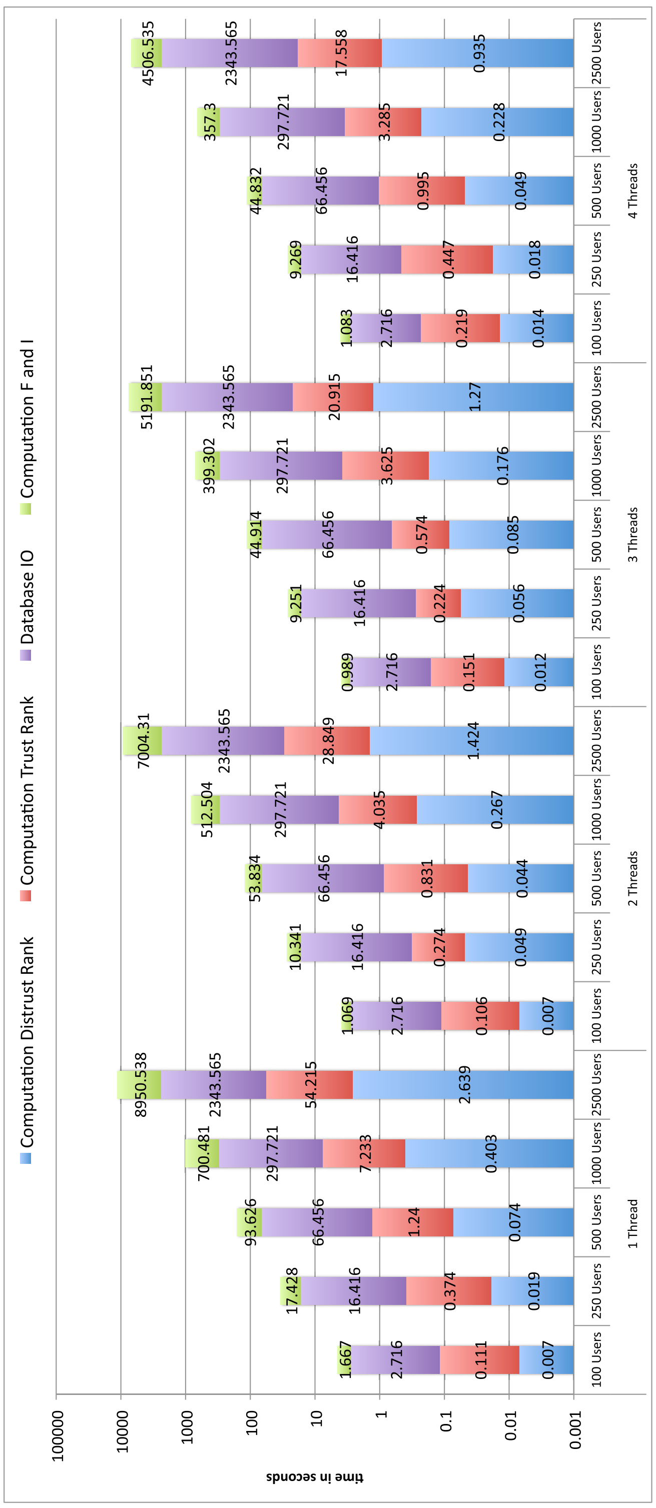


Figure 12.4: Result of Trust Computation for the 100%100%20% Scenario.

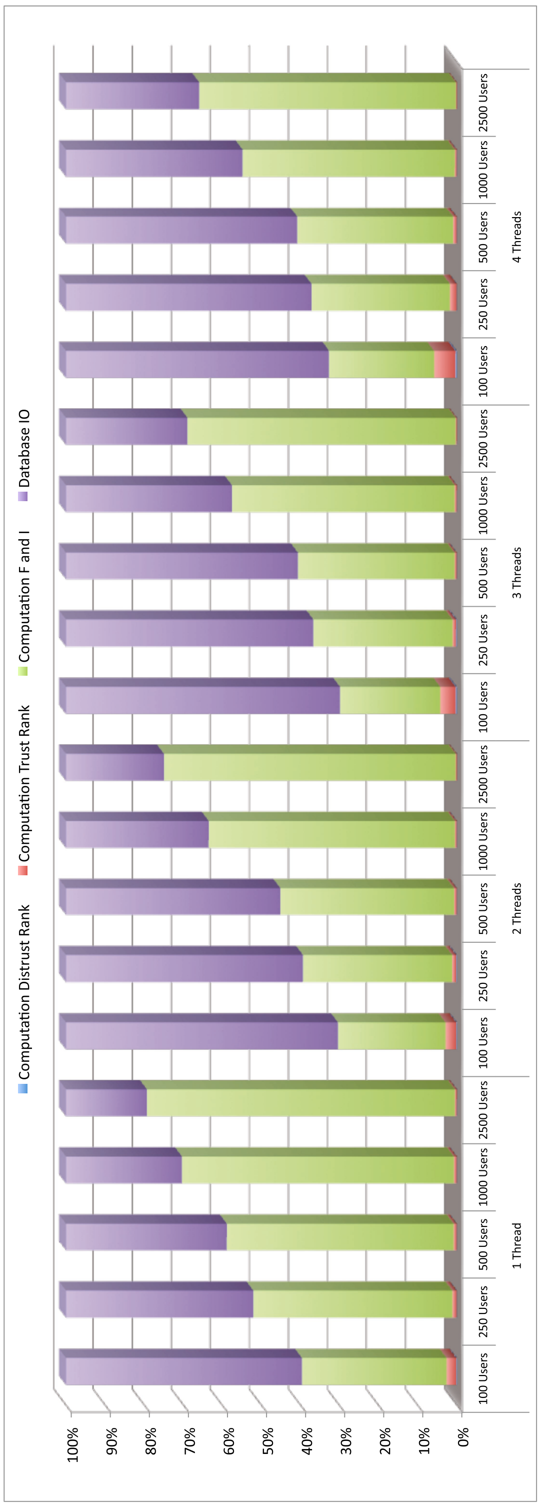


Figure 12.5: Distribution of Time to Different Tasks in the 100%100%20% Scenario.

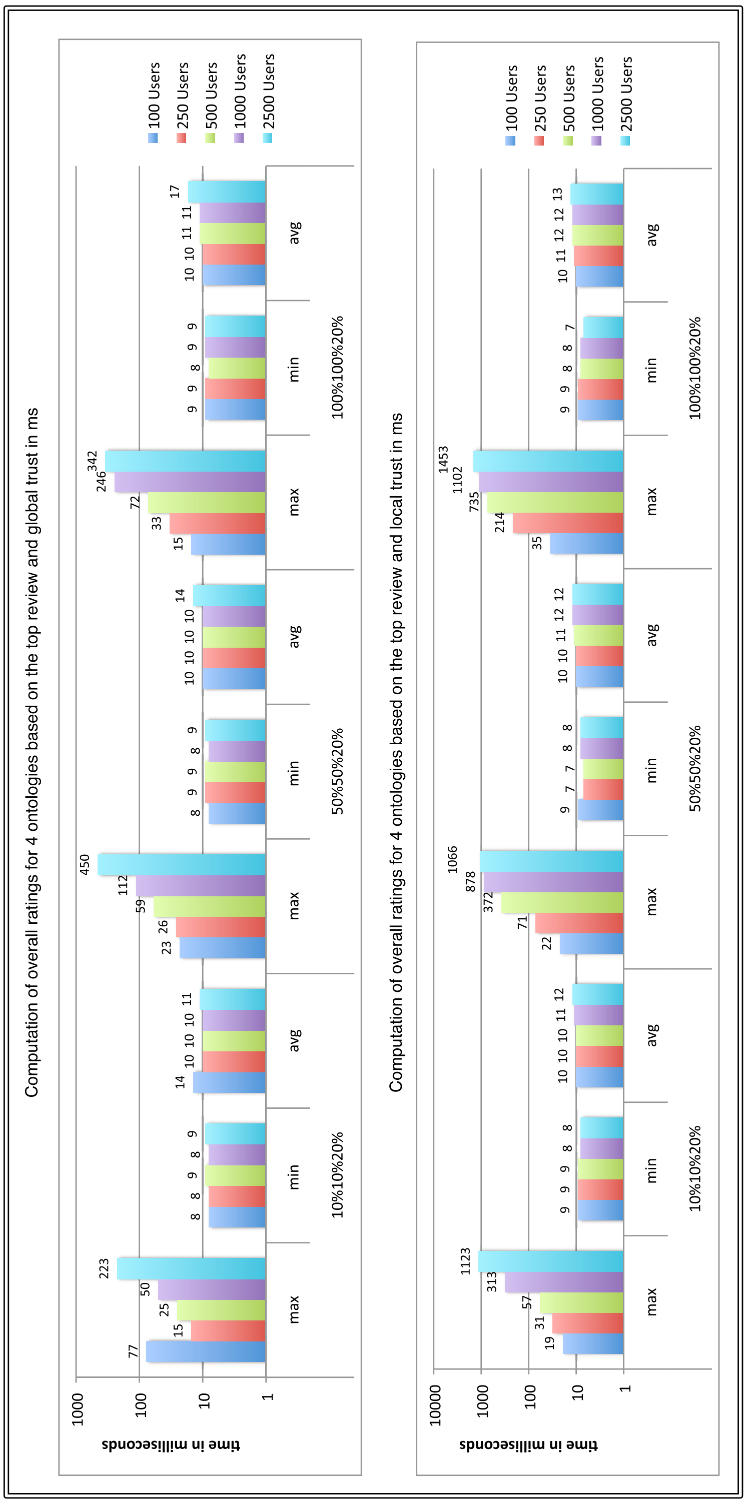


Figure 12.6: Result of Overall Rating Benchmark (Top Review) with 500 runs.

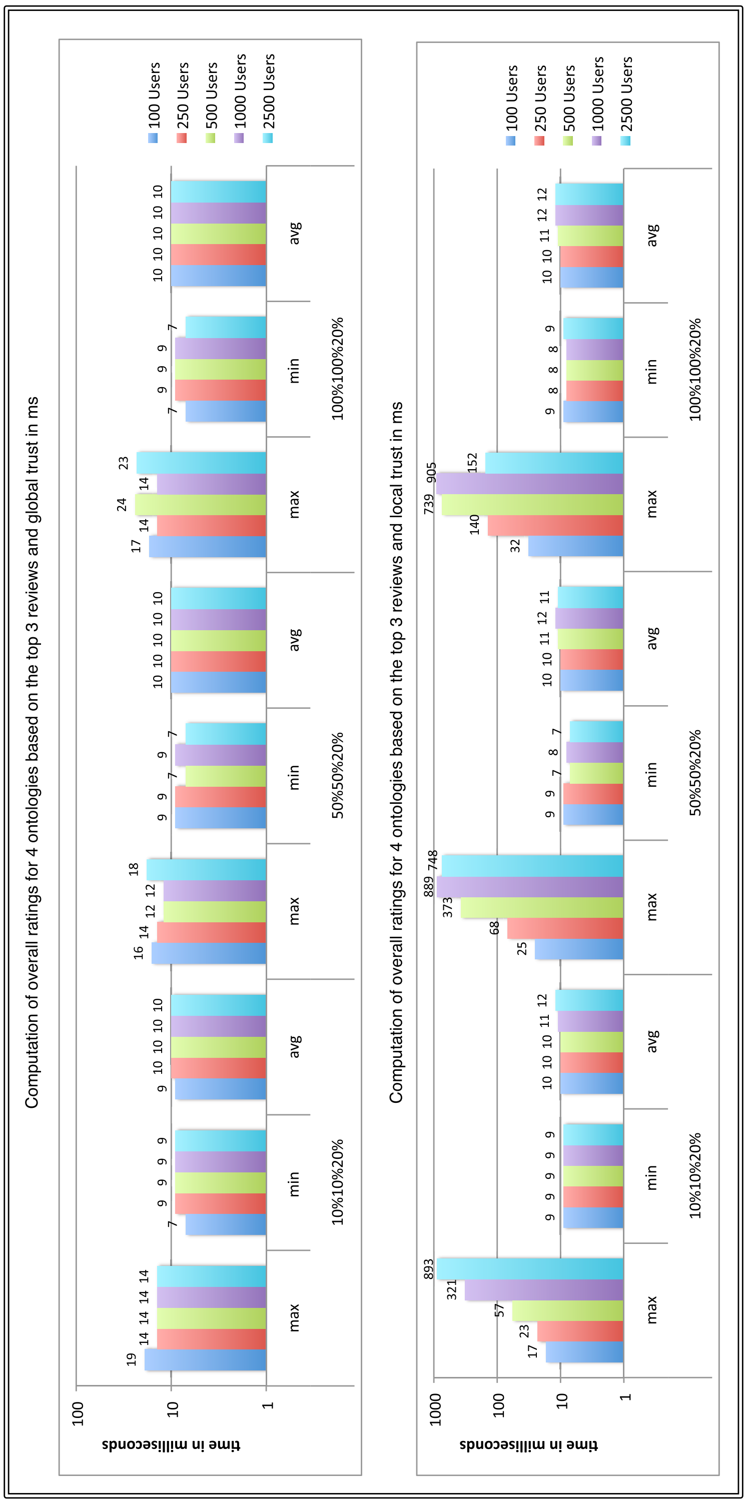


Figure 12.7: Result of Overall Rating Benchmark (Top 3 Reviews) with 500 runs.

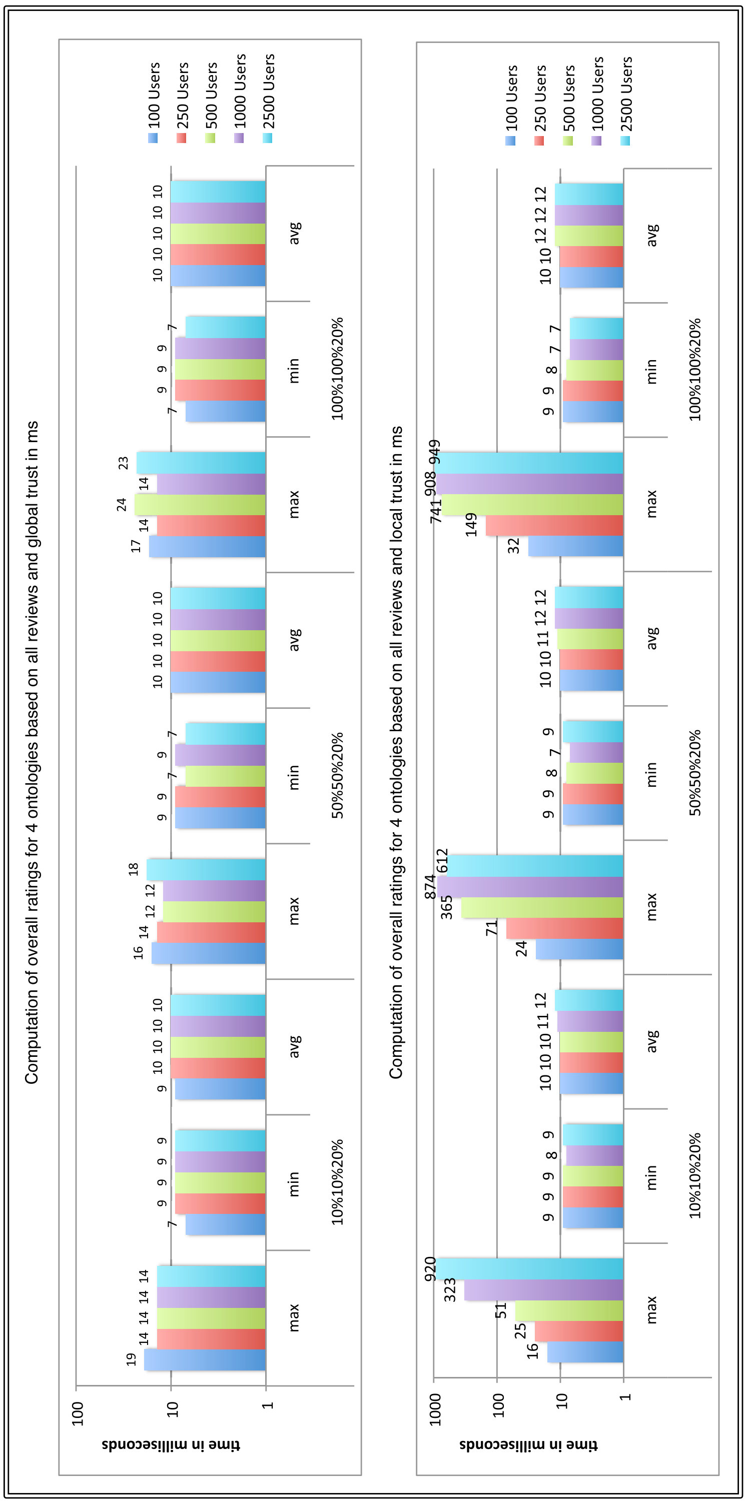


Figure 12.8: Result of Overall Rating Benchmark (All Reviews) with 500 runs.

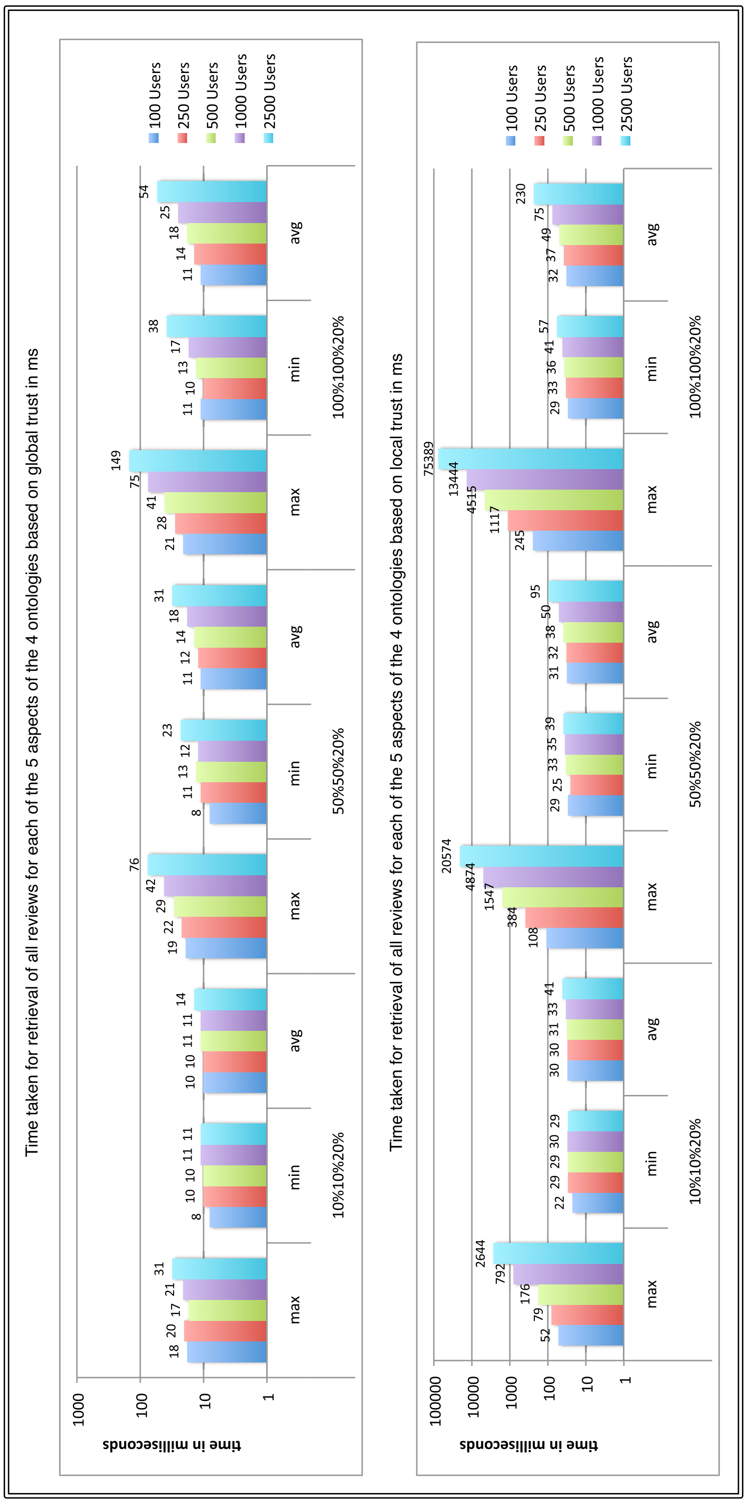


Figure 12.9: Result of Review Ranking Benchmark with 500 runs.

Meta-trust Materialization

While the general algorithm of meta-trust materialization is described in Algorithm 1, we now focus more on the implementational details. In order to implement the materialization as fast as possible, the materialization is entirely performed within the database. For each type of meta-trust statement, the respective statements from the database are retrieved and stored in a temporary table, already with their potential scope. For example, if a global meta-trust statement is materialized, the temporary table contains trust statements for all ontologies and aspects. Then we delete from the temporary table all entries for which a trust or distrust statement between these users for this ontology–aspect combination already exists. The remaining values are then added to the *temp* database which stores the trust information used for the computations. This way it is ensured, that no more specific trust statement is overwritten by a more general meta-trust statement.

Since each W_O statement covers all 4 ontologies, and each W_X statement covers all 5 aspects in our benchmarking setting, a W_X statement takes longer to materialize than a W_O statement. In case the setting were different, for example having more ontologies than aspects, the materialization of W_O statements would take longer.

The results found in Figure 12.1 indicate what was expected, namely that the duration increases if more reviews are in the system, and also if the number of users increases. Since this method almost solely relies on database operations, it can profit from a fast hard disk. The decrease in execution time for the 100%100%20% scenario can be explained by the fact that in this scenario, no meta-trust can be materialized, because all possible combinations are already covered by regular trust statements. The algorithms stop execution when they find that the temporary table is empty after removing entries for which a trust statement already existed. Therefore, the time seen for the 100%100%20% scenario is the time needed to delete the data from the table and realize there is no meta-trust left to materialize.

Given that the execution time is below a minute for the more realistic scenarios, the meta-trust propagation should not take too much processing time in an application.

Trust Computation

The results found in Figures 12.2, 12.3, and 12.4 reflect what we anticipated in the complexity analysis in Section 12.1. Because the computation of *DistrustRank* only takes one iteration, it takes virtually no time compared to the overall duration of the computations. The computation of *TrustRank* also is very quick in comparison to the overall duration, since in the execution we limit the number of iterations to 1000 in case the results have not converged before that (which they normally do). While the database interaction amounts for a big part of the total duration of the computations, for an increasing number of users, the percentage of the total time needed for database

IO is decreasing. We did only measure database IO for the one thread setting, and then used the value for the remaining computations with 2, 3, and 4 threads, since the number of threads does not influence the database IO, and we wanted to concentrate on changes in the computation time, not potential fluctuations in database IO time.

The biggest factor in the trust computation are the matrix multiplications needed to compute the local trust matrix **F**. Therefore, the computation of **F** and **I** take the longest of all the computations, especially when the number of user increases. For a large amount of users, the duration of the **F** and **I** computation is larger than the duration of database IO.

The graphs do not only show how the time needed for computation increases with the number of users, but also that the multi-threading approach can save execution time. Because our CPU had 4 cores, in the 4 thread setting each core could process one thread, leading to a complete utilization of available processing power. The computations should thus be parallelized using a number of threads equal to the number of cores in the CPU.

Another finding from the figures is that our optimization technique of only running the computations for users who are connected to the WOT saves time. This is visible when comparing the execution time of the algorithms with the same number of users, but different rating and trust density. In the case of less dense trust, the size of the matrix is smaller, directly affecting the computation time.

In order to get the execution time for one single ontology–aspect combination, the reported execution time has to be divided by 20. For a realistic setting, the total duration of the computation per ontology–aspect combination is still very fast, i.e., below a second for 100 users and a fully meshed WOT. Even for 500 users, the time taken per ontology–aspect combination is below 6 seconds for a fully meshed WOT. If the number of users connected to the WOT is around 500 users, there is barely any delay in processing updates to the trust data. The changes can directly be presented in the application during runtime. Only meta-trust statements, which affect possibly all ontology–aspect combinations should not be processed directly at runtime.

For that reason, the trust computations should be performed at fixed points in time to ensure the system is working with the most recent data. How often a re-computation is needed depends on the system and the frequency of changes.

Overall Computation

In order to see how the different parameters influence the time for retrieving the overall rating for an ontology, we have based the computation on only the top review (see Figure 12.6), the top 3 reviews (see Figure 12.7), and all reviews (just for worst-case considerations, see Figure 12.8). For each of these settings, we base the computation on both local trust (Algorithm 5 together with Algorithm 3), and global trust (Algorithm 5 together with Algorithm 4). In order to receive an accurate average, we ran each computation 500 times. We measured the execution time for each of the 500 runs, and present the maximum time as well as average and minimum time needed for providing the result. It is intended to make use of the databases caching techniques, since they can also be exploited in real systems, for example by caching results when a user logs in.

The overall computation is one of the most important features of the TS-ORS since it is performed constantly at runtime. So here a quick response time is far more important than for the larger computations which are performed offline and only at dedicated time-points.

The results indicate that while the maximal time in case of a cache miss or for other system-specific reasons can be up to half a second per ontology for local trust based computation with many users and ratings, the more significant average is relatively independent of the amount of users or the number of reviews in the system.

For both local and global trust based computation, the average time needed for computing an overall ontology rating is around 3 milliseconds. Because the costly part of the overall rating computation is the ranking of reviews and ratings within the database, the number of reviews considered for the computation does not influence execution time too much. In all cases, the reviews have to be sorted, and this takes much longer than later using the rating in the computation.

Review Retrieval

When a user wants to browse reviews for an ontology–aspect combinations, it is important to rank the reviews a personalized order, according to the trust information available. So when a user is logged in, the reviews are ranked in an order based on local trust of this user, otherwise they are ordered according to global trust. In our benchmark, we have retrieved a ranking of reviews for all 20 ontology–aspect combinations, both based on local trust and on global trust.

Each ranking is retrieved 500 times, to ensure an accurate average. In Figure 12.9 the results of the benchmark can be found, both for local and global trust.

The ranking of reviews is also a task which is often needed at runtime when a user is browsing different ontology–aspect combinations. For a realistic setting, the ranking can on average be retrieved in around 2 milliseconds. The worst execution time was around 1 second for an ontology–aspect combination, which is still acceptable. The execution time increases with the number of reviews that have to be sorted.

12.2.5 Lessons Learnt from the Benchmark

The execution time can be improved when investing in fast computer hardware. A fast multi-core CPU and sufficient RAM in combination with a fast hard-disk can speed up the execution. Over the years, we have optimized the code several times, to achieve these good results. We believe that the system as such should not produce a bottleneck in a Web-Application. When a multi-core CPU is available, all available CPU cores should be used for TS-ORS computations by enabling the multi-threading functionality within the TS-ORS code.

Depending on how important it is to take the latest user data into account, the frequency of overall re-computation can be increased or decreased. Amazon.com, for example, takes 24 hours to take a trust statement into account. In case a really fast re-computation is needed, the computation can be distributed among different machines, each containing a database filled with only the necessary information. Since the computation of trust is independent for all ontology–aspect combinations, in the most extreme case, one could use one machine per combination and later merge the results.

Whenever possible, a large database cache should be used to cache results and speed up the execution. Our TS-ORS implementation provides a method for caching results, which can be used to initialize the database cache with results. In case a user logs-in, for example, results could be cached for that user.

Chapter 13

Agent Simulation

Running a simulation to evaluate a system is a common form of evaluation. Using synthetic data to test the system behavior has several advantages over purely analyzing real world data. While it is important to see whether the algorithms function with real world data as well, it is difficult to check the system behavior in edge cases without generating the data.

Furthermore, there is no control over user behavior in real world test data. That means, it is not possible to check how an increase or decrease in user errors affect the results of the algorithms. Therefore we decided to run a simulation for our TS-ORS analyzing the behavior of the algorithms and comparing our TS-ORS against Guha's ORS.

The main purpose of the simulation is to check the system behavior in a controlled environment and to draw conclusions for its use in real world systems. In order to achieve this, we have run the same task (computing overall ratings for objects based on the trust and rating information in the system) several times, altering parameters in order to see how different coverage or errors made by the agents affect the outcome. In the remainder of this chapter, our simulation setup is presented in Section 13.1, while the results and the interpretation of the results can be found in Section 13.2. The chapter ends with a conclusion in Section 13.3. Parts of this chapter are based on (Lewen, 2009d; Angeletou *et al.*, 2010).

13.1 Setup

For the execution of the simulation we used the TS-ORS code also deployed within Cupboard, and used for the benchmark in Chapter 12.

13.1.1 Goal of the Simulation

One of our main motivations for developing the TS-ORS were overcoming the limitations in the way trust can be assigned in the ORS. For that, in the simulation we compared the TS-ORS against the ORS in exactly the situation we encountered:

Reviewers write on the one hand helpful (good), but on the other hand also not helpful (bad) reviews. While in the ORS trust can only be assigned globally (equal to W_{OX}-meta-trust statements in our model), the TS-ORS allows fine-grained trust assignment (W and the meta-trust statements discussed in Table 3.1). Furthermore, we want to analyze the behavior of the algorithms in general, namely their ability to provide user-specific rankings, and their robustness against spammers.

13.1.2 Approach

In order to see if users with a taste different from the mainstream could receive personalized ratings representing their taste, and to see how malicious users can influence the ranking results, we distinguish three types of agents in our simulation.

Different Types of Agents

We created 60 regular (good) agents, 20 special agents, and 20 spammers. The taste of regular agents represents the mainstream, while the taste of special agents differs from the mainstream. We chose to have more than one agent in each group, to even out randomness and to get a more accurate result. While the distribution of agents in the system influences some of the results (we will mention which results later in the analysis), we felt that in a normal TS-ORS, most users should qualify as regular users, while some have special taste, and some might try to promote spam.

Different Types of Objects

We defined rules to represent agent taste, based on which the agents each rated 25 good objects, 10 special objects (these objects polarize agents in contrast to the good objects), and 20 bad objects (that spammers want to promote). Agents only trust other agents from their group. This means, e.g., that special agents only trust special agents.

Creating the Gold Standard by Defining Rules for Rating Objects

Regular (good) agents rate good objects 5 stars, bad objects 1 star, and special objects half of the time 2 stars, the other half 4 stars. Special agents rate good objects 4 stars, special objects 5 stars, and bad objects 1 star. Spammers rate bad objects 5 stars and the rest 1 star. These rules are applied for all 5 aspects of each object and act as a gold standard against which simulation results are compared.

Scenarios to Compare TS-ORS Against ORS

We defined 3 scenarios which allow a comparison between TS-ORS and ORS:

- In scenario 1, agents rate all aspects of an object according to rules we defined, while trust can only be assigned globally.
- Scenario 2 assumes that the agents only have expertise to review one of the five aspects of an object correctly, but still review all of them. In case they write a review for an aspect for which they have no expertise, we mimic a bad review by inverting the rating defined by the rule. For example, a special agent making a bad review would rate a special object 1 star instead of 5 stars (this does not apply to the spammers, who will automatically rate bad objects 5 stars and the rest 1 star). Trust in this scenario can also only be assigned globally.
- In scenario 3, we use the same rating data as in scenario 2, but allow topic-specific trust statements. Agents in scenario 3 trust the reviews for aspects for which the agents have expertise, and distrust the reviews for which the agents don't have expertise.

Scenario 1 simulates the behavior of the ORS in the case all agents write either only good or bad reviews. Scenario 2 simulates the behavior encountered in the ORS when some reviews of a reviewer are good and others are bad, but trust can only be assigned globally. Scenario 3 simulates the behavior of the TS-ORS given the same data as used in scenario 2, but topic-specific trust statements.

Simulating Data Sparsity and Agent Mistakes

To include an analysis of the effects of data sparsity and agent mistakes, for each of the three scenarios we ran the simulations with varying data densities (100%, 50%, 10%, and 5%) and percentage of agent errors (0%, 10%, and 20%), leading to a total of 36 simulation settings.

Data Sparsity: The varying data density simulates the problem of data sparsity, which is encountered in many real-word systems which rely on data to provide recommendations or computations (Herlocker *et al.*, 2004). 100% data density means that each agent rates all object-aspect combinations in the system, and states trust to all other agents. 10% data density means that each agent randomly selects 10% of the objects in the system, and then rates all of the aspects for each of the selected objects. Trust is only stated to 10% of the other agents, which again are chosen randomly.

User Mistakes: In the real world, users make mistakes and do not always act rationally. We simulated user errors as well in the system, where 0% means that users make no mistakes, and 10% means each user makes an error in 10% of the cases when giving a trust statement. For example, a special agent will trust another special agent, unless he makes an error, in which case he will distrust the other special agent. In the scenarios with errors, all three agent types make mistakes.

Result Computation

For each of the 36 simulation settings, we computed the trust ranks for the agents, and an overall rating for all agents on all objects, both based on global trust and on local trust. We then averaged the computed overall ratings over all objects in a specific group for all agents in a specific group. The result was compared to the gold standard. For example, if the computed average for all 25 good objects for all 60 regular agents based on local trust equals to 5 stars, that means that each agent received the correct 5 stars rating according to our rules for the 25 good objects. The computations were performed using the default parameters introduced in Section 5.3. In addition to the average ratings for each group based on local and global trust, we also computed the average over all ratings for all objects within a group to compare to.

In order to interpret the results, it is important to remember that global trust is influenced by the largest agent group in the system. Global trust can be seen as a majority vote, which is not personalized. Local trust can only be used in case an agent is connected to the WOT.

13.2 Results

In this section we will present and analyze the outcomes of the 36 simulation runs. The simulation results can be reproduced by running our code.[1]

13.2.1 Scenario 1

As mentioned before, each scenario consists of 12 runs total, 4 different data density settings, and 3 different error settings. In the figures one can find the results for the different data density settings side-by-side in one diagram. The different error settings can be found in separate figures.

0% Error

The 0% error setting of scenario 1 is the ideal scenario to be encountered in an ORS. No agent makes mistakes, and all agents only provide either good or bad reviews. If we look at the results shown in Figure 13.1, it can be seen that given perfect coverage (all objects are reviewed, and all agents state their trust towards each other), the algorithms indeed can replicate the gold standard we defined in Section 13.1.2 for the

[1]The files needed can be found online at: `http://people.aifb.kit.edu/hle/thesis/simulation/`, last checked on 24.11.2010.

results which are based on local trust. As expected, the average rating for regular (good) agents for good objects is 5 stars, the same holds true for special agents and special objects. Furthermore, the average rating for bad objects is 1 star, both for good and special agents. This shows, that the local trust-based results are not influenced by the rating of agents to which no trust connection exists in the WOT. The global results, which are not personalized, are influenced by the largest group, in this case the regular (good) agents. As the data density goes down (objects have less reviews, and less agents express trust towards each other), the results change towards majority taste. The is due to the size of the test data we have used and will be explained further in Section 13.2.4. Basically the problem is that not all agents are connected to the WOT, and when they are not connected, the global trust metric has to be consulted for trust information. In the scenarios with high data sparsity, the rating results based only on the top review is closer to the gold standard than the results based on the top 3 reviews. This can be explained by the lack of trusted reviews. If there are not 3 locally trusted reviews, globally trusted reviews also have to be factored into the rating computation, thus affecting the computation result. As a general observation, global trust based results are closer to our gold standard than the simple average, and local trust based results are closer to the gold standard than global trust based results. In general, the results based on the top rating are closer to the gold standard than the ratings based on the top 3 ratings.

10% Error

The results for our 10% error setting can be found in Figure 13.2. One main observation is that when agents trust agents they should not be trusting, or distrust agents they should be trusting (i.e., when they make an error), the largest agent group dominates the results. When trust or distrust is placed mistakenly, a connection between two groups is created, which then is used during the trust propagation. This leads to the largest group with inter-group links accumulating more trust, and being ranked higher in the reviewer ranking. The regular agent group is the biggest in our experiment, therefore dominating the results of the special agents and spammers.

Unlike in the setting with no errors, the special agent and spammer group cannot retrieve a review from their peers as top review, since once one agent from these group trusts an agent from the regular agent group, this agent will receive enough trust to become the predominant reviewer. The gold standard can only be met for the cases where it coincides with the gold standard for the regular agent group. The difference in results for the different coverage levels can be explained with a lack of connectedness to the WOT.

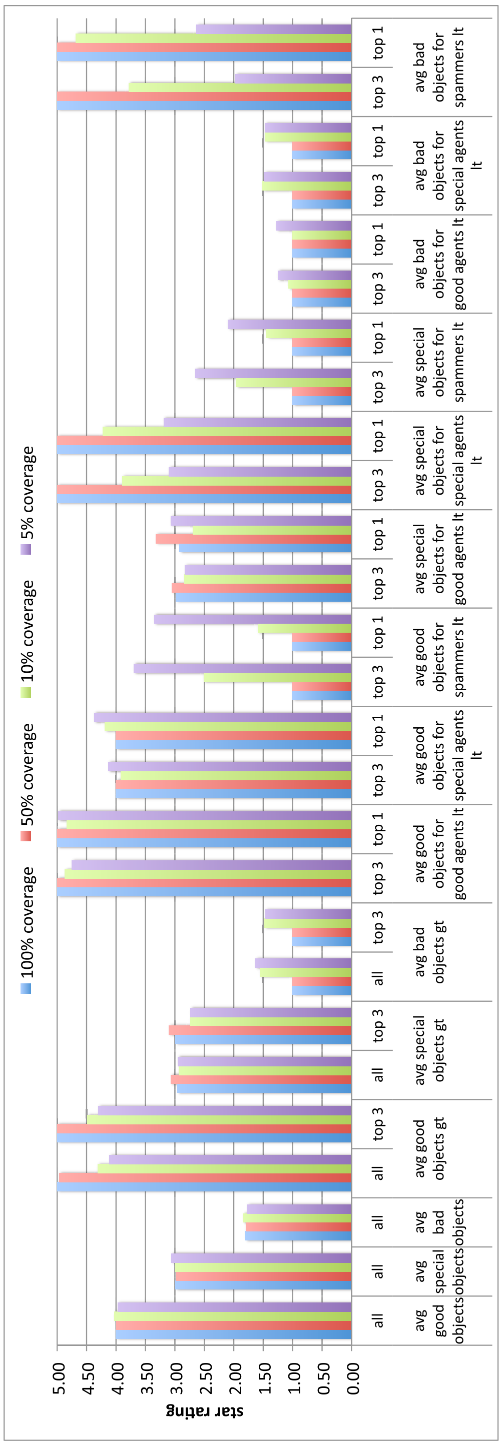


Figure 13.1: First Scenario with 0% Error, results based on global trust are marked gt, results based on local trust lt.

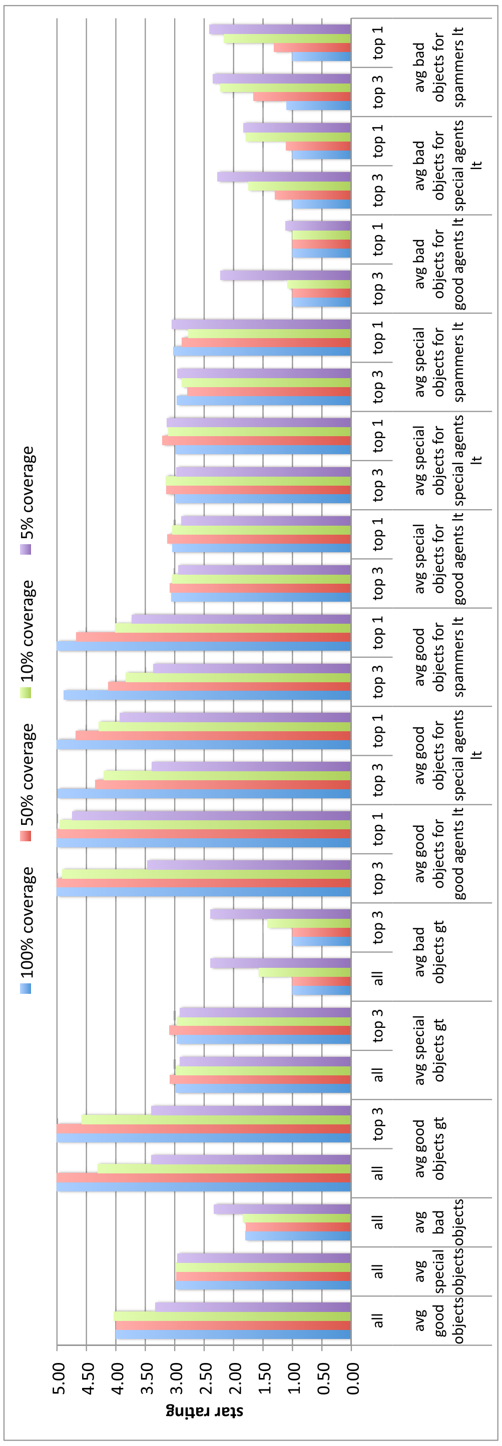


Figure 13.2: First Scenario with 10% Error, results based on global trust are marked gt, results based on local trust lt.

20% Error

In a setting with 20% error (see Figure 13.3), the results resemble very much the results of the 10% setting. The effects described above are just more visible. There is not much difference between making errors in 10% or 20% of the cases.

13.2.2 Scenario 2

Scenario 2 represents the case where agents write both bad and good reviews, but other agents can only choose to trust or distrust them globally (equivalent to W_{XO} statements in the TS-ORS). In our setting we decided that only one of the 5 aspects could be reviewed correctly, leading to 80% of the reviews being bad reviews. Choosing a different value here (for example deciding that 3 out of 5 aspects can be reviewed correctly) affects the outcome, but what was important for us was to determine whether there is a difference in ranking results, and for that reason we chose this setting. In order to at least get good results for the aspects that agents can provide good reviews for, agents have to still trust members of their group, even though most of their reviews are bad. The alternative of not trusting other agents at all (since 80% of the reviews are bad) is no real alternative, since otherwise the overall rating would have to completely rely on global trust, which would then favor spammers, who would trust each other by default.

0% Error

In this setting, as expected, results significantly differ from our gold standard (see Figure 13.4). The good objects receive the lowest overall ratings and the bad objects the highest. This is the result of agents having to trust bad reviews. Similar to scenario 1, also in scenario 2, it can be seen that the overall rating results move towards the results which are based on global trust when data density decreases and less agents are connected to the WOT.

10% and 20% Error

Here the same observations can be made as in the 10% and 20% error cases of scenario 1, i.e., the biggest agent group affects the ratings of the other groups. The results can be found in Figures 13.5 and 13.6.

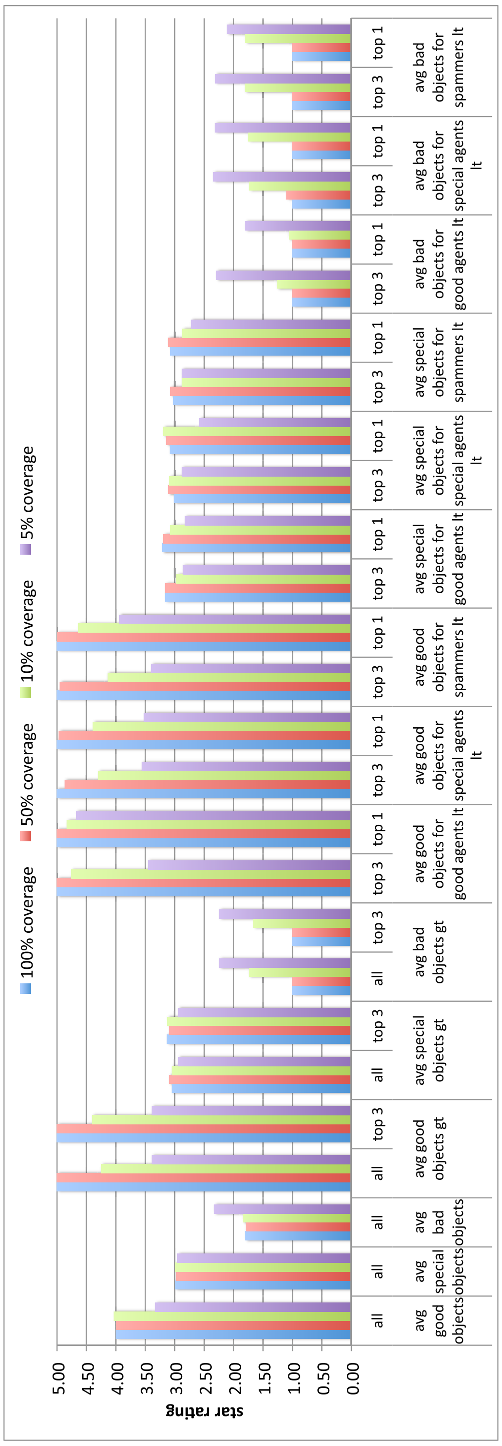


Figure 13.3: First Scenario with 20% Error, results based on global trust are marked gt, results based on local trust lt.

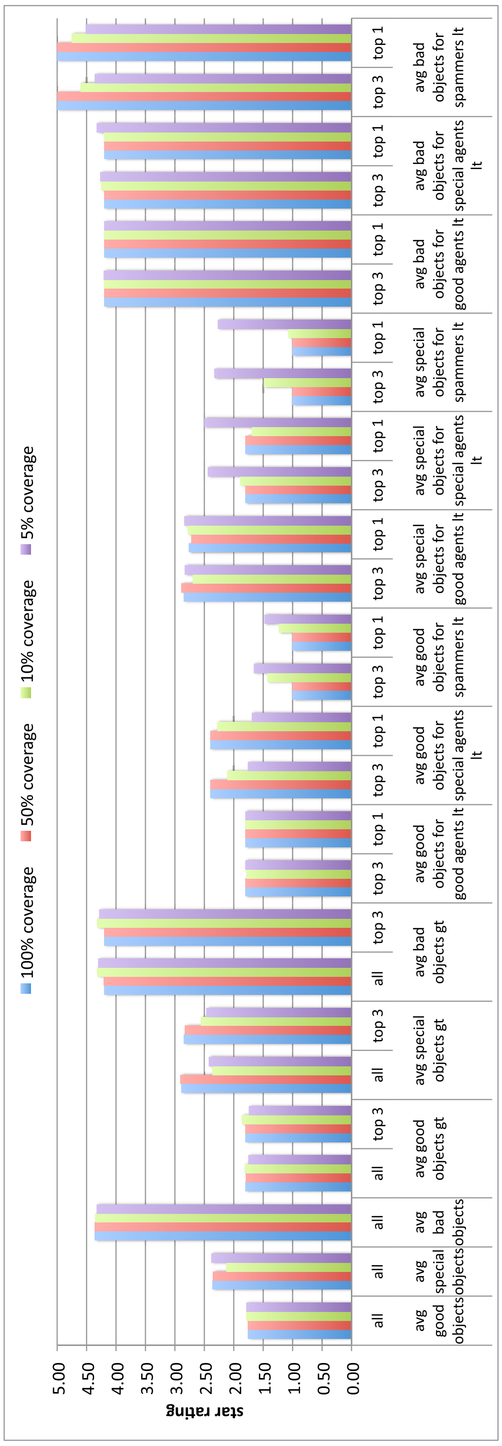


Figure 13.4: Second Scenario with 0% Error, results based on global trust are marked gt, results based on local trust lt.

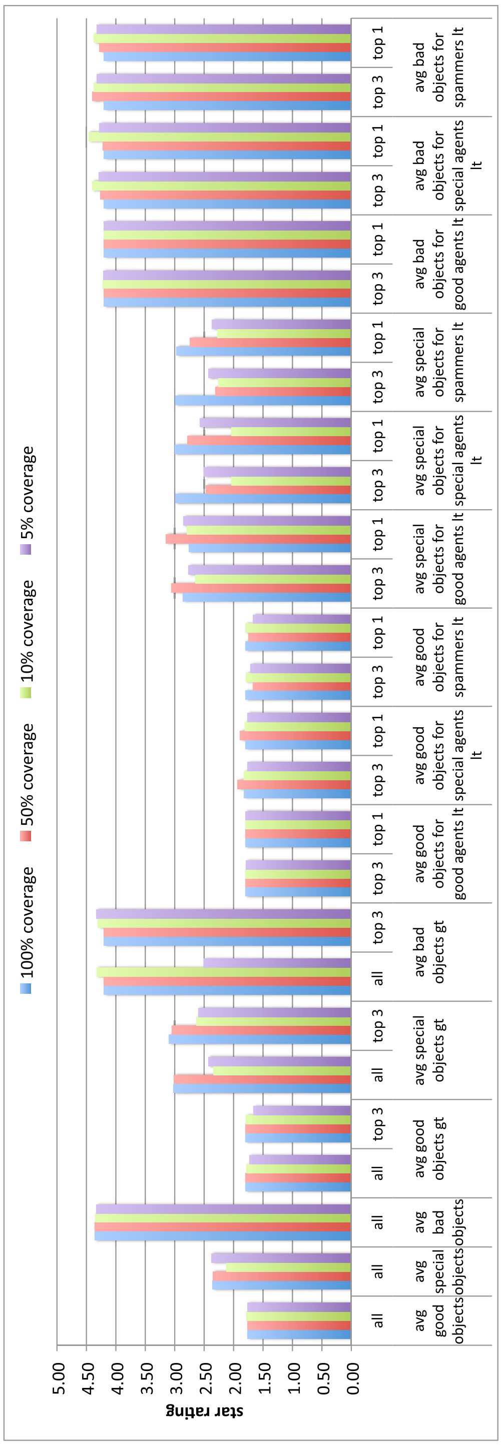


Figure 13.5: Second Scenario with 10% Error, results based on global trust are marked gt, results based on local trust lt.

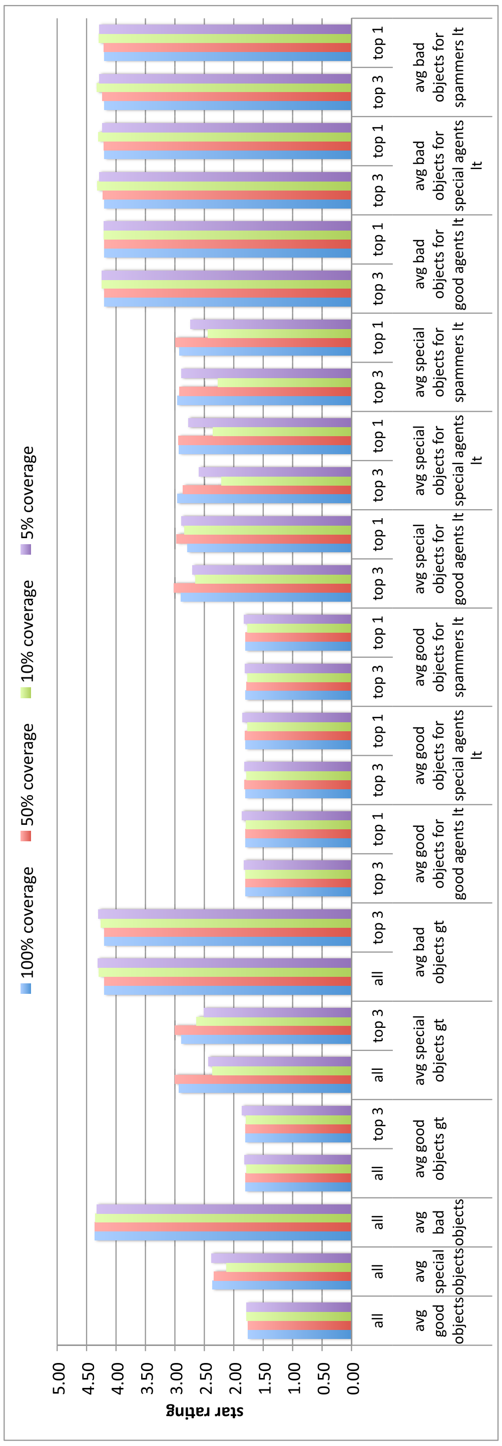


Figure 13.6: Second Scenario with 20% Error, results based on global trust are marked gt, results based on local trust lt.

13.2.3 Scenario 3

Scenario 3 reuses the review and rating data from scenario 2, but in this scenario, we allow topic-specific trust statements. In the setup, reviewers are only trusted for the aspects for which they can provide good reviews, and distrusted for the rest.

0% Error

As Figure 13.7 shows, using topic-specific trust, the result matches our gold standard. With the topic-specific trust statement, it is possible to only trust good reviews. The total number of bad reviews has not changed, as can be seen by looking at the basic average of ratings for the objects, which are the same as for scenario 2. It is only that now agents have the possibility to issue more specific trust statements and do not have to decide whether to trust another agent globally or not. So even though there are 80% bad reviews in the system, the algorithms still produce the desired outcome (namely match our gold standard), based on accurate trust assignment and sufficient connectivity. With decreasing coverage we can once again see how the increasing lack of connectivity to the WOT blends local trust ratings and global trust ratings.

10% and 20% Error

Also in scenario 3 we can find the described effect of the predominant group influencing the rating results of the other subgroups (see Figures 13.8 and 13.9). Still it is noteworthy that even with 20% of all trust statements being wrong, the average rating is outperformed. In other words, even if not all trust information is correct, it is better to employ trust-based algorithms for ranking than relying on basic arithmetic measures like average.

13.2.4 Comparison of a 5% Coverage Scenario once with 100 Agents and once with 1000 Agents (0% Error)

As mentioned before, the difference from the gold standard at smaller coverage levels (with few reviews and inter-agent trust statements) is due to the missing connectedness of agents to the WOT. After all, a 5% coverage (data density) equals to only two reviews and 5 trust statements per agent in the 100 agent setting. We have investigated the effect of increasing the number of agents in the simulation while keeping the number of objects and level of data density constant. The results can be found in Figure 13.10.

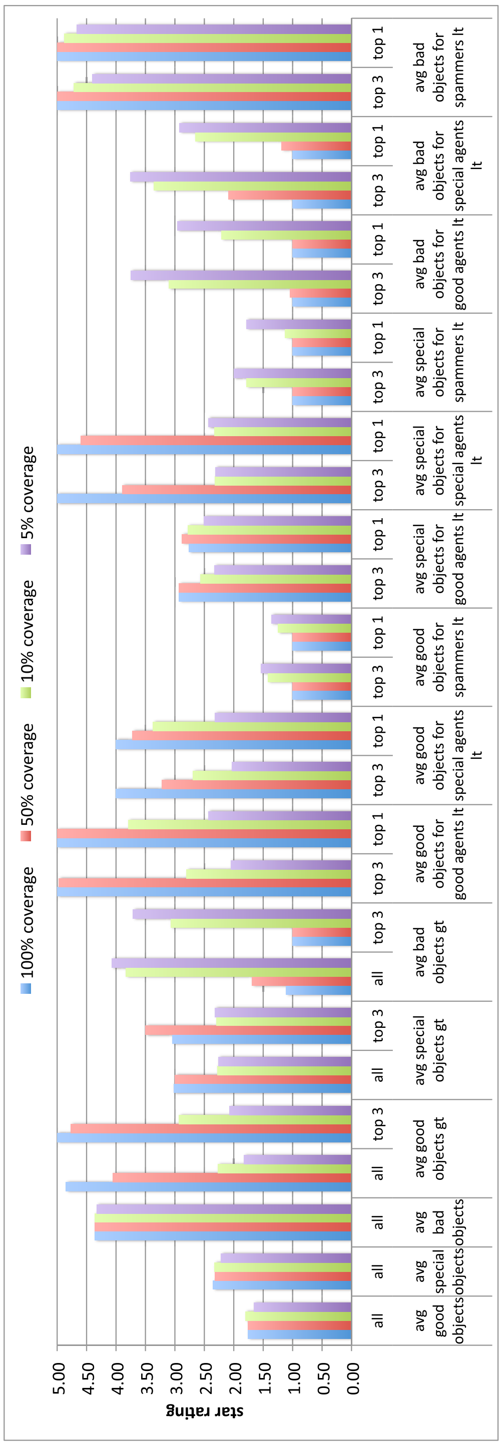


Figure 13.7: Third Scenario with 0% Error, results based on global trust are marked gt, results based on local trust lt.

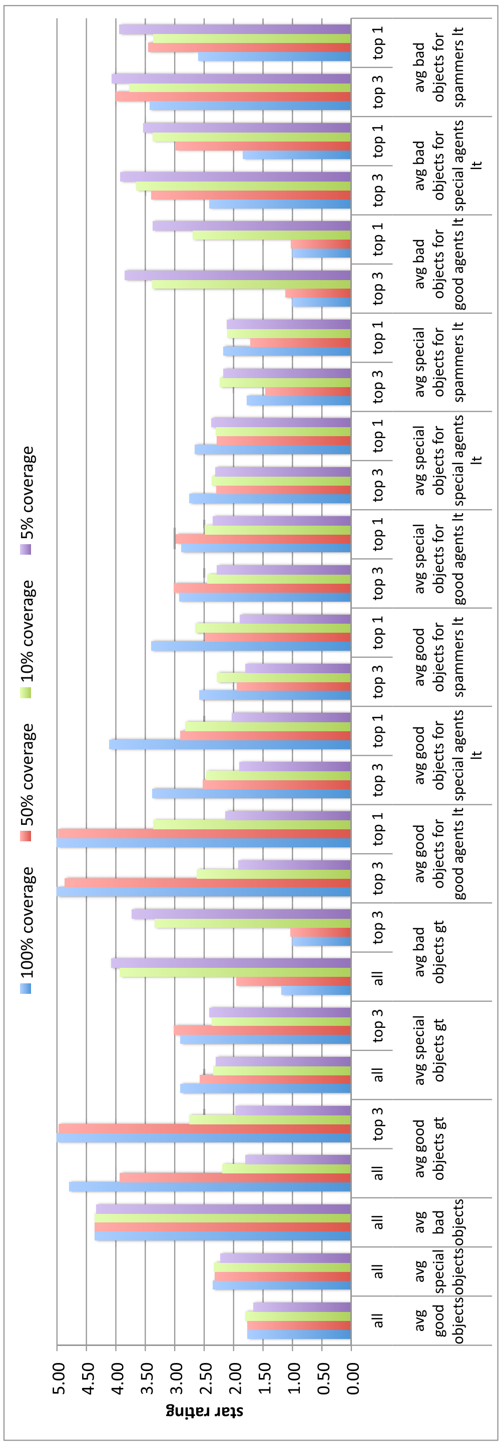


Figure 13.8: Third Scenario with 10% Error, results based on global trust are marked gt, results based on local trust lt.

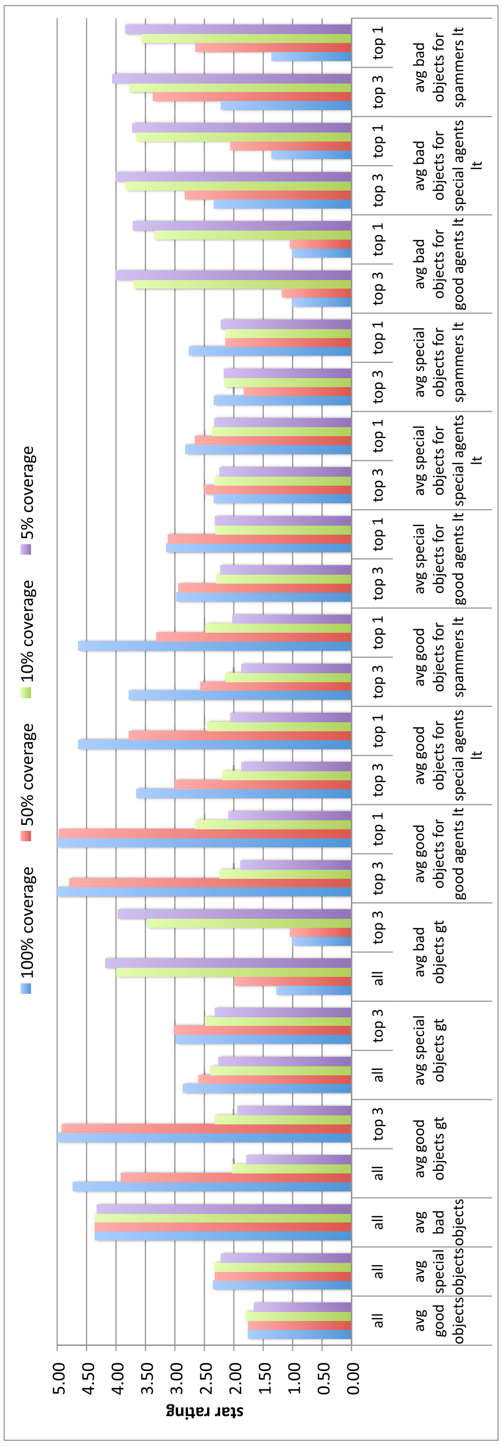


Figure 13.9: Third Scenario with 20% Error, results based on global trust are marked gt, results based on local trust lt.

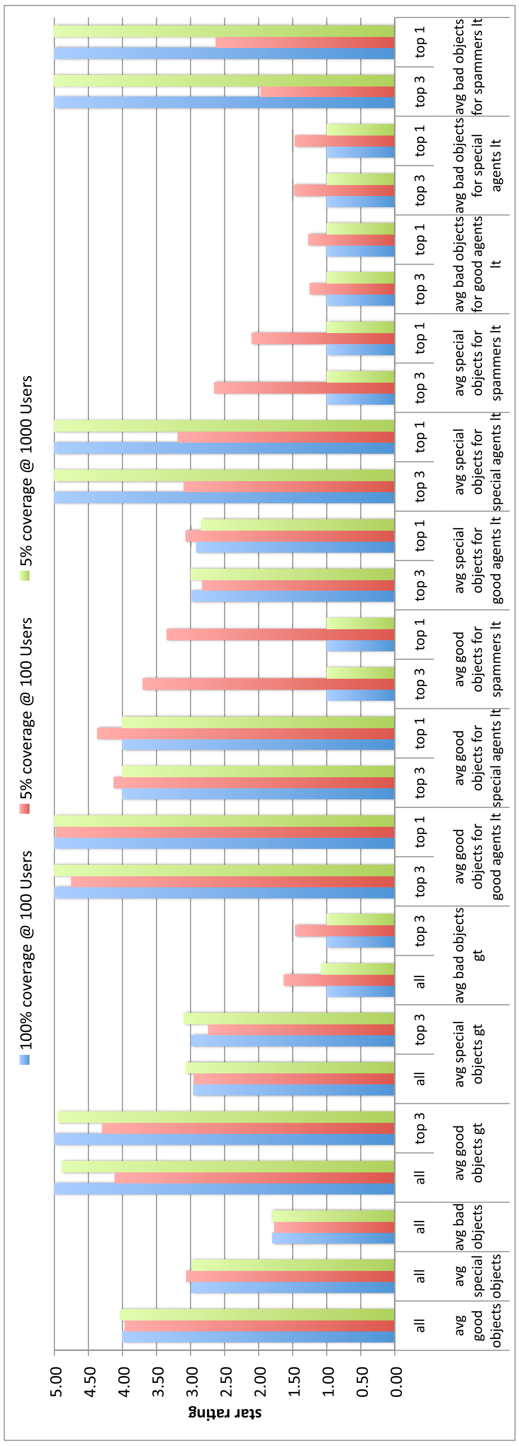


Figure 13.10: Comparison of Results For Scenario 1 With 0% Error

We furthermore analyzed the setting with 1000 agents in more detail, to find out how the derivations from the expected outcome can occur. As discussed in Section 13.2.5, the theoretical lower boundary for a setting that matches the gold standard for our setup lies with 3 ratings per object (one for each group), and one trusted peer agent for each object per agent. That amounts to 3% coverage in reviews and 1% coverage in trust per object (in a 100 agent setting). Since we have employed randomization for the simulation, the reviews and trust were not assigned optimally, but randomly based on the rules explained above. This could lead to settings where agents are not connected to the WOT, and therefore for them, no local trust information can be used to compute the overall rating. In order to see how many agents are disconnected for each object (since for all these agents, in our experiment setup their trust results would be based on global trust), we ran an analysis. For the setup 100 agents, 5% coverage and 0% error, the results can be found in Figure 13.11. The x-axis represents 10% buckets, where 0-10% means that only 0-10% of the agents of a group are not connected to the WOT for an object. The y-axis shows the number of objects in each bucket. As is evident from Figure 13.11, for several objects a large percentage of agents was not connected to the WOT, in some cases even more than 90%. Therefore it is not surprising that the average ratings based on local trust deviate towards the ratings based on global trust, since when agents are not connected to the WOT, the ranking algorithm has to rely on global trust. In the rest of the cases, the correct review is retrieved based on local trust and is blended into the results. This is why the results always deviate in direction of the result based on global trust. Given a setting of 1000 agents, 5% coverage and 0% error, there are almost no agents disconnected from the WOT, as can be seen in Figure 13.12. If we remember that the minimal setting for achieving the expected results is only partially dependent on the agent size (the number of reviews required is independent from the number of agents, but is based on the number of objects), we are not surprised to see improved results in the setting with more agents (since the 5% here equal to 10 times more reviews and trust statements than in the 100 agent setting). As a conclusion of the comparison we note that the number of reviews and trust statement in total is not as important as having the right ones (the ones described in Section 13.2.5).

13.2.5 Minimal Setting Matching the Gold Standard

As discussed above, most of the problems with results deviating from the gold standard are due to agents that are not connected to the WOT, which causes the system to use global trust. So the more agents are not connected to the WOT, the more the local trust based results resemble global trust based results. A minimum setting matching the gold standard would be having at least one agent from each of the three groups

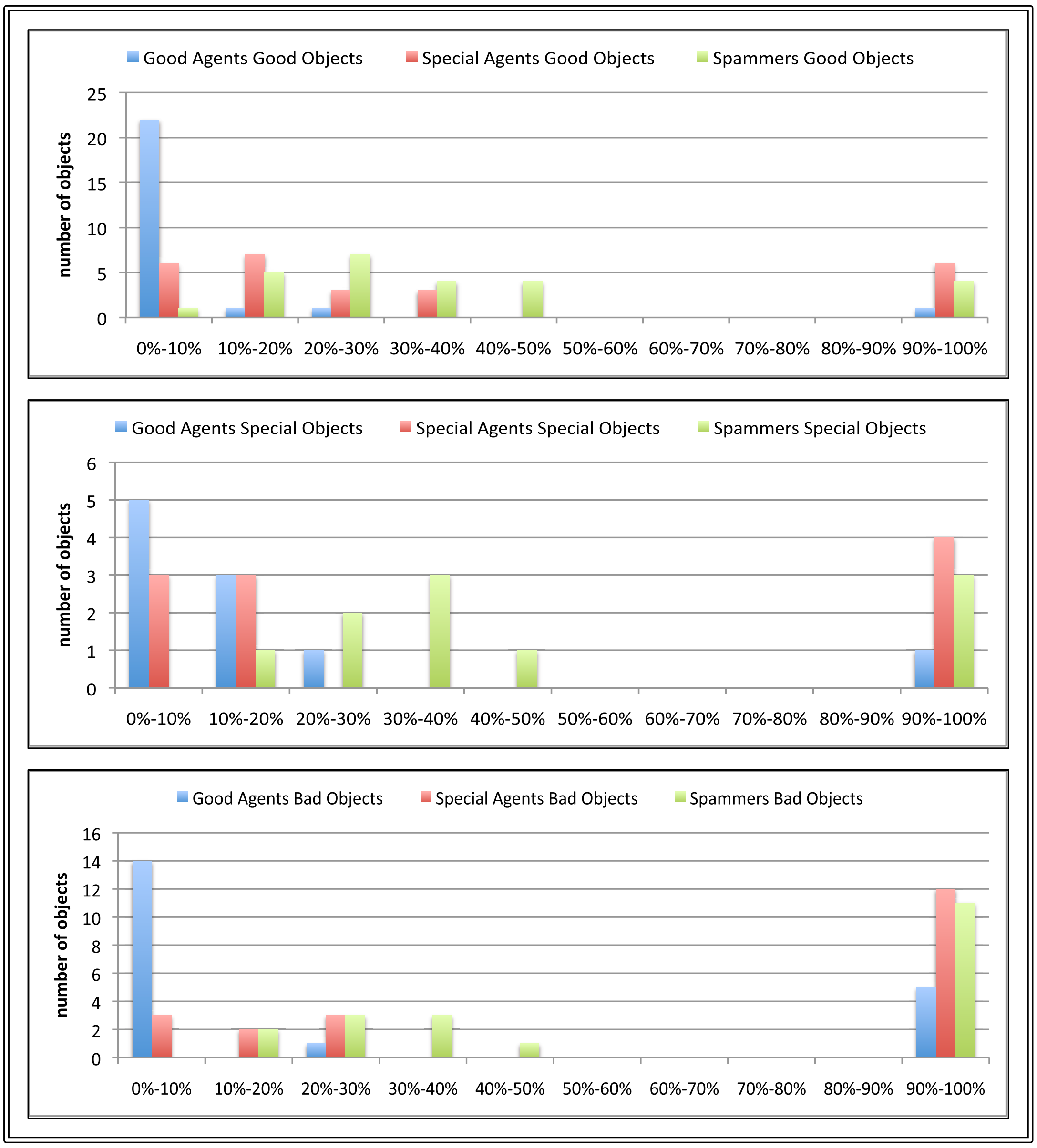


Figure 13.11: For this graph, we have checked for each object in the investigated group, which percentage of the agents in the group investigated has no local trust information for this object. The findings were sorted into 10% buckets. The computations were performed with 100 agents, 5% coverage and 0% error.

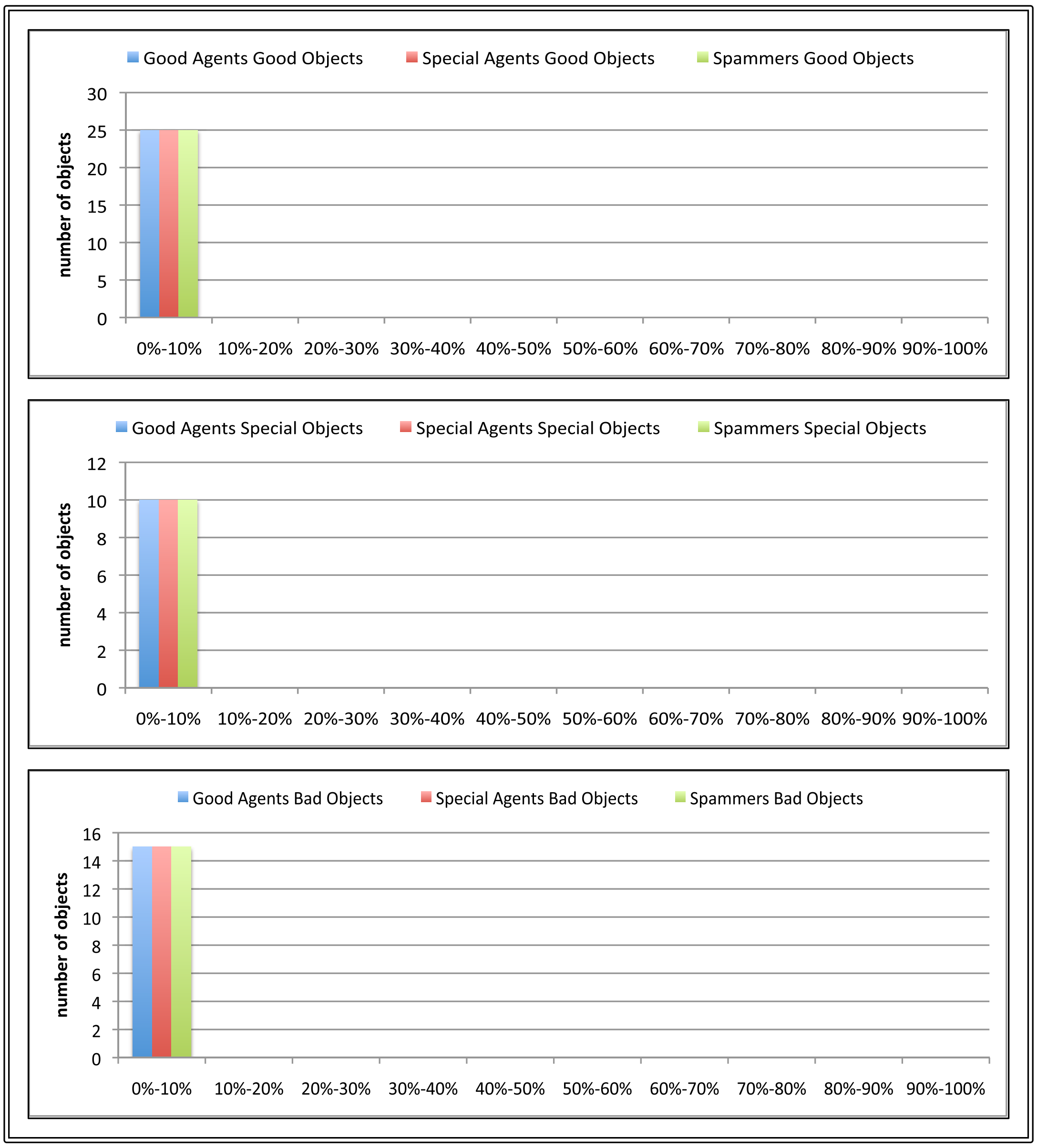


Figure 13.12: For this graph, we have checked for each object in the investigated group, which percentage of the agents in the group investigated has no local trust information for this object. The findings were sorted into 10% buckets. The computations were performed with 1000 agents, 5% coverage and 0% error.

reviewing all objects, and then having the rest of the agents in each group trusting this agent completely, and distrusting the 2 other reviewing agents. This setup results in all agents being connected to the WOT, and the having their reviewer provide the top-ranked rating for the objects. Of course, if only one good review exists for each group, the results have to rely on the top review, and not the top three reviews.

13.3 Conclusion

In the concluding section we want to revise what we have learned from the simulations, and how this can help to improve the accuracy of ranking in real world systems like Cupboard.

13.3.1 Validity of Simulation Results for Real World Systems

We think that the simulation we ran on the one hand shows that the algorithms do work in a perfect scenario, but also show how derivations from the ideal scenario can influence the results. We purposely chose to employ randomness when generating data for the less dense scenarios and those that contained errors, because randomness is the worst case scenario for the TS-ORS. In real life, it is more likely that users will not trust or distrust random users, but users whose reviews they read. This would lead to a scenario with more specific trust data and will thus be closer to what we described as the minimal setting with expected behavior. We created the simulation this way to ensure that deviations in the results are visible on the graph. Another way to analyze the algorithms would have been to just check for each agent whether the top ranked review came from an agent from the same group. In the end, a perfect personalized rating can only be computed if the right agents are trusted beforehand. The outcome of the TS-ORS algorithms can only be as good as the quality of the data they operate on. Since in (Guha *et al.*, 2004) the authors already performed an analysis of the trust propagation algorithms on a large real-word data set (namely the Epinions data set), we focused in the thesis on this simulation to better understand the algorithms behavior in certain scenarios.

13.3.2 Attacks on the System

The simulation results show that spammers cannot affect the overall ratings which are based on local trust, if no agent outside the spammer group trusts a spammer. In case the spammers are only a small group in relation to the overall group, they cannot even influence the ranking result too much in case they are accidentally trusted. Only when spammers are the largest group in a system, and they are trusted by a non-spammer, can they influence the local trust based results of other agents.

Of course, global trust based ratings and simple average ratings can be affected (compare to linkfarms (Wu and Davison, 2005) on the web trying to increase their Google Page-Rank), but these are only used for users who cannot be identified or are not connected to the WOT. Now one can argue that the spammers could try to gain the trust of regular users by acting like a regular user in, e.g., 90% of the cases, and then rate objects maliciously in the rest of the cases. Even this attack is not easy to pull off, since we compute the WOT separately for each of the object–aspect combinations. That means that the spammer, under normal circumstances, would only be trusted in the cases where he is acting like a regular user, and not in the other cases. The only way for the spammer to draw an advantage of such behavior would be by convincing a user to issue a global meta-trust statement, also covering potentially malicious reviews. Even if the spammer manages to convince a regular user A to issue a meta-trust statement, this would primarily affect the A's ratings and the ratings of other users that trust A. Only in case all other users have to trust each other globally can the scores of many other users be affected easily. In order to prevent being a victim of a spammer, users should be very cautious to use W_{OX} meta-trust statements.

13.3.3 Condensed Simulation Results

We have condensed the most important findings of our simulation in one graph, which can be found in Figure 13.13. The figure displays the averaged overall ratings for each agent-group and object combination for each of the three simulation scenarios given perfect data coverage. This figure comprises the main findings of the simulation, namely that the gold standard cannot be matched in case only global trust can be expressed and reviewers write both good and bad reviews. Using topic-specific trust data, on the other hand, even with a smaller number of good reviews, the gold standard can be matched perfectly. We therefore argue that the simulation was successful in showing in which scenarios the TS-ORS provides more accurate rankings and overall ratings than the ORS.

13.3.4 Lessons Learned

The main lessons learned from the simulation are that it is very important for users to express their trust towards other users in an ORS or TS-ORS. Only when a user is connected to the WOT, can the rankings and overall ratings provided by the system be personalized. Otherwise, a user's experience will be the same as for users who are not identifiable or not connected to the WOT because of missing trust statements. Therefore there is an incentive for users to register with the TS-ORS, log-in before using it and state their trust and distrust towards other users. Also users should use meta-trust statements carefully when they do not know another user.

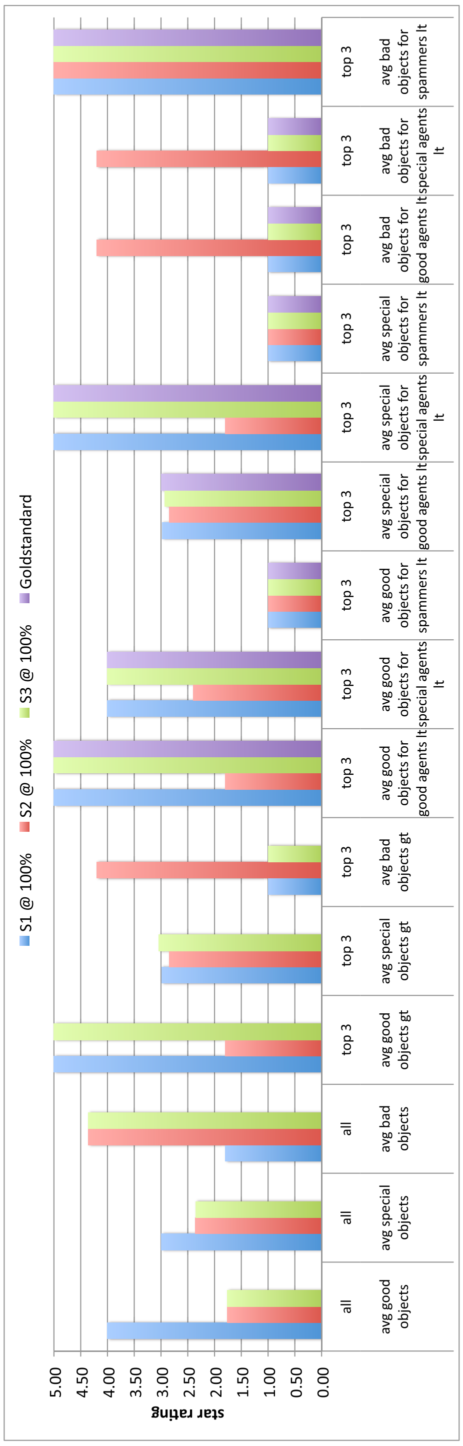


Figure 13.13: The Results of the Three Simulation Scenarios for 100% Data Coverage and 0% Error Side-by-side the Gold Standard.

Because reviewers can have different areas of expertise, in which they can provide qualified reviews, our trust model extension in the TS-ORS allows to trust and distrust selectively. Even if spammers in general cannot easily affect the local trust based results of other users, it is advisable to ban them from the system as soon as they are identified, so that the rest of the users have a better usage experience and global trust based results cannot be affected. For that purpose, a button can be introduced which users can use to inform the system operators of potential spammers. After confirming that a user is indeed a spammer, the user can then be removed from the system. Some systems also require administrators to manually clear new user registrations to increase the barrier for spammers to join the system.

Chapter 14

User Study

Even though good performance and theoretical ranking accuracy is important for real-word system, the most important gauge is user acceptance. We therefore decided to perform a user experiment in which we compared our Cupboard Plugin for the NeOn Toolkit (see Section 10.2) against both the Watson Plugin for the NeOn Toolkit (see Section 10.1) and systems found on the World Wide Web for the task of ontology reuse.

In particular, we wanted to examine both how the integration of a tool supporting ontology reuse with the ontology engineering environment on the one hand, and quality information on the ontologies on the other hand can facilitate the reuse of ontological resources.

For the experiment we found 20 researchers from the Semantic Web community willing to participate. The participants came from 6 different academic institutions. While some of the participants worked on the NeOn project[1], the majority of the participants were not members of the NeOn project.

We partitioned the users into three different groups, each with different tools at their disposal. All participants were using the NeOn Toolkit[2] as ontology engineering environment, and had an Internet browser at their disposal. The task all participants should complete was extending an ontology solely containing the concept "Fish" to an ontology representing the fish domain, by only reusing ontological content found on the Internet or in special ontology search engines or repositories.

It was left up to the participants to decide for which concrete use case the ontology was modeled (i.e., the participant could decide on which aspects or parts of the fish domain to focus, and how to model them within the ontology).

One group (group 2) had the Watson Plugin at their disposal, allowing them to query Watson and integrate results directly from within the NeOn Toolkit. Group 3 had the Cupboard Plugin at their disposal, which allowed them to query Cupboard, receive a result ranking based on overall ratings for the ontologies as provided by the TS-ORS, and integrate the results from within the NeOn Toolkit.

[1] `http://www.neon-project.org/`, last checked on 24.11.2010
[2] `http://www.neon-toolkit.org/`, last checked on 24.11.2010

Group 1 was the control group. Participants did not have a tool to support ontology reuse integrated in the ontology engineering environment. Furthermore, they had to create all ontology statements they wanted to reuse by hand in the NeOn Toolkit, and could only turn to the Internet browser to find results.

After the experiment was completed, the participants filled out a questionnaire.

In the following, we will first explain the experiment setup in Section 14.1. We then describe the execution of the experiment in Section 14.2 and the results of the experiment in Section 14.3. The results are analyzed in Sections 14.4 and 14.5. The chapter ends with a conclusion in Section 14.6. Parts of this chapter are based on (Lewen, 2009a; García-Castro *et al.*, 2008; Dzbor *et al.*, 2009; Dzbor *et al.*, 2010).

14.1 Experiment Setup

In this section, we describe the setup and execution of the user experiment.

14.1.1 Goal of the Experiment

As mentioned before, the goal of the experiment was to evaluate empirically how ontology reuse can be facilitated by both tool integration and quality information on the ontologies. Facilitating ontology reuse means both allowing the user to reuse content more easily (offering help during the different steps of the reuse process), and also to produce an ontology with reused knowledge in a shorter time-frame. Since it is difficult to measure objectively if ontology reuse has been facilitated for a user, we asked the users directly in the experiment questionnaire (to get their subjective opinions).

The motivation to limit the time participants had available for performing the task to 20 minutes was twofold. On the hand hand, it is important to see how much time can be saved during the reuse process, and on the other hand, people were more likely to participate in the experiment if the duration was known beforehand and acceptable. In theory, given enough time, each user could evaluate each ontology by hand and only import the best parts by hand. Since this would take too long, we wanted to see what an ontology engineer could achieve in 20 minutes, given the tool support they had in the different groups.

14.1.2 Technology Used in the Experiment

The four technologies used in the experiment are the NeOn Toolkit, an Internet browser, the Watson plugin for the NeOn Toolkit, and the Cupboard plugin for the NeOn Toolkit.

NeOn Toolkit

The NeOn Toolkit[3] (Haase *et al.*, 2008) is an ontology engineering environment which is based on Eclipse and allows an easy integration of plugins through the Eclipse Plugin Framework.

Internet Browser

The users were given access to the Internet and provided a list of semantic web search engines[4] as a starting point (the page was set as the homepage of the browser).

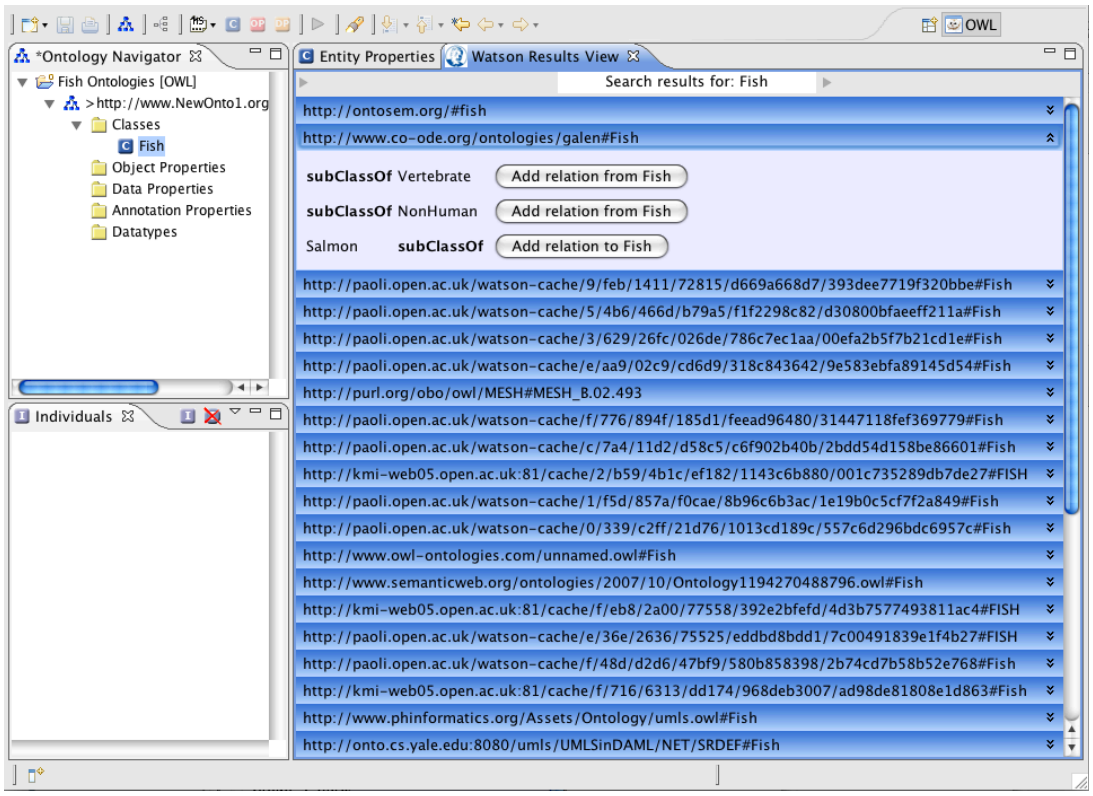
Watson Plugin

The Watson plugin for the NeOn Toolkit (see also Section 10.1) allows the users to directly access the Watson Semantic Web gateway from within the NeOn Toolkit. Results presented within the plugin are ranked based on Lucene. That means that users have to browse the result list to find suitable ontologies to reuse, because the ranking does not reflect the quality of the ontologies. A screenshot of the result view within the Watson plugin can be found in Figure 14.1.

Cupboard Plugin

The Cupboard plugin for the NeOn Toolkit (also see Section 10.2) is an extension of the Watson plugin, but connects to a specified ontology space within Cupboard.[5] We have loaded the ontology space with ontologies we found on the web and in Watson, as well as with some other ontologies we created. The ontologies were then reviewed by members of the NeOn Project. A special feature of the Cupboard plugin is that it can retrieve the overall ratings for each of the ontologies in the result-set, and rank the results accordingly (see Figure 10.2). During the course of the experiment, we relied on global trust ratings, that means the identity of the user was unknown to the system, and thus all users were presented with the same ranking order. Another feature we added to the Cupboard plugin was the ability to add multiple super-classes (all super-classes from found concept to the root) and sub-classes (all classes below the concept) with one-click. This functionality is currently not available in the Watson plugin, since without knowing anything about the quality of the ontology, we felt that it would be too risky to allow adding too many statements blindly. Since the user

[3]`http://www.neon-toolkit.org/`, last checked on 24.11.2010
[4]`http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/SemanticWebSearchEngines`, last checked on 24.11.2010
[5]The ontology space can be found at: `http://cupboard.open.ac.uk:8081/cupboard/Experiment1`, last checked on 24.11.2010



Search results for: Fish

http://ontosem.org/#fish
http://www.co-ode.org/ontologies/galen#Fish

subClassOf Vertebrate — Add relation from Fish
subClassOf NonHuman — Add relation from Fish
Salmon **subClassOf** — Add relation to Fish

http://paoli.open.ac.uk/watson-cache/9/feb/1411/72815/d669a668d7/393dee7719f320bbe#Fish
http://paoli.open.ac.uk/watson-cache/5/4b6/466d/b79a5/f1f2298c82/d30800bfaeeff211a#Fish
http://paoli.open.ac.uk/watson-cache/3/629/26fc/026de/786c7ec1aa/00efa2b5f7b21cd1e#Fish
http://paoli.open.ac.uk/watson-cache/e/aa9/02c9/cd6d9/318c843642/9e583ebfa89145d54#Fish
http://purl.org/obo/owl/MESH#MESH_B.02.493
http://paoli.open.ac.uk/watson-cache/f/776/894f/185d1/feead96480/31447118fef369779#Fish
http://paoli.open.ac.uk/watson-cache/c/7a4/11d2/d58c5/c6f902b40b/2bdd54d158be86601#Fish
http://kmi-web05.open.ac.uk:81/cache/2/b59/4b1c/ef182/1143c6b880/001c735289db7de27#FISH
http://paoli.open.ac.uk/watson-cache/1/f5d/857a/f0cae/8b96c6b3ac/1e19b0c5cf7f2a849#Fish
http://paoli.open.ac.uk/watson-cache/0/339/c2ff/21d76/1013cd189c/557c6d296bdc6957c#Fish
http://www.owl-ontologies.com/unnamed.owl#Fish
http://www.semanticweb.org/ontologies/2007/10/Ontology1194270488796.owl#Fish
http://kmi-web05.open.ac.uk:81/cache/f/eb8/2a00/77558/392e2bfefd/4d3b7577493811ac4#FISH
http://paoli.open.ac.uk/watson-cache/e/36e/2636/75525/eddbd8bdd1/7c00491839e1f4b27#FISH
http://paoli.open.ac.uk/watson-cache/f/48d/d2d6/47bf9/580b858398/2b74cd7b58b52e768#Fish
http://kmi-web05.open.ac.uk:81/cache/f/716/6313/dd174/968deb3007/ad98de81808e1d863#Fish
http://www.phinformatics.org/Assets/Ontology/umls.owl#Fish
http://onto.cs.yale.edu:8080/umls/UMLSinDAML/NET/SRDEF#Fish

Figure 14.1: This screenshot shows the Watson plugin displaying results within the NeOn Toolkit

is presented with quality information on the ontologies in the Cupboard plugin, we tested the feature to also see whether users like it.

14.1.3 Tasks to be Executed

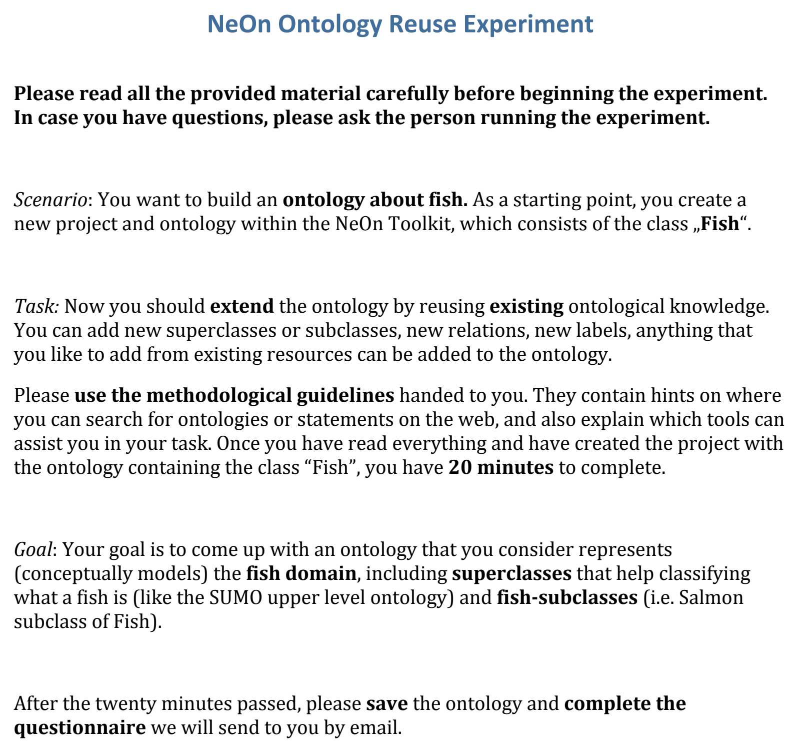
Each user was given a task description (see Figure 14.2) explaining the task to be executed. The task was to extend an ontology containing only the concept "Fish" with ontological content found on the web. The ontology should be extended both with super-classes (ideally aligning the ontology to an upper level ontology) and with sub-classes. A facilitator was present at all times to provide individual help and answer questions related to handling the tools. Questions potentially affecting the outcome of the study (e.g., "which ontology should I reuse") were not answered. The participants were assigned alternating to one of three groups, each having different tool-support at their disposal. Each group was provided an adapted version of the NeOn ontology reuse methodology (del Carmen Suárez-Figueroa *et al.*, 2008), mentioning the tools (if available for that group) that could help in each step. All participants had at most 20 minutes to complete the task, but could stop earlier if they felt the ontology they created was satisfactory for them. We will now explain the distinctions between the three groups.

Group 1

The participants in group one act as a control group. Participants in the control group were not allowed to use the Watson- or Cupboard plugin for NeOn Toolkit. The participants therefore had to create the axioms they wanted to reuse manually within the NeOn Toolkit. They also had to search for content to reuse using the Internet browser. The methodological guidelines (see Figure 14.3) provided a sort of best practice for ontology statement reuse.

Group 2

The second group also had access to the Internet, and in addition the Watson plugin for NeOn Toolkit. The plugin allowed the users to search Watson directly from within the NeOn Toolkit and also to import statements by the click of a button (see Figure 14.1). Only content not available within Watson had to be added manually. The methodological guidelines (see Figure 14.4) were adapted to mention the Watson plugin in the steps that could benefit from it.



NeOn Ontology Reuse Experiment

Please read all the provided material carefully before beginning the experiment. In case you have questions, please ask the person running the experiment.

Scenario: You want to build an **ontology about fish.** As a starting point, you create a new project and ontology within the NeOn Toolkit, which consists of the class „**Fish**“.

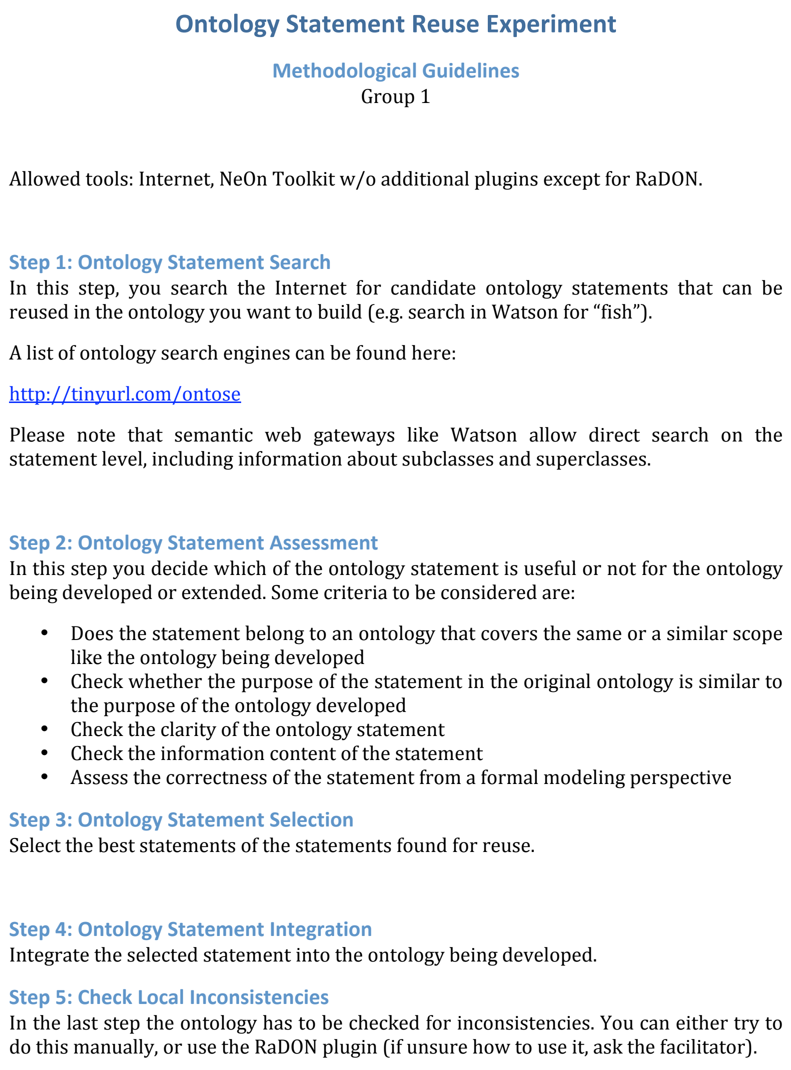
Task: Now you should **extend** the ontology by reusing **existing** ontological knowledge. You can add new superclasses or subclasses, new relations, new labels, anything that you like to add from existing resources can be added to the ontology.

Please **use the methodological guidelines** handed to you. They contain hints on where you can search for ontologies or statements on the web, and also explain which tools can assist you in your task. Once you have read everything and have created the project with the ontology containing the class "Fish", you have **20 minutes** to complete.

Goal: Your goal is to come up with an ontology that you consider represents (conceptually models) the **fish domain**, including **superclasses** that help classifying what a fish is (like the SUMO upper level ontology) and **fish-subclasses** (i.e. Salmon subclass of Fish).

After the twenty minutes passed, please **save** the ontology and **complete the questionnaire** we will send to you by email.

Figure 14.2: This document was given to all participants of the experiments. It lays out the task they were asked to do.



Ontology Statement Reuse Experiment

Methodological Guidelines
Group 1

Allowed tools: Internet, NeOn Toolkit w/o additional plugins except for RaDON.

Step 1: Ontology Statement Search
In this step, you search the Internet for candidate ontology statements that can be reused in the ontology you want to build (e.g. search in Watson for "fish").

A list of ontology search engines can be found here:

http://tinyurl.com/ontose

Please note that semantic web gateways like Watson allow direct search on the statement level, including information about subclasses and superclasses.

Step 2: Ontology Statement Assessment
In this step you decide which of the ontology statement is useful or not for the ontology being developed or extended. Some criteria to be considered are:

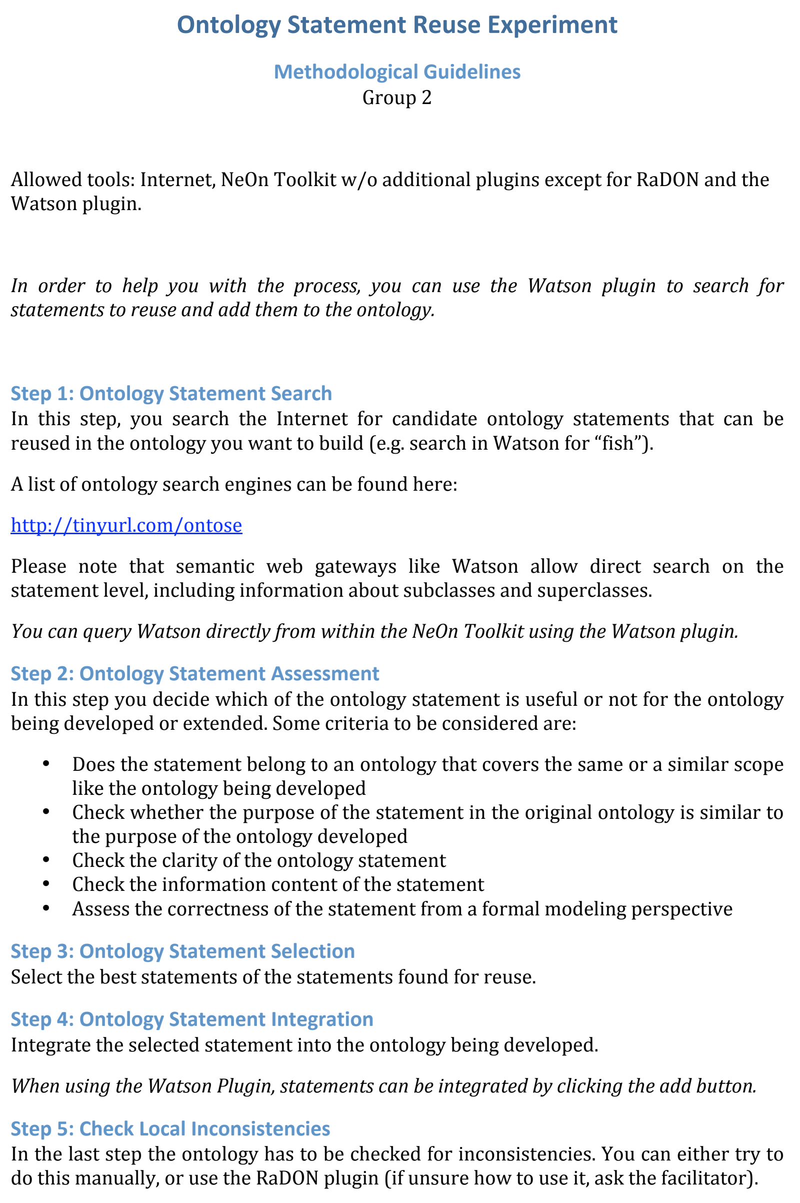
- Does the statement belong to an ontology that covers the same or a similar scope like the ontology being developed
- Check whether the purpose of the statement in the original ontology is similar to the purpose of the ontology developed
- Check the clarity of the ontology statement
- Check the information content of the statement
- Assess the correctness of the statement from a formal modeling perspective

Step 3: Ontology Statement Selection
Select the best statements of the statements found for reuse.

Step 4: Ontology Statement Integration
Integrate the selected statement into the ontology being developed.

Step 5: Check Local Inconsistencies
In the last step the ontology has to be checked for inconsistencies. You can either try to do this manually, or use the RaDON plugin (if unsure how to use it, ask the facilitator).

Figure 14.3: The methodological guidelines for group 1.



Ontology Statement Reuse Experiment

Methodological Guidelines

Group 2

Allowed tools: Internet, NeOn Toolkit w/o additional plugins except for RaDON and the Watson plugin.

In order to help you with the process, you can use the Watson plugin to search for statements to reuse and add them to the ontology.

Step 1: Ontology Statement Search

In this step, you search the Internet for candidate ontology statements that can be reused in the ontology you want to build (e.g. search in Watson for "fish").

A list of ontology search engines can be found here:

http://tinyurl.com/ontose

Please note that semantic web gateways like Watson allow direct search on the statement level, including information about subclasses and superclasses.

You can query Watson directly from within the NeOn Toolkit using the Watson plugin.

Step 2: Ontology Statement Assessment

In this step you decide which of the ontology statement is useful or not for the ontology being developed or extended. Some criteria to be considered are:

- Does the statement belong to an ontology that covers the same or a similar scope like the ontology being developed
- Check whether the purpose of the statement in the original ontology is similar to the purpose of the ontology developed
- Check the clarity of the ontology statement
- Check the information content of the statement
- Assess the correctness of the statement from a formal modeling perspective

Step 3: Ontology Statement Selection

Select the best statements of the statements found for reuse.

Step 4: Ontology Statement Integration

Integrate the selected statement into the ontology being developed.

When using the Watson Plugin, statements can be integrated by clicking the add button.

Step 5: Check Local Inconsistencies

In the last step the ontology has to be checked for inconsistencies. You can either try to do this manually, or use the RaDON plugin (if unsure how to use it, ask the facilitator).

Figure 14.4: The methodological guidelines for group 2.

Group 3

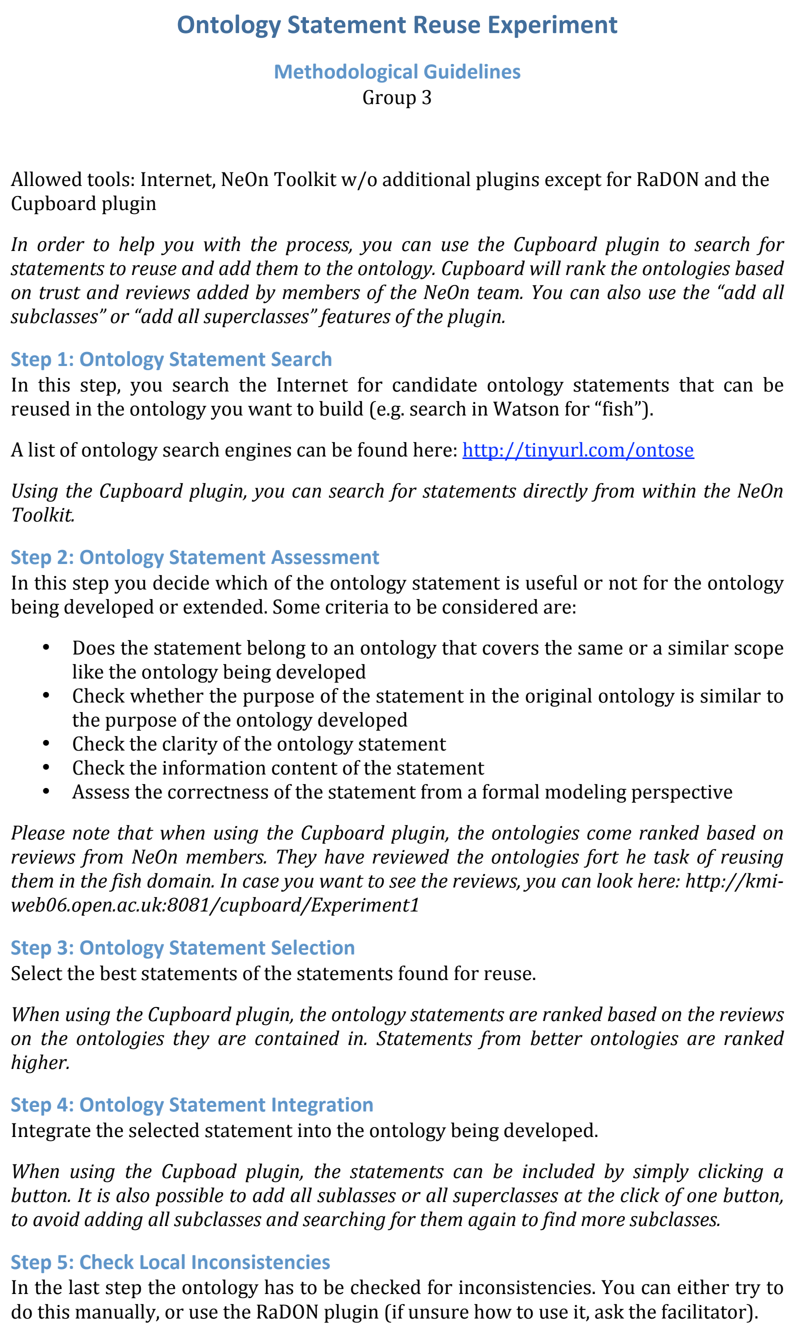
The third group had access to the Internet, and in addition to the Cupboard plugin. The Watson plugin was not available. The Cupboard plugin allows to search for ontologies within Cupboard, and to import statements (or multiple statements) with one click. Furthermore, an overall rating is displayed for the ontologies based on reviews written within Cupboard. The ontologies are ordered based on that rating (see Figure 10.2). In the alpha version of the plugin used for the experiment, advanced features such as local trust based ratings or displaying the reviews from within the plugin were not available. The methodological guidelines (see Figure 14.5) were adapted to mention the Cupboard plugin in the steps that could benefit from it.

14.1.4 Role of the Facilitator

The facilitator was given guidelines (see Figures 14.6 and 14.7), explaining how to conduct the experiment. The facilitator's role was to find participants and assign them to one of the three groups. Furthermore, the facilitator handed out a printed version of the experiment's task description and of the methodological guidelines. The hardware was also set up by the facilitator. The facilitator was responsible to take care of screen-capturing and saving the ontologies. During the experiment, questions regarding the handling of the tools were also answered by the facilitator. After the experiment ended, the facilitator gave a demonstration of the Cupboard plugin to participants from groups 1 and 2. Ultimately, the facilitator was also responsible for sending the questionnaire to the participants and collecting them afterwards.

14.2 Running the Experiment

When a participant was found and assigned to one of the three groups, the facilitator handed out the task description and the methodological guidelines for the respective group. After the participant had read the two documents, the initial step was creating an ontology with the concept "Fish" using the NeOn Toolkit. The creation of the ontology and concept were not part of the experiment, but rather an initial step to ensure the user could work with the NeOn Toolkit (e.g., create classes). After this initial step was finished and the participant was ready, the screen-capturing was started, and time was measured. After 20 minutes (users could stop earlier if they felt the task was completed) the ontology was saved and the screen-capturing stopped. Users from group 1 or 2 were given a quick demonstration of the Cupboard plugin after they finished the task. Afterwards the questionnaire was sent to the participants to gather feedback.



Ontology Statement Reuse Experiment

Methodological Guidelines
Group 3

Allowed tools: Internet, NeOn Toolkit w/o additional plugins except for RaDON and the Cupboard plugin

In order to help you with the process, you can use the Cupboard plugin to search for statements to reuse and add them to the ontology. Cupboard will rank the ontologies based on trust and reviews added by members of the NeOn team. You can also use the "add all subclasses" or "add all superclasses" features of the plugin.

Step 1: Ontology Statement Search
In this step, you search the Internet for candidate ontology statements that can be reused in the ontology you want to build (e.g. search in Watson for "fish").

A list of ontology search engines can be found here: http://tinyurl.com/ontose

Using the Cupboard plugin, you can search for statements directly from within the NeOn Toolkit.

Step 2: Ontology Statement Assessment
In this step you decide which of the ontology statement is useful or not for the ontology being developed or extended. Some criteria to be considered are:

- Does the statement belong to an ontology that covers the same or a similar scope like the ontology being developed
- Check whether the purpose of the statement in the original ontology is similar to the purpose of the ontology developed
- Check the clarity of the ontology statement
- Check the information content of the statement
- Assess the correctness of the statement from a formal modeling perspective

Please note that when using the Cupboard plugin, the ontologies come ranked based on reviews from NeOn members. They have reviewed the ontologies fort he task of reusing them in the fish domain. In case you want to see the reviews, you can look here: http://kmi-web06.open.ac.uk:8081/cupboard/Experiment1

Step 3: Ontology Statement Selection
Select the best statements of the statements found for reuse.

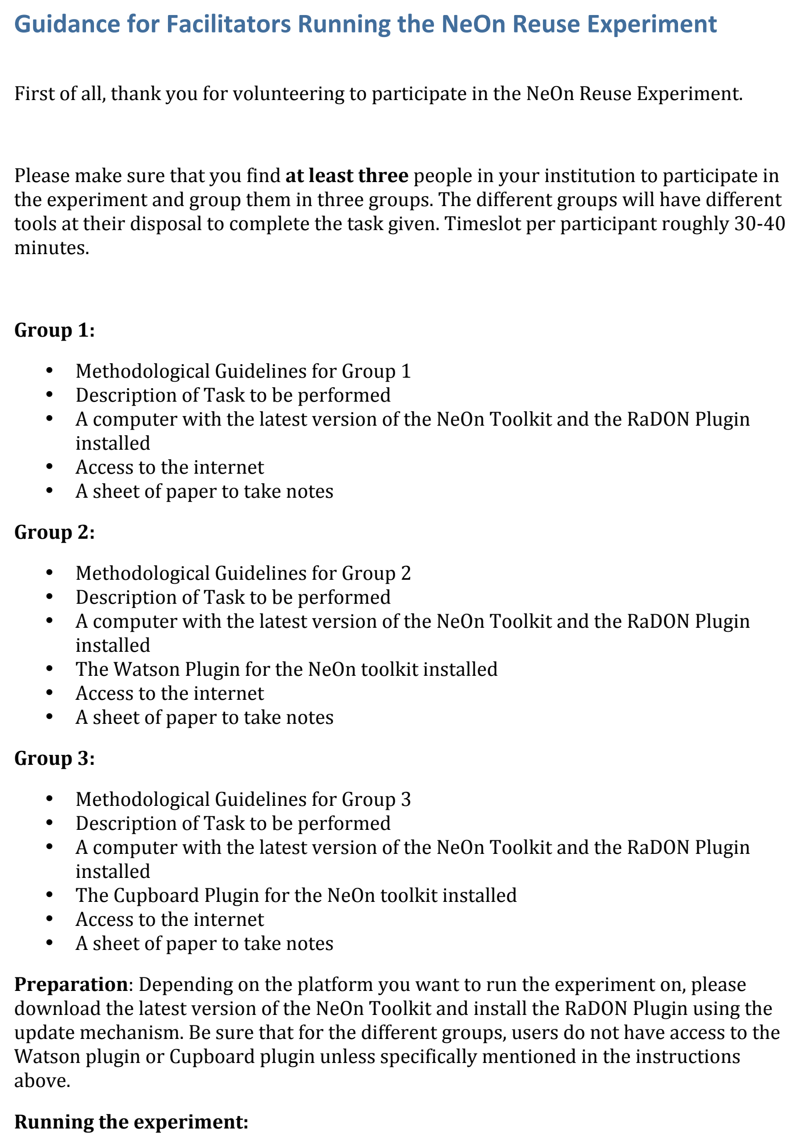
When using the Cupboard plugin, the ontology statements are ranked based on the reviews on the ontologies they are contained in. Statements from better ontologies are ranked higher.

Step 4: Ontology Statement Integration
Integrate the selected statement into the ontology being developed.

When using the Cupboad plugin, the statements can be included by simply clicking a button. It is also possible to add all sublasses or all superclasses at the click of one button, to avoid adding all subclasses and searching for them again to find more subclasses.

Step 5: Check Local Inconsistencies
In the last step the ontology has to be checked for inconsistencies. You can either try to do this manually, or use the RaDON plugin (if unsure how to use it, ask the facilitator).

Figure 14.5: The methodological guidelines for group 3.



Guidance for Facilitators Running the NeOn Reuse Experiment

First of all, thank you for volunteering to participate in the NeOn Reuse Experiment.

Please make sure that you find **at least three** people in your institution to participate in the experiment and group them in three groups. The different groups will have different tools at their disposal to complete the task given. Timeslot per participant roughly 30-40 minutes.

Group 1:

- Methodological Guidelines for Group 1
- Description of Task to be performed
- A computer with the latest version of the NeOn Toolkit and the RaDON Plugin installed
- Access to the internet
- A sheet of paper to take notes

Group 2:

- Methodological Guidelines for Group 2
- Description of Task to be performed
- A computer with the latest version of the NeOn Toolkit and the RaDON Plugin installed
- The Watson Plugin for the NeOn toolkit installed
- Access to the internet
- A sheet of paper to take notes

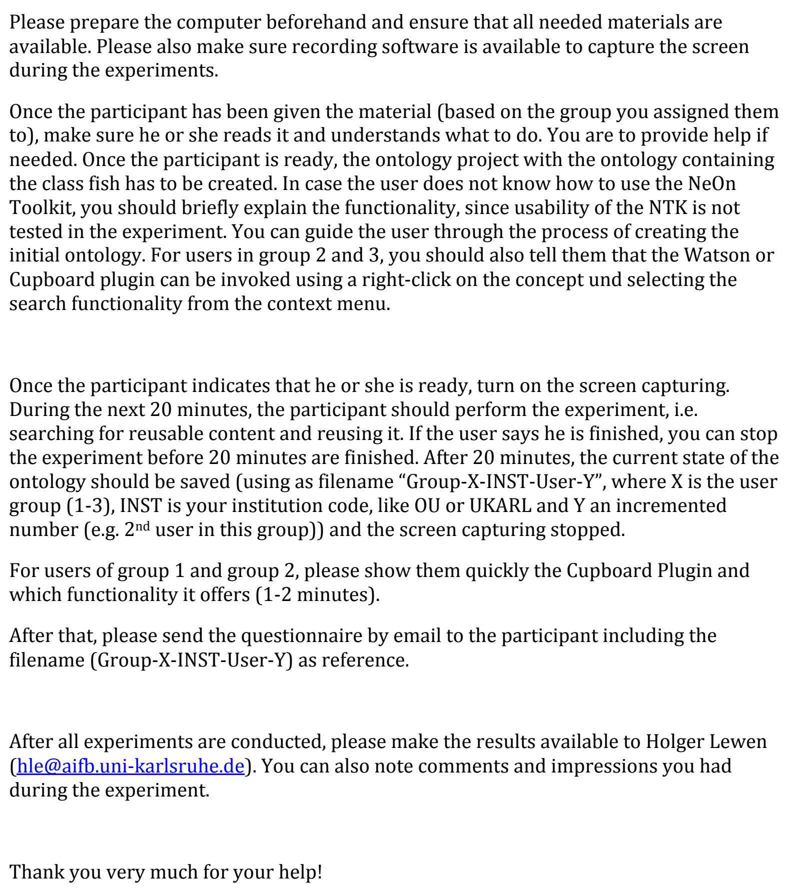
Group 3:

- Methodological Guidelines for Group 3
- Description of Task to be performed
- A computer with the latest version of the NeOn Toolkit and the RaDON Plugin installed
- The Cupboard Plugin for the NeOn toolkit installed
- Access to the internet
- A sheet of paper to take notes

Preparation: Depending on the platform you want to run the experiment on, please download the latest version of the NeOn Toolkit and install the RaDON Plugin using the update mechanism. Be sure that for the different groups, users do not have access to the Watson plugin or Cupboard plugin unless specifically mentioned in the instructions above.

Running the experiment:

Figure 14.6: First page of the facilitator guidelines.



Please prepare the computer beforehand and ensure that all needed materials are available. Please also make sure recording software is available to capture the screen during the experiments.

Once the participant has been given the material (based on the group you assigned them to), make sure he or she reads it and understands what to do. You are to provide help if needed. Once the participant is ready, the ontology project with the ontology containing the class fish has to be created. In case the user does not know how to use the NeOn Toolkit, you should briefly explain the functionality, since usability of the NTK is not tested in the experiment. You can guide the user through the process of creating the initial ontology. For users in group 2 and 3, you should also tell them that the Watson or Cupboard plugin can be invoked using a right-click on the concept und selecting the search functionality from the context menu.

Once the participant indicates that he or she is ready, turn on the screen capturing. During the next 20 minutes, the participant should perform the experiment, i.e. searching for reusable content and reusing it. If the user says he is finished, you can stop the experiment before 20 minutes are finished. After 20 minutes, the current state of the ontology should be saved (using as filename "Group-X-INST-User-Y", where X is the user group (1-3), INST is your institution code, like OU or UKARL and Y an incremented number (e.g. 2nd user in this group)) and the screen capturing stopped.

For users of group 1 and group 2, please show them quickly the Cupboard Plugin and which functionality it offers (1-2 minutes).

After that, please send the questionnaire by email to the participant including the filename (Group-X-INST-User-Y) as reference.

After all experiments are conducted, please make the results available to Holger Lewen (hle@aifb.uni-karlsruhe.de). You can also note comments and impressions you had during the experiment.

Thank you very much for your help!

Figure 14.7: Second page of the facilitator guidelines.

14.2.1 Questionnaire

The questionnaire was sent to all participants with the purpose of finding out basic information about the participants (e.g., their level of expertise, understanding of the task) and gathering feedback on the experiment. The questions can be found in Figures 14.8, 14.9, and 14.10. Because the answers are all subjective statements, they provide a good insight on how well the tool was received by the participants, and make sure the participants understood the task given to them. In addition to the results from the questionnaire, we analyzed the videos and ontologies created by each participant.

14.2.2 Preparation of Cupboard for the Experiment

Since gathering reviews for all ontologies in Watson was unrealistic in the time available for preparing the user study, we decided to focus on a limited subset of the Watson corpus. We therefore searched Watson for all ontologies containing the concept "fish" and uploaded them to a specific Cupboard space called "Experiment1". We then asked qualified members of the NeOn project to review these ontologies. The ontologies with their reviews can be found in the Experiment1 Cupboard space.[6]

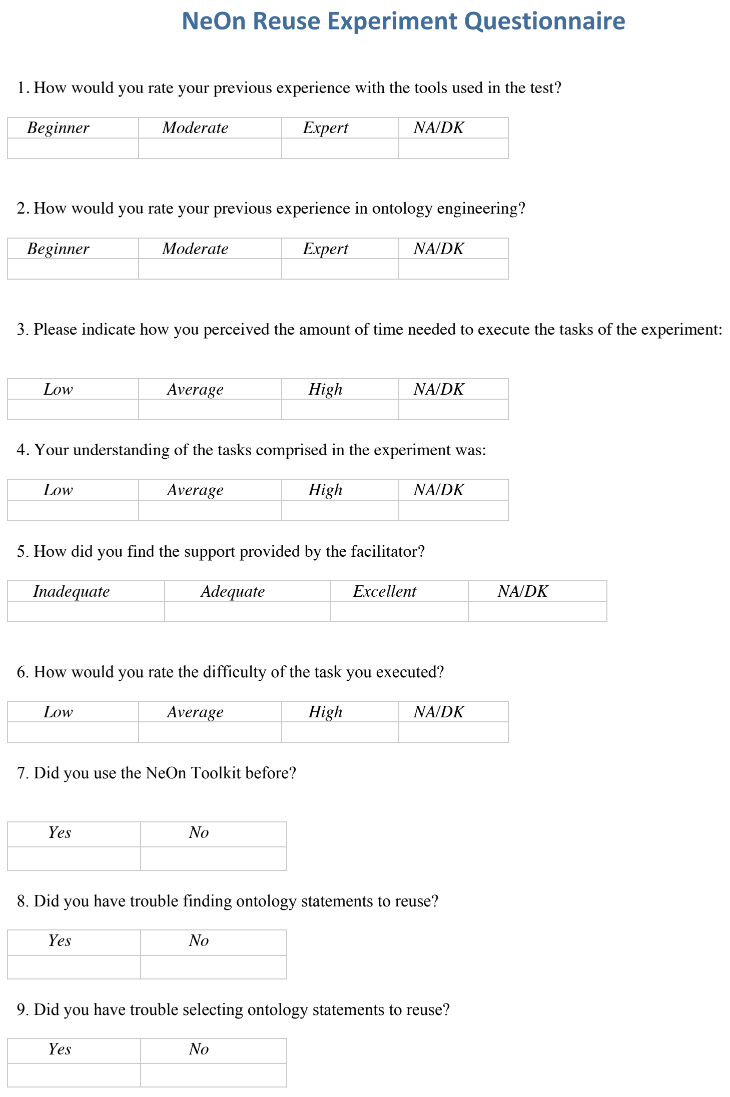
The reviewers could also add trust or distrust statements to other reviewers. For the experiment we decided to use the average over the five aspects (meaning all weights μ_k were given the same value) based on global trust to compute the overall rating for each ontology. This had the advantage that the ranking order and results was the same for all users, which allowed for a better comparison between users of the three groups.

14.2.3 Hardware and Software Used

The experiments were conducted on a 15.4 inch MacBook Pro running MacOS X Leopard. All groups were using the NeOn Toolkit version 1.2.2 B904 extended for Mac. This NeOn Toolkit version was the latest available release at the time of the experiment. Since the functionality used for the experiments did not change in newer versions of the NeOn Toolkit, the results still hold for the current version of the NeOn Toolkit.

Group 2 already had the Watson plugin for NeOn Toolkit installed, and group 3 had the Cupboard plugin for NeOn Toolkit installed.

[6] `http://cupboard.open.ac.uk:8081/cupboard/Experiment1`, last checked on 24.11.2010



NeOn Reuse Experiment Questionnaire

1. How would you rate your previous experience with the tools used in the test?

Beginner	*Moderate*	*Expert*	*NA/DK*

2. How would you rate your previous experience in ontology engineering?

Beginner	*Moderate*	*Expert*	*NA/DK*

3. Please indicate how you perceived the amount of time needed to execute the tasks of the experiment:

Low	*Average*	*High*	*NA/DK*

4. Your understanding of the tasks comprised in the experiment was:

Low	*Average*	*High*	*NA/DK*

5. How did you find the support provided by the facilitator?

Inadequate	*Adequate*	*Excellent*	*NA/DK*

6. How would you rate the difficulty of the task you executed?

Low	*Average*	*High*	*NA/DK*

7. Did you use the NeOn Toolkit before?

Yes	*No*

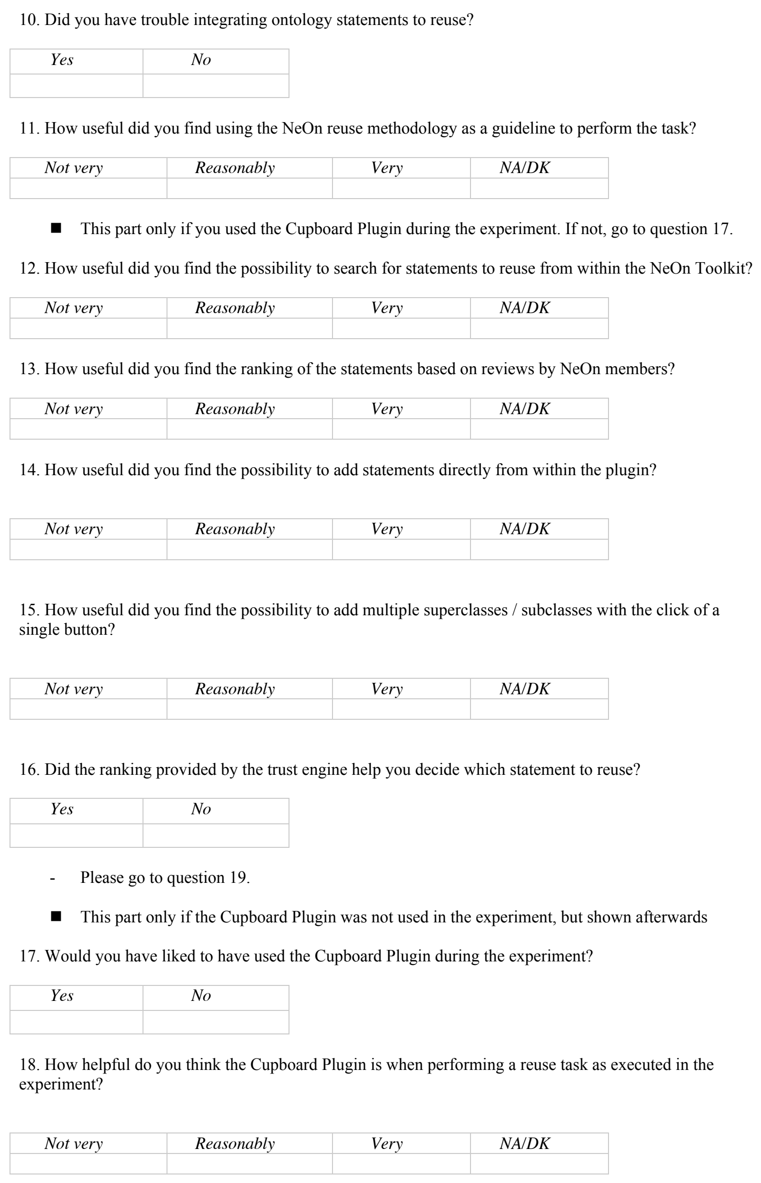
8. Did you have trouble finding ontology statements to reuse?

Yes	*No*

9. Did you have trouble selecting ontology statements to reuse?

Yes	*No*

Figure 14.8: First page of the experiment questionnaire.



10. Did you have trouble integrating ontology statements to reuse?

Yes	*No*

11. How useful did you find using the NeOn reuse methodology as a guideline to perform the task?

Not very	*Reasonably*	*Very*	*NA/DK*

- ■ This part only if you used the Cupboard Plugin during the experiment. If not, go to question 17.

12. How useful did you find the possibility to search for statements to reuse from within the NeOn Toolkit?

Not very	*Reasonably*	*Very*	*NA/DK*

13. How useful did you find the ranking of the statements based on reviews by NeOn members?

Not very	*Reasonably*	*Very*	*NA/DK*

14. How useful did you find the possibility to add statements directly from within the plugin?

Not very	*Reasonably*	*Very*	*NA/DK*

15. How useful did you find the possibility to add multiple superclasses / subclasses with the click of a single button?

Not very	*Reasonably*	*Very*	*NA/DK*

16. Did the ranking provided by the trust engine help you decide which statement to reuse?

Yes	*No*

- Please go to question 19.

- ■ This part only if the Cupboard Plugin was not used in the experiment, but shown afterwards

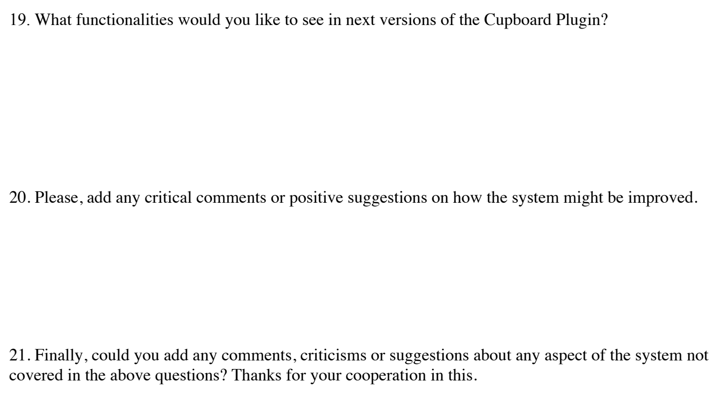
17. Would you have liked to have used the Cupboard Plugin during the experiment?

Yes	*No*

18. How helpful do you think the Cupboard Plugin is when performing a reuse task as executed in the experiment?

Not very	*Reasonably*	*Very*	*NA/DK*

Figure 14.9: Second page of the experiment questionnaire.



19. What functionalities would you like to see in next versions of the Cupboard Plugin?

20. Please, add any critical comments or positive suggestions on how the system might be improved.

21. Finally, could you add any comments, criticisms or suggestions about any aspect of the system not covered in the above questions? Thanks for your cooperation in this.

Figure 14.10: Third page of the experiment questionnaire.

14.3 Results

In this section we present the results of the experiment, with the analysis following in the next section. We will both provide a table displaying information about the ontologies produced, and tables summarizing the results of the questionnaires. In order to keep the evaluation transparent, the videos, ontologies and questionnaires can be found online.[7]

14.3.1 Ontologies

In order to somehow quantify the ontologies produced by the different groups, we took a look at both the number of axioms and the quality of the ontology with respect to good engineering practices. Table 14.1 presents a table with information on the ontologies engineered during the experiment. In the table, "different sources used" refers to the number of ontologies from which statements were reused. "Self-Created Axioms" refers to axioms which were not found in another ontology, but created from scratch or based on knowledge acquired from other non-ontological sources.

14.3.2 Questionnaire Results

To gather demographic data about our users and get feedback, each participant had to fill out a questionnaire after the experiment was finished. As can be seen in Figures 14.8, 14.9, and 14.10, some questions are group specific (only for participants in group 3, or only for participants not in group 3).

We have created tables for the questionnaire results, each also containing the distribution within a group and over all groups in percent. Table 14.2 provides the responses for questions 1 through 7, while Table 14.3 covers questions 8 through 11. Questions 12 through 16 were only answered by group 3. The results for these questions can be found in Table 14.4. The remaining 2 questions were only answered by groups 1 and 2, and the results can be found in Table 14.5.

14.4 Analysis

In order to draw conclusions from the experiment, we analysed both the questionnaire and the ontologies produced. We will start by giving some basic information about our

[7]The resulting files can be found at: `http://people.aifb.kit.edu/hle/thesis/user-experiment/`, last checked on 24.11.2010.

Table 14.1: Quantitative information on the ontologies created in the experiment.

Group	User in Group	Class Count	Object Property Count	SubClass Count	Equivalent Class Count	Different Sources Used	Self-Created Axioms
1	1	7	3	5	0	5	2
1	2	6	0	2	1	4	1
1	3	10	1	8	0	2	7
1	4	14	1	12	0	1	12
1	5	5	0	2	2	3	0
1	6	11	0	8	2	3	0
2	1	49	1	45	0	7	6
2	2	141	0	140	0	10	0
2	3	16	1	13	0	2	3
2	4	131	0	132	0	4	0
2	5	76	0	75	0	8	0
2	6	24	0	23	0	14	0
2	7	20	0	19	0	5	0
3	1	1429	0	1431	0	3	0
3	2	581	0	582	0	3	0
3	3	591	0	593	0	2	0
3	4	863	0	877	0	3	0
3	5	591	0	293	0	2	0
3	6	592	0	594	0	2	1
3	7	591	0	593	0	3	0

Table 14.2: Questionnaire Results Questions 1 – 7

Q1: How would you rate your previous experience with the tools used in the test?				
	Group 1	Group 2	Group 3	Overall
Beginner	3 (50%)	4 (57%)	5 (71%)	12 (60%)
Moderate	2 (33%)	3 (43%)	2 (29%)	7 (35%)
Expert	1 (17%)	0 (0%)	0 (0%)	1 (5%)

Q2: How would you rate your previous experience in ontology engineering?				
	Group 1	Group 2	Group 3	Overall
Beginner	1 (17%)	2 (29%)	2 (29%)	5 (25%)
Moderate	1 (17%)	5 (71%)	5 (71%)	11 (55%)
Expert	4 (67%)	0 (0%)	0 (0%)	4 (20%)

Q3: Please indicate how you perceived the amount of time needed to execute the tasks of the experiment				
	Group 1	Group 2	Group 3	Overall
Low	2 (33%)	2 (29%)	6 (86%)	10 (50%)
Average	3 (50%)	3 (43%)	1 (14%)	7 (35%)
High	1 (17%)	2 (29%)	0 (0%)	3 (15%)

Q4: Your understanding of the tasks comprised in the experiment was?				
	Group 1	Group 2	Group 3	Overall
Low	0 (0%)	1 (14%)	0 (0%)	1 (5%)
Average	3 (50%)	0 (0%)	0 (0%)	3 (15%)
High	3 (50%)	6 (86%)	7 (100%)	16 (80%)

Q5: How did you find the support provided by the facilitator?				
	Group 1	Group 2	Group 3	Overall
Inadequate	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Adequate	1 (17%)	3 (43%)	1 (14%)	5 (25%)
Excellent	5 (83%)	4 (57%)	6 (86%)	15 (75%)

Q6: How would you rate the difficulty of the task you executed?				
	Group 1	Group 2	Group 3	Overall
Low	2 (33%)	4 (57%)	4 (57%)	10 (50%)
Average	2 (33%)	3 (43%)	2 (29%)	7 (35%)
High	2 (33%)	0 (0%)	1 (14%)	3 (15%)

Q7: Did you use the NeOn Toolkit before				
	Group 1	Group 2	Group 3	Overall
Yes	3 (50%)	4 (57%)	4 (57%)	11 (55%)
No	3 (50%)	3 (43%)	3 (43%)	9 (45%)

Table 14.3: Questionnaire Results Questions 8 – 11

Q8: Did you have trouble finding ontology statements to reuse?				
	Group 1	Group 2	Group 3	Overall
Yes	6 (100%)	1 (14%)	1 (14%)	8 (40%)
No	0 (0%)	6 (86%)	6 (86%)	12 (60%)

Q9: Did you have trouble selecting ontology statements to reuse?				
	Group 1	Group 2	Group 3	Overall
Yes	5 (83%)	5 (71%)	0 (0%)	10 (50%)
No	1 (17%)	2 (29%)	7 (100%)	10 (50%)

Q10: Did you have trouble integrating ontology statements to reuse?				
	Group 1	Group 2	Group 3	Overall
Yes	5 (83%)	1 (14%)	0 (0%)	6 (30%)
No	1 (17%)	6 (86%)	7 (100%)	14 (70%)

Q11: How useful did you find using the NeOn reuse methodology to perform the task?				
	Group 1	Group 2	Group 3	Overall
Not very	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Reasonably	1 (17%)	4 (57%)	1 (14%)	6 (30%)
Very	2 (33%)	6 (86%)	6 (86%)	11 (55%)
NA/DK	3 (50%)	6 (86%)	0 (0%)	3 (15%)

Table 14.4: Questionnaire Results Questions 12 – 16 (only for Group 3)

Q12: How useful did you find the possibility to search for statements to reuse from within the NeOn Toolkit?	
	Group 3
Not very	0 (0%)
Reasonably	0 (0%)
Very	7 (100%)
Q13: How useful did you find the ranking of the statements to reuse based on reviews by NeOn members?	
	Group 3
Not very	0 (0%)
Reasonably	4 (57%)
Very	3 (43%)
Q14: How useful did you find the possibility to add statements directly from within the plugin?	
	Group 3
Not very	0 (0%)
Reasonably	0 (0%)
Very	7 (100%)
Q15: How useful did you find the possibility to add multiple superclasses / subclasses with the click of a single button?	
	Group 3
Not very	0 (0%)
Reasonably	0 (0%)
Very	7 (100%)
Q16: Did the ranking provided by the trust engine help you decide which statements to reuse?	
	Group 3
Yes	6 (86%)
No	1 (14%)

Table 14.5: Questionnaire Results Questions 17 – 18 (only for Groups 1 & 2)

Q17: Would you have liked to have used the Cupboard Plugin during the experiment?			
	Group 1	Group 2	Overall
Yes	6 (100%)	7 (100%)	13 (100%)
No	0 (0%)	0 (0%)	0 (0%)

Q18: How helpful do you think the Cupboard Plugin is when performing a reuse task as executed in the experiment?			
	Group 1	Group 2	Overall
Not very	0 (0%)	0 (0%)	0 (0%)
Reasonably	0 (0%)	0 (0%)	0 (0%)
Very	6 (100%)	7 (100%)	13 (100%)

participants. We had a total of 20 participants from 6 different academic institutions. Most of the participants were PhD students, but we also had 2 postdocs and 1 professor participating. We selected people both from within the NeOn project (4 participants) and outside the NeOn project (16 participants). The idea was to have a heterogeneous group and not to be biased in the selection of participants. As can be seen in Table 14.2, more than half of the total participants gauged their experience with the NeOn Toolkit and other tools as beginner (Question 1). Also more than half of the total participants thought of themselves as beginners with regard to ontology engineering (Question 2). Nevertheless, the vast majority of the users had no trouble understanding the task (Question 3), regardless of whether they were ontology engineering experts or beginners. We will now analyze the results group by group and then relate the different groups to each other.

14.4.1 Group 1

Since group 1 is our control group without tool support, it is supposed to serve as a baseline for how ontology engineers nowadays can reuse ontological content on the Internet without having tool support within the ontology engineering environment. Of the six participants in group 1, four are ontology engineering experts, one is a beginner and one has moderate ontology engineering experience (see Table 14.2 Question 2).

Analysis of the Videos

While analysing the video, it was surprising to see that even expert users had trouble finding ontological statements to reuse and to integrate them into their ontology in the NeOn Toolkit. Most had problems using the ontology search engines on the Internet, despite trying out several different ones. Most of the tools had usability issues, leading to situations were the participants expected a certain action and triggered another. This can be seen in the videos when a user clicks somewhere, only to immediately go back and click another button until the desired outcome is obtained. Most of the participants gave up searching for content to reuse, and started modeling the ontology directly from within the toolkit. Some where using their own knowledge about the fish domain, some were consulting Google[8] or Wikipedia[9]. Most participants did give up on the idea of reusing existing ontological knowledge after trying to find adequate content at the beginning and reverted back to creating the content from scratch.

Analysis of the Questionnaire Results

Analyzing the videos, we got the impression that the participants had trouble finding and integrating statements to reuse. The impression was confirmed by the questionnaire answers of the participants. All of the participants in group 1 said they had trouble finding statements to reuse (see Table 14.3 Question 8) and all but one had trouble integrating found statements (Question 10). Also, all but one participant had problems selecting ontology statements to reuse from the statements found during the search (Question 9). Regarding the methodology, which was presented as a guideline to facilitate the reuse process, half of the participants found it at least reasonably useful, while the other half did not respond (Question 11). Since group 1 contained mainly expert users, these might already have their own methodology internalized, and not rely on the one presented in the experiment.

Analysis of the Resulting Ontologies

When checking the metrics for ontologies produced by group 1 (see Table 14.1), it becomes evident that all of the resulting ontologies are small in size (ranging from 7 to 14 classes). While all participants from group 1 tried to reuse at least some ontology statements they found on the web, the majority started to add their own axioms at some point in the experiment (compare the self-created axioms column). In the Linked Data spirit, 3 participants linked the reused classes to the original class either using the equivalent class axiom, or directly reusing the URI for the reused class. One problem that we also observed mainly with participants in group 2 was the reuse of super-classes from different (upper level) ontologies. For example, in one ontology "Fish" was both a subclass of "Vertebrate" (this statement comes from CYC) and "ColdBloodedVertebrate" (this statement comes from SUMO), with "ColdBloodedVertebrate" being a subclass of "Vertebrate" (coming from SUMO). Here the participant should have corrected the taxonomy by removing the superclass "Vertebrate" from "Fish". Other

[8]`http://www.google.com/search?hl=en&q=fish&btnG=Suche&meta=`, last checked on 24.11.2010
[9]`http://en.wikipedia.org/wiki/Fish`, last checked on 24.11.2010

participants created ontologies that contained only references to the concept “Fish” in other ontologies.

For most of the ontologies created, their purpose was not clear and the requirements from the task description were not fulfilled. It was evident that all participants had problems both finding content to reuse and also assessing and integrating found content into their ontology. This is why most participants started modeling without reusing ontological content. The resulting ontologies can be found online.[10] It is important to note that some of the axioms found in these ontologies were created manually and not reused (the videos show which axioms were reused and which created by hand).

14.4.2 Group 2

More than half of the users in group 2 judged themselves beginners with respect to the NeOn Toolkit and Watson plugin (see Table 14.2 Question 1), but most of them have moderate ontology engineering experience (Question 2).

Analysis of the Videos

While the user interface of the Watson plugin is simple and straight forward to use (see Figure 14.1), most participants had problems selecting the best statements to reuse from the list of results. They browsed the list of results, and then started adding statements from all over the list. It seems that many participants did not take the time to actually assess the found statements and check whether they integrate well into the current ontology (e.g., the statement serves a similar purpose in the original ontology). They rather started adding statements from as many sources as possible. Also people started to look for statements related to reused statements (doing a Watson search on parts of axioms they imported from Watson), not necessarily focusing on the fish domain. In two cases, the participants started to create classes themselves, not using the plugin.

Analysis of the Questionnaire Results

As we expected after analysing the videos, all but 1 participant had no trouble finding potential statements to reuse (see Table 14.3 Question 8), but most had trouble selecting the ontology statement from the list of found statements (Question 9). While triggering the search (and thereby receiving results from Watson) from within the NeOn Toolkit is easy, selecting the best statements from this list is not. Since the ranking of results in Watson (and therefore also in the Watson plugin) is mainly based on the Lucene engine, the quality of the ontologies is not factored into the ranking.

[10] `http://people.aifb.kit.edu/hle/thesis/user-experiment/ontologies/`, last checked on 24.11.2010

Therefore a user had to look at all found statements to know which were good and which not. The integration of found axioms was then easy (Question 10), since axioms could be integrated with the click of a button. All users found the provided methodology at least reasonably useful (Question 11). Note that the methodology was adapted for group 2 to explain in which steps the Watson plugin can be used.

Analysis of the Resulting Ontologies

First of all, all ontologies produced by group 2 (see Table 14.1), are larger in size than those produced by group 1. Also, on average, twice as many different sources were used by participants of group 2 compared to group 1 (7.1 different sources on average in group 2 compared to 3 different sources on average in group 1). The increased size of the ontology, and the bigger number of reused content can easily be explained by the very nature of the Watson plugin. When using the Watson plugin, many results are presented from which statements can then easily be reused.

The problem with the resulting ontologies is that users often blindly reused statements without checking whether they need them in their ontology. Most ontologies found on the Internet were build with a specific use case in mind, so the way the world or a domain is modeled varies based on the use case requirements. Sometimes the taxonomy is very fine grained, while in other cases only relevant information is included. When reusing blindly from too many sources, the resulting ontologies face quality problems. In one ontology for example, “Fish” is a subclass of “AnimalFoods”, “AquaticOrganism” and “Seafood”. In another ontology “Fish” is a subclass of “AquaticOrganism”, “MarineAnimal”, “Organic”, “Seafood” and “Vertebrate”. Normally one would expect that these super-classes would themselves be in some sort of hierarchy, if they were at all needed within one ontology. As said before, there are good reasons to have each of them as a superclass in one ontology, depending on the purpose of the ontology, but when combined, the hierarchy does not make much sense anymore. In another ontology, which described fish dishes, “Fish” is a subclass of “NonHuman”. It is unclear why this statement would be needed in this context. Most of the ontologies created face similar problems. The users reused content from various sources, but without assessing which statements they might need. So one can say that reusing is easy with the Watson plugin, but knowing what to reuse is still hard.

14.4.3 Group 3

Most of the users in group 3 judged themselves beginners with respect to the NeOn Toolkit and the Cupboard plugin (see Table 14.2 Question 1), but the majority of them has moderate ontology engineering experience (Question 2).

Analysis of the Videos

Most of the users finished the task very quickly, reusing mostly two or more of the first four ontologies in the result list. The users also heavily used the “add all superclasses” and “add whole sub-branch” feature. This feature was added to the Cupboard plugin, since the ontologies in Cupboard can be reviewed, and the ratings seen from within the plugin. We felt it was more secure if a user knew an ontology was rated 4 stars and then chose to add more content based on a single statement than in the case of the Watson plugin. Since the user can only see one axiom in both Cupboard and the Watson plugin, a certain trust should be placed in the ontology before blindly reusing statements. This was offered in the form of ontology reviews and ratings from NeOn members. These reviews and ratings also influenced the ranking of the results within the plugin.

Analysis of the Questionnaire Results

The questionnaire answers confirmed (as we anticipated from analyzing the videos) that neither finding, nor selecting or integrating ontology statements posed a problem for participants of group 3. All but one participant said they did not have trouble finding ontology statements to reuse (see Table 14.3 Question 8), and none had trouble selecting it (Question 9), or integrating it (Question 10) into their ontology. All but one users found the provided methodology very useful (Question 11). Note that the methodology was adapted for group 3 to explain where the Cupboard plugin can be used.

Also the time needed to execute the task of the experiment was perceived low by all but one participant (see Table 14.2 Question 3), in contrast to participants from groups 1 and 2. Group 3 had some specific questions related to the functionality offered by the Cupboard plugin. All of the participants found the possibility to search for statements to reuse from within the NeOn Toolkit, and adding them (and potentially multiple super- and subclasses) directly very useful (see Table 14.4 Question 12, 14, and 15). All participants found the ranking of the statements based on the reviews at least reasonably useful, 3 found it very useful (Question 13). All but one participant said that the ranking helped in the selection process of the statements (Question 16).

Analysis of the Resulting Ontologies

All ontologies produced by group 3 (see Table 14.1) are big in size, which is mainly due to the possibility of adding multiple statements with one click and the size and structure of the ontologies available in Cupboard. All participants reused at most 3 ontologies, mostly SUMO as upper level ontology, and two ontologies containing information about different fish type and species. Except for one participant, all participants used only SUMO as upper level ontology, and not CYC. One user decided to use both. 2 users used the scientific classification of fish, one together with the information on fish types from another ontology, and one only the scientific classification. The rest chose to only reuse information on the different fish type. In general only ontologies rated highly were reused by the participants, so there is no obvious quality problem with the resulting ontology. In comparison to participants using the Watson plugin, participants using the Cupboard plugin did not blindly add statements from all ontologies, but focused on the best rated ones.

14.4.4 Relation Between Groups

As discussed in the group analysis before, one could say that group 1 had the hardest time finding, selecting and integrating ontology statements to reuse in their ontology. The participants did not manage to produce a suitable ontology satisfying the task requirements given the 20 minutes time limit. Participants of group 2 had no trouble finding and integrating ontology statements thanks to the Watson plugin, but did not know which statements to reuse. This often led to ontologies containing statements from many different ontologies, resulting in serious modeling issues. The last group completed the task very quickly, based on the quality information provided through the Cupboard plugin. After the recording was stopped, all participants of group 1 and 2 were given a quick demonstration of the Cupboard plugin. In the questionnaire, they unanimously stated they thought the plugin was very helpful for the reuse task performed in the experiment (see Table 14.5 Question 18) and that they would have liked to use it during the experiment (Question 19).

14.4.5 Remarks on Linked Data

As stated before, some participants in group 1 manually took care of linking the ontology created the ontology they reused, be it by reusing the URI (which can be problematic) or creating equivalent classes. Both the Watson and the Cupboard plugin offer the functionality to automatically create equivalent classes for each statement that is imported. We have disabled this feature for the experiments, since the equivalent classes show up in the ontology (which results in the user seeing two classes with the same name, one of which is the local class and the other is the equivalent class with the URI), which can confuse the user and hinder usability. For future versions, we plan to automatically provide mapping files linking the created ontology to the ontologies from which statements were reused.

14.5 Statistical Analysis of Questionnaire Results

We have discussed the results of the user experiment without using a statistical analysis of the data up to this point. In this section we employ statistical techniques to verify

Table 14.6: P values for pairwise group comparison using Fisher's exact test with Yate's continuity correction.

Q8: Did you have trouble finding ontology statements to reuse?						
	Gr.1	Gr.2	Gr.3	**p** Gr.1 vs Gr 2.	**p** Gr.1 vs Gr.3	**p** Gr.2 vs Gr.3
Yes	6	1	1	0.0047	0.0047	1
No	0	6	6			

Q9: Did you have trouble selecting ontology statements to reuse?						
	Gr.1	Gr.2	Gr.3	**p** Gr.1 vs Gr 2.	**p** Gr.1 vs Gr.3	**p** Gr.2 vs Gr.3
Yes	5	5	0	1	0.0047	0.021
No	1	2	7			

Q10: Did you have trouble integrating ontology statements to reuse?						
	Gr.1	Gr.2	Gr.3	**p** Gr.1 vs Gr 2.	**p** Gr.1 vs Gr.3	**p** Gr.2 vs Gr.3
Yes	5	1	0	0.0291	0.0047	1
No	1	6	7			

the validity of our findings.

The three most important questions in the questionnaire where questions 8 through 10, which were designed to cover the three basic steps of the ontology reuse process. Question 8 covers search, question 9 selection, and question 10 integration. Table 14.6 provides two-sided p values for pairwise group comparison based on Fisher's exact test (Fisher, 1922) with Yate's continuity correction (Yates, 1934) for Questions 8–10. Fisher's exact test measures the statistical significance in small contingency tables, and is an exact test, meaning the deviation from the null hypothesis can be computed exactly. Yate's continuity correction is used to prevent overestimation of p-values for our small sample size. The null hypothesis for Fisher's exact test is that there is no difference between the two groups, which is rejected for small p values. As is evident from the table, users using either the Watson or the Cupboard plugin had neither trouble finding nor integrating ontological content, compared to users who only could use ontology search engines on the Internet (based on p values, the findings are highly significant for $p = 0.0047$ and significant for $p=0.0291$). Group 3, which had the results ranked based on user ratings, stated they had no problem selecting ontology statements compared to both the Watson group (whose result ranking was based on Lucene), or group 1 which was using a plethora of Semantic Web search engines on the Internet. This result is statistically significant and shows that users have less problems selecting content when offered TS-ORS ranking and scores compared to the current state of the art.

14.6 Conclusion

In the course of the experiment we could see what problems also experienced ontology engineers face when trying to reuse ontological content. The main three problems are finding the statements to reuse, then assessing them and lastly integrating them. The problem of finding statements to reuse is nowadays addressed by many ontology search engines.[11] However, most of them simply store all RDF, OWL, or FOAF documents they find on the Internet without prior quality checks. Also their user interface is still confusing users, and when statements have been found and the decision has been made which statements to reuse, users still have to manually add them to their ontologies. Participants from group 1 faced all these problems, since they had no integrated tool support for search, selection or integration.

The Watson plugin addresses the problems of search and integration, since it uses the Watson API to expose the search functionality directly in the NeOn Toolkit as a plugin, and also allows for easy integration of found statements. But it does not offer quality information on the indexed ontologies, thus leaving the selection process entirely to the user. As seen in the experiment, users could create larger ontologies more easily with the Watson plugin, but still had trouble deciding which statements to reuse. One can say that with the Watson plugin it is easy to reuse ontological content, but it is also easy to create an ontology that does not adhere to good ontology engineering practice.

Cupboard offers users the possibility to review ontologies and trusting reviewers. The TS-ORS computes ratings on each ontology based on weights for each ontology aspect and the best reviews for each ontology–aspect combination. The best reviews are determined by algorithms that compute both local (user-specific) trust and global (not user-specific) trust. The Cupboard plugin uses these ratings to produce a ranking of the results, and also displays the overall rating as stars and numerically. This feature was welcomed by the users and facilitated the selection process. Overall the ranking must have been quite accurate, since the users only reused statements from one of the top four ranked ontologies, even though they checked the statements in the lower ranked ontologies as well. In comparison users of the Watson plugin reused statements from all over the result-list, since the list had no quality specific order.

Of course the quality of an ontology resulting only from reuse can only be as good as the ontologies available to the search engine. And in most of the cases ontologies will have to be adapted for a specific application or use case, and thus differ from existing ontologies. It is thus unfeasible to create them only reusing content. There are many cases, however, where a small ontology can quickly be populated with existing axioms, which can then later be extended, moved, re-factored or deleted. For our experiment

[11] `http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/SemanticWebSearchEngines`, last checked on 24.11.2010

in the fish domain, it was not too difficult finding ontologies with which we populated Cupboard or to find people reviewing them. We believe that taking the extra effort of reviewing the ontologies beforehand goes a long way in facilitating and encouraging reuse. For the participants of group 3, building an initial ontology about fish which they could then later extend and re-engineer was a matter of a few minutes. Compared to the results of the other two groups, we feel confident to say that we have shown that the Cupboard Plugin facilitates reuse by solving all problems normally encountered by the users, namely finding, selecting and integrating existing ontological content. We have also shown that having the quality information and ranking of the TS-ORS available significantly facilitated the reuse process for the participants of our study (see Section 14.5).

Part V

Related Work and Conclusions

Chapter 15

Related Work

While most of the related work has been introduced and referenced in Chapter 2, we summarize the differences between our TS-ORS and related work in this chapter. We start by discussing additional related work in ontology evaluation in Section 15.1. We then cover other trust networks and trust propagation techniques in Section 15.2. The TS-ORS is compared to recommender systems in Section 15.3 and to Open Rating Systems in Section 15.4.

15.1 Related Work in Ontology Evaluation

Apart from the already mentioned evaluation techniques, sometimes a gold standard exists, against which an ontology can be compared automatically by using similarity measures (Maedche and Staab, 2002; Dellschaft and Staab, 2006). This approach is encountered mostly in the area of ontology learning, where the learning procedure is evaluated by comparing the produced ontology against a known or created gold standard. For the case of ontology reuse, this approach is not relevant, since if the gold standard is known, there is no need to find other ontologies.

Vrandecic extended Gangemi's framework (Vrandecic, 2010) for ontology evaluation. He identifies eight criteria according to which ontologies can be evaluated:

- *Accuracy:* States whether the knowledge about a domain is modeled accurately by the axioms of an ontology.
- *Adaptability:* Measures how well an ontology can be adapted to fit its intended usage. For example, it should be possible to extend an ontology monotonically, i.e., without the need to remove axioms.
- *Clarity:* Measures how well an ontology communicates its intended meaning in terms of its defined terms. Ideally, definitions are objective and context independent. Difficult axioms should be documented.
- *Completeness:* Measures whether the domain of interest is covered appropriately, that means, whether all questions an ontology should be able to answer can be

answered. The different aspects of completeness are completeness with regards to the language, the domain, or the applications requirements.

- *Computational efficiency:* Measures the ability of employed tools and applications to work with an ontology. This covers required reasoning time for fulfilling required tasks. Related to the reasoning time is the use of certain type of axioms, that can slow down certain reasoners.
- *Conciseness:* States if an ontology includes irrelevant axioms that are not needed for its intended usage. For example, if only a certain domain should be covered, axioms covering other domains are irrelevant and make the ontology less concise.
- *Consistency:* Describes whether an ontology allows for contradictions. Logical consistency is one aspect, alongside the consistency of formal and informal descriptions in an ontology. For consistency, also the compliance with other principles can be tested. One example is an taxonomy's adherence to the OntoClean constraints (Guarino and Welty, 2002; Guarino and Welty, 2004).
- *Organizational fitness:* This criteria describes how easily an ontology can be deployed within an organization. Deployment can be facilitated by providing sufficient meta-data on the ontology, for example on used methodologies or compatible tools. Also the alignment of an ontology to an upper level ontology like DOLCE (Gangemi *et al.*, 2002) can facilitate organizational fitness, for example if all ontologies in an organization share the same upper level ontology.

Since the development of Cupboard preceded Vrandecic's framework, we could not use his framework in our system. His framework is more accurate than the 5 aspects we use, but it has to be kept in mind that an increase of aspects X in the TS-ORS has a direct effect on the computation time needed. The trust computations are performed for each ontology–aspect combination separately, and each increase in X increases the amount of computations necessary by at least the number of ontologies in the system.

In general our approach is different than other ontology evaluation approaches in that we rely on users to provide the evaluations, and not on automatic techniques. Still, automatic techniques can be incorporated into the TS-ORS if their results can be interpreted on the five star rating scale.

15.2 Related Work in Trust Networks and Trust Propagation

Parts of this section are based on (Kubias *et al.*, 2007). There are several reputation systems relying on trust and trust propagation. Normally, trust or reputation is computed by transitively propagating trust over multiple iterations through looped or arbitrary long chains. Models following this procedure are often called "flow models". Some flow models, like Advogato's reputation scheme (Levien, 2003) and Google's PageRank (Page *et al.*, 1998) assume a constant reputation weight for the entire community, and the value is divided among all the members. In these systems, a user can only increase reputation at the cost of others. Other flow models do not require a constant sum of the reputation scores. The EigenTrust model (Kamvar *et al.*, 2003) computes reputations scores based on repeated and iterative multiplication and aggregation of scores along transitive chains until all scores converge to stable values.

Reputation-based trust systems are used for example in Peer-to-Peer (P2P) networks, in Web environments and also on the Semantic Web.

In P2P networks, reputation-based trust is used to tackle the problem of data quality. In contrast to the classical Web infrastructure, a user is both server and client at the same time. Since there are no quality control mechanisms to hinder putting malicious files on the network, trust systems have to be used to act as a quality control and protect the users from loading unreliable content. The idea is that a well-reputed user would not jeopardize his reputation by putting malicious content on the network. Concrete systems for P2P networks have been proposed by (Aberer and Despotovic, 2001; Cornelli *et al.*, 2002; Damiani *et al.*, 2002; Kamvar *et al.*, 2003; Fahrenholtz and Lamersdorf, 2002; Liau *et al.*, 2003; Gupta *et al.*, 2003). Some the systems derive their ranking algorithms from the PageRank (Page *et al.*, 1998) algorithm, like the EigenTrust algorithm (Kamvar *et al.*, 2003) or the P2PRep system (Cornelli *et al.*, 2002). In these systems, the global reputation score for each agent represents the perceived upload quality of a peer. The upload quality sometimes is measured by simply counting the number of successful uploads. Aberer's approach (Aberer and Despotovic, 2001) uses statistical analysis to characterize trust, leading to a more scalable system, since no performance history has to be maintained. With the XRep protocol, (Damiani *et al.*, 2002) uses a method for reputation management that allows an automatic vote using users' feedback on the best host for a given resource. A survey of common problems of reputation-based systems with possible solutions is provided in (Jøsang *et al.*, 2007).

In Web environments trust is usually modeled transitive, meaning that trusting one site or information requires trusting associated sites or information. In general, in Web environments reputation is defined as a measure of trust. A so called "web of trust" (WOT) is created by the reputation information each entity has recorded on another entity. How trust is transferred between hyperlinks on the Web is examined in (Stewart, 1999; Stewart and Zhang, 2003). A similar approach is used in (Gyöngyi *et al.*, 2004) to combat Web spam. Given a set of user assignments on Web pages which state whether a users consider a page spam, their approach TrustRank uses the link structure between Web pages to derive which other Web pages are spam. For this approach to work on Web links, which are by default untyped, the assumption in accordance with (Page *et al.*, 1998) is that all Web links are positive endorsements and thus indications of trust. Since this general assumption does not always hold, (Massa and Hayes, 2005) proposes an HTML extension that allows the authors of Web pages to specify whether a link should be interpreted as positive, negative or neutral. One problem with most models is that they do not consider context and thus do not differentiate between "topic-specific" and referral trust. Methods that do distinguish between the domain of knowledge (i.e., the context) and referral trust, can be found in (Ding *et al.*, 2004b; Ding *et al.*, 2003).

Ideas on how to use trust on the Semantic Web to verify knowledge encoded in the triples can be found in (Bizer and Oldakowski, 2004). As a proof of concept, (Bizer *et al.*, 2005), provides a browser able of filtering content based on a user-defined policy. In (Ding *et al.*, 2003), agents can use both context and reputation to decide which information on the Semantic Web can be trusted. Referral trust is used to collect reputation, but the context is determined using semantic technologies. Ontologies are used to express trust and reputation information in (Golbeck and Hendler, 2004a; Golbeck and Hendler, 2004b). In (Golbeck, 2006a), the Semantic Web is used together with "provenance" to infer trust relations. Provenance in this case refers to details about the source and origin of information that can be used to evaluate trust. Examples are author, citations, or publisher. The provenance relates people with information, and the Semantic Web contains the social network data needed for trust score computation. Another method using provenance and computations over a WOT for information sources selection can be found in (Ding *et al.*, 2005). Here, information about determined provenance is used to find more trusted sources, while taking the concept of ignorance, i.e., not having any trust information, into account. Other key works also taking distrust and robustness to noise into account when computing trust transitively for Web applications are (Richardson *et al.*, 2003; Guha *et al.*, 2004). How to deal with controversial users, i.e., users who are trusted by some and distrusted by others is examined in (Massa and Avesani, 2005).

Jennifer Golbeck (Golbeck, 2005) presents the TidalTrust algorithm that allows to propagate trust inside a social network. She uses 10 discrete values to express trust based on the claim that discrete values are easier to handle for humans than a continuous trust scale. The algorithm is centered around the idea of neighborhood exploration. The concept of distrust however is not covered, meaning it is not possible to explicitly state one does not want to trust another user.

Surveys on trust and also trust propagation can be found in (Ziegler and Lausen, 2005; Golbeck, 2006b; Artz and Gil, 2007).

The approach we use from (Guha *et al.*, 2004) differs from other approaches in that it can handle distrust statements between users. Furthermore, trust in the TS-ORS is subjective, therefore different to credential based approaches which rely objective trust. In peer-to-peer systems, for example, it is easy to verify whether an uploaded file is genuine or not, while a review in the TS-ORS can be helpful for some users, and not helpful for others. Recent work (Leskovec *et al.*, 2010) has investigated the use of machine-learning techniques for predicting whether another user should be trusted or distrusted. Their results seem promising within their evaluation scenario, but their technique is not usable for our algorithms. Within the TS-ORS, we do not only need to know who is trusted by a given user, but we also need trust scores that allow us to rank reviewers. Just knowing who is likely trusted and who is likely distrusted does not suffice, since a ranking of reviewers is needed for the algorithms.

15.3 Relation to Recommender Systems

First of all, it is important to compare the TS-ORS to classical recommender systems (Resnick and Varian, 1997). Recommender systems try to group users into clusters with similar taste and then recommend items also favored by their closest neighbors. Most recommender systems rely on a user's behavior in the past to provide recommendations, without explicitly asking the user for data that might be needed to make good recommendations. First of all, this approach differs from the TS-ORS in that we work exclusively on data explicitly entered by the users, and do not rely on assumptions. For example, a recommender system might deduce that two users are similar when they rate many ontologies in a similar way. The recommender system then would use this relationship in the recommendation process. In the TS-ORS, we rely on explicit trust statements made by the users. In a recommender system, a user cannot simply state that he wants to receive recommendations based on the behavior of another user, but the system tries to infer these relations automatically. Furthermore, recommender systems can easily be attacked by malicious users just copying a users ratings for all objects to be the most similar user, and then rate a bad object highly so it gets recommended to the copied user. For that reason, (Massa and Avesani, 2009) investigated how explicit trust can be incorporated into recommender systems instead of relying only on an analysis of existing data. They conclude that trust can significantly improve recommendations since malicious users can be avoided, and the cold start problem of collaborative filtering can also be circumvented. Furthermore, classical recommender systems do not incorporate the notion of distrust. We could however employ a recommender systems within the TS-ORS to provide recommendations on who a user might want to trust. A user can then choose to take a closer look at the recommended user and then decide whether to trust or distrust this user.

15.4 Relation to Open Rating Systems

Our closest related work is Guha's work on Open Rating Systems (Guha, 2004), and trust propagation (Guha *et al.*, 2004). To overcome limitations of ORS, we have extended the model with ratings on aspects of objects, and provided a comprehensive framework for fine grained topic-specific trust and meta-trust statements. In contrast to Guha, we present a full algorithmic description and complete framework for the computation of personalized ratings based on ratings on aspects of objects and fine-grained user trust.

Chapter 16

Conclusions and Open Questions

The conclusions of this thesis are presented in Section 16.1. The chapter ends with open questions in Section 16.2.

16.1 Conclusions

When first thinking about how to facilitate ontology reuse for the user, we quickly came to the conclusion that of the three general phases of the ontology reuse process—ontology discovery, ontology selection, and ontology integration—ontology selection posed the most difficulties to the end user. Many existing tools provide help in discovering or integrating ontologies. Still, the ontology selection step has not been supported well. Many approaches tried to apply metrics to automatically determine the suitability of an ontology for a given task, but none could compete with the review of a human expert. Therefore, we decided to investigate how best to incorporate user-based ontology reviews into the ontology reuse process.

Open Rating Systems offer mechanisms to compute rankings for ontology reviews and ratings. However, state of the art Open Rating Systems had two major limitations, namely only allowing to review and rate an object in its entirety, and only allowing users to trust each other globally.

In this thesis, we presented our extension of the state of the art Open Rating System model, the Topic-Specific Trust Open Rating System. The TS-ORS allows both multi-aspect object review and fine-granular trust management, including topic-specific trust and meta-trust. We furthermore adapted the TS-ORS for ontology evaluation and reuse, and implemented it in an online ontology repository named Cupboard. Cupboard here acts as a platform where users can upload ontologies and enter reviews on aspects of the ontologies. Furthermore users can express trust towards other users. Our performance benchmark shows that the system is fast enough to be employed at runtime in an ontology repository. Our simulation has shown how the TS-ORS provides more accurate overall ratings for objects, and a better review ranking than the ORS in our motivating scenario.

With the Cupboard plugin for the ontology engineering environment NeOn Toolkit, we provide tool support for all three steps of the ontology reuse process. Users can discover ontologies directly from within their ontology engineering environment. The result list from Cupboard is ranked based on overall ratings of the ontologies, thus reflecting their quality according to a user-based evaluation. The overall rating of an ontology is displayed next to the ontology, and facilitates the selection process for the user. Result integration is also as simple as clicking a button. The Cupboard plugin thus provides tool support for the complete ontology reuse process, and our user study has shown that it facilitates ontology reuse for the end user compared to other systems. When using the Cupboard Plugin, which relies on our TS-ORS for result ranking, users had significantly less problems finding, selecting, and integrating ontologies to reuse.

16.2 Open Questions

In the future it will be important to monitor whether users participate in Cupboard, and which incentives can be provided to reward reviews. We have already provided first ideas in Chapter 7 Section 7.3 on which incentives can be provided to ensure sufficient review data is in the system to attract users. Whether one of these incentives, or a combination of them will provide the best outcome, remains to be seen.

Another open question remains how the TS-ORS performs against recommender systems within explicit systems, and whether a hybrid solution provides most benefit.

We have presented our TS-ORS and its adaptation for the use case of ontology reuse. We expect that other areas can adapt the idea of rating to solve problems in diverse areas, the rating of services being just one example. Determining for which application areas the TS-ORS can be employed, remains another open question.

Part VI

Appendix

List of Tables

List of Figures

Bibliography

Alfarez Abdul-Rahman and Stephen Hailes. Supporting Trust in Virtual Communities. In *HICSS*, 2000.

Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, November 5-10, 2001*, pages 310–317. ACM, 2001.

Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors. *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*. Springer, 2007.

Sudhir Agarwal. *Formal Description of Web Services for Expressive Matchmaking.* PhD thesis, Fakultät für Wirtschaftswissenschaften, Universität Karlsruhe (TH), May 2007.

Harith Alani, Christopher Brewster, and Nigel Shadbolt. Ranking Ontologies with AKTiveRank. In *Proc. of the 5th Int. Semantic Web Conference (ISWC 2006)*, volume 4273 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

Sofia Angeletou, Holger Lewen, and Boris Villazó n Terrazas. D2.2.4 Final version of methods for re-engineering and evaluation. Technical Report D2.2.4, Knowledge Media Institute, the Open University, January 2010.

Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Paslaru Bontas Simperl, editors. *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, volume 5554 of *Lecture Notes in Computer Science*. Springer, 2009.

Donovan Artz and Yolanda Gil. A Survey of Trust in Computer Science and the Semantic Web. *Journal of Web Semantics*, 5(2):58–71, 2007.

Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald. M. Rubin, and Gavin Sherlock. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.

Vijayalakshmi Atluri, editor. *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washingtion, DC, USA, November 18-22, 2002*. ACM, 2002.

Alejandro Peña Ayala. Ontology Agents and Their Applications in the Web-Based Education Systems: Towards an Adaptive and Intelligent Service. In Nguyen and Jain (2009), pages 249–278.

Paul Gustav Heinrich Bachmann. *Die analytische Zahlentheorie / dargestellt von Paul Bachmann.* Teubner, 1894.

Amos Bairoch, Sarah Cohen Boulakia, and Christine Froidevaux, editors. *Data Integration in the Life Sciences, 5th International Workshop, DILS 2008, Evry, France, June 25-27, 2008. Proceedings*, volume 5109 of *Lecture Notes in Computer Science*. Springer, 2008.

Daniel Barrell, Emily Dimmer, Rachael P. Huntley, David Binns, Claire O'Donovan, and Rolf Apweiler. The GOA database in 2009 - an integrated Gene Ontology Annotation resource. *Nucleic Acids Research*, 37(Database-Issue):396–403, 2009.

Kurt Bauknecht, A. Min Tjoa, and Gerald Quirchmayr, editors. *E-Commerce and Web Technologies, Third International Conference, EC-Web 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings*, volume 2455 of *Lecture Notes in Computer Science*. Springer, 2002.

Joyce Berg, John Dickhaut, and Kevin McCabe. Trust, Reciprocity, and Social History. *Games and Economic Behavior*, 10(1):122–142, 1995.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors. *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, volume 5823 of *Lecture Notes in Computer Science*. Springer, 2009.

Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2003.

Christian Bizer and Radoslaw Oldakowski. Using Context- and Content-Based Trust Policies on the Semantic Web. In Feldman et al. (2004a), pages 228–229.

Christian Bizer, Richard Cyganiak, Tobias Gauss, and Oliver Maresch. The TriQL.P browser: Filtering Information using Context-, Content- and Rating-Based Trust Policies. In *Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference*, volume 7, pages 12–20. Citeseer, 2005.

Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked Data on the Web (LDOW2008). In Huai et al. (2008), pages 1265–1266.

Judith A. Blake and Carol J. Bult. Beyond the data deluge: Data integration and bio-ontologies. *Journal of Biomedical Informatics*, 39(3):314–320, 2006.

Stephan Bloehdorn, Marko Grobelnik, Peter Mika, and Duc Thanh Tran, editors. *Proceedings of the Workshop on Semantic Search (SemSearch 2008) at the 5th European Semantic Web Conference (ESWC 2008), Tenerife, Spain, June 2nd, 2008*, volume 334 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

Willem Nico Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Enschede, Enschede, The Netherlands, 1997.

Janez Brank, Marko Grobelnik, and Dunja Mladenić. A Survey of Ontology Evaluation Techniques. In *In In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005*, 2005.

Christopher Brewster, Harith Alani, Srinandan Dasmahapatra, and Yorick Wilks. Data Driven Ontology Evaluation. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)*, pages 164–168, Lisbon, Portugal, 2004. European Language Resources Association.

Dan Brickley and Ramanathan V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

Dan Brickley and Libby Miller, editors. *FOAF Vocabulary Specification 0.98*. Namespace Document, 09 August 2010. Available at `http://xmlns.com/foaf/spec/20100809.html`.

Paul Buitelaar, Thomas Eigner, and Thierry Declerck. OntoSelect: A Dynamic Ontology Library with Support for Ontology Selection. In *Proceedings of the Demo Session at the International Semantic Web Conference*, 2004.

Andrew Burton-Jones, Veda C. Storey, Vijayan Sugumaran, and Punit Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data and Knowledge Engineering*, 55(1):84–102, October 2005.

Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Anni-Yasmin Turhan, and Sergio Tessaris, editors. *Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, near Bozen-Bolzano, Italy, 8-10 June, 2007*, volume 250 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Domenico Lembo, Antonella Poggi, and Riccardo Rosati. MASTRO-I: Efficient Integration of Relational Data through DL Ontologies. In Calvanese et al. (2007a).

Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors. *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*. ACM, 2006.

Paolo Ceravolo, Zhan Cui, Ernesto Damiani, Alex Gusmini, and Marcello Leida. ODDI: Ontology-Driven Data Integration. In Lovrek et al. (2008), pages 517–524.

Artem Chebotko, Shiyong Lu, Farshad Fotouhi, and Anthony Aristar. Ontology-Based Annotation of Multimedia Language Data for the Semantic Web. *CoRR*, abs/0902.3027, 2009.

Gong Cheng, Weiyi Ge, and Yuzhong Qu. Falcons: Searching and Browsing Entities on the Semantic Web. In *Proc. of the 17th Int. Conf. on World Wide Web (WWW 2008)*, pages 1101–1102. ACM, 2008.

Roger H. L. Chiang, Alberto H. F. Laender, and Ee-Peng Lim, editors. *Fifth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2003), New Orleans, Louisiana, USA, November 7-8, 2003*. ACM, 2003.

Chin-Wan Chung, Chong kwon Kim, Won Kim, Tok Wang Ling, and Kwan Ho Song, editors. *Web Communication Technologies and Internet-Related Social Issues - HSI 2003, Second International Conference on Human Society@Internet, Seoul, Korea, June 18-20, 2003, Proceedings*, volume 2713 of *Lecture Notes in Computer Science*. Springer, 2003.

Philipp Cimiano and Sofia Pinto, editors. *Knowledge Engineering and Management by the Masses, 17th International Conference, EKAW 2010, Lisbon, Portugal, October 2010, Proceedings*, volume 6317 of *Lecture Notes in Computer Science*. Springer, 2010.

Dwaine E. Clarke, Jean-Emile Elien, Carl M. Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

James Suamuel Coleman. *Foundations of social theory.* Belknap Press, 1994.

Óscar Corcho and Asunción Gómez-Pérez. A Roadmap to Ontology Specification Languages. In Dieng and Corby (2000), pages 80–96.

Fabrizio Cornelli, Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Choosing reputable servents in a P2P network. In *WWW*, pages 376–386, 2002.

Isabel F. Cruz and Huiyong Xiao. Ontology Driven Data Integration in Heterogeneous Networks. In Tolk and Jain (2009), pages 75–98.

Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness, editors. *Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001*, 2001.

Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors. *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*. Springer, 2006.

Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In Atluri (2002), pages 207–216.

Mathieu d'Aquin and Holger Lewen. Cupboard - A Place to Expose Your Ontologies to Applications and the Community. In Aroyo et al. (2009), pages 913–918.

Mathieu d'Aquin, Peter Haase, Chan Le Duc, and Antoine Zimmermann. D1.1.4 NeOn Formalism for Modularization: Implementation and Evaluation. Technical Report D1.1.4, Knowledge Media Institute, the Open University, October 2008.

Mathieu d'Aquin, Peter Haase, Sebastian Rudolph, Jérôme Euzenat, Antoine Zimmermann, Martin Dzbor, Marta Iglesias, Yves Jacques, Caterina Caracciolo, Carlos Buil Aranda, and Jose Manuel Gomez. D1.1.3 NeOn Formalisms for Modularization: Syntax, Semantics, Algebra. Technical Report D1.1.3, Knowledge Media Institute, the Open University, February 2008.

Mathieu d'Aquin, Enrico Motta, Martin Dzbor, Laurian Gridinoc, Tom Heath, and Marta Sabou. Collaborative Semantic Authoring. *IEEE Intelligent Systems*, 23(3):80–83, 2008.

Mathieu d'Aquin, Enrico Motta, Marta Sabou, Sofia Angeletou, Laurian Gridinoc, Vanessa Lopez, and Davide Guidi. Toward a New Generation of Semantic Web Applications. *IEEE Intelligent Systems*, 23(3):20–28, 2008.

Mathieu d'Aquin, Jérôme Euzenat, Chan Le Duc, and Holger Lewen. Sharing and Reusing Aligned Ontologies with Cupboard. In Gil and Noy (2009), pages 179–180.

Mari del Carmen Suárez-Figueroa, Guadalupe Aguado de Cea, Carlos Buil, Klaas Dellschaft, Mariano Fernández-López, Andrés García, Asunción Gómez-Pérez, German Herrero, Elena Montiel-Ponsoda, Marta Sabou, Boris Villazon-Terrazas, and Zheng Yufei. D5.4.1 NeOn Methodology for Building Contextualized Ontology Networks. Technical Report D5.4.1, Universidad Politécnica de Madrid, February 2008.

Klaas Dellschaft and Steffen Staab. On How to Perform a Gold Standard Based Evaluation of Ontology Learning. In Cruz et al. (2006), pages 228–241.

Rose Dieng and Olivier Corby, editors. *Knowledge Acquisition, Modeling and Management, 12th International Conference, EKAW 2000, Juan-les-Pins, France, October 2-6, 2000, Proceedings*, volume 1937 of *Lecture Notes in Computer Science*. Springer, 2000.

Ying Ding and Dieter Fensel. Ontology Library Systems: The key to successful Ontology Reuse. In Cruz et al. (2001), pages 93–112.

Li Ding, Lina Zhou, and Timothy W. Finin. Trust Based Knowledge Outsourcing for Semantic Web Agents. In *2003 IEEE / WIC International Conference on Web Intelligence, (WI 2003), 13-17 October 2003, Halifax, Canada*, pages 379–387. IEEE Computer Society, 2003.

Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: A Search and Metadata Engine for the Semantic Web. In Grossman et al. (2004), pages 652–659.

Li Ding, Pranam Kolari, Shashidhara Ganjugunte, Tim Finin, and Anupam Joshi. Modeling and Evaluating Trust Network Inference. In *Seventh International Workshop on Trust in Agent Societies at AAMAS 2004*. Citeseer, 2004.

Li Ding, Pranam Kolari, Tim Finin, Anupam Joshi, Yun Peng, and Yelena Yesha. On Homeland Security and the Semantic Web: A Provenance and Trust Aware Inference Framework. In *Proceedings of the AAAI Spring Symposium on AI Technologies for Homeland Security*, 2005.

John Domingue and Chutiporn Anutariya, editors. *The Semantic Web, 3rd Asian Semantic Web Conference, ASWC 2008, Bangkok, Thailand, December 8-11, 2008. Proceedings*, volume 5367 of *Lecture Notes in Computer Science*. Springer, 2008.

Chan Le Duc, Mathieu d'Aquin, Jesus Barrase, Jérôme David, Jérôme Euzenat, Raúl Palma, Rosario Plaza, Marta Sabou, and Boris Villazón-Terrezas. D3.3.2 Matching ontologies for context: The Alignment plug-in. Technical Report D3.3.2, INRIA, February 2008.

Martin Dzbor, Mari del Carmen Suárez-Figueroa, Eva Blomqvist, Holger Lewen, Mauricio Espinoza, Asunción Gómez-Pérez, and Raul Palma. D5.6.2 Experimentation and Evaluation of the NeOn Methodology. Technical Report D5.6.2, Knowledge Media Institute, the Open University, February 2009.

Martin Dzbor, Holger Lewen, and Mari del Carmen Suárez-Figueroa. D5.6.3 Experimentation and Evaluation of the NeOn Methodology. Technical Report D5.6.3, Knowledge Media Institute, the Open University, January 2010.

Allan Ellis and Tatsuya Hagino, editors. *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*. ACM, 2005.

Allan Ellis and Tatsuya Hagino, editors. *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005 - Special interest tracks and posters*. ACM, 2005.

Carl M. Ellison, Bill Frantz, Butler Lampson, Ron L. Rivest, Brian M Thomas Tatu, and Ylönen. SPKI Certificate Theory, 1999.

Carola Eschenbach and Michael Grüninger, editors. *Formal Ontology in Information Systems, Proceedings of the Fifth International Conference, FOIS 2008, Saarbrücken, Germany, October 31st - November 3rd, 2008*, volume 183 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.

Jérôme Euzenat, Adrian Mocan, and François Scharffe. Ontology Alignments. In Hepp et al. (2008), pages 177–206.

Jérôme Euzenat. An API for Ontology Alignment. In McIlraith et al. (2004), pages 698–712.

Jérôme Euzenat. Algebras of Ontology Alignment Relations. In Sheth et al. (2008), pages 387–402.

Dietrich Fahrenholtz and Winfried Lamersdorf. Transactional Security for a Distributed Reputation Management System. In Bauknecht et al. (2002), pages 214–223.

Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors. *Proceedings of the 13th international conference on World Wide Web - Alternate Track Papers & Posters, WWW 2004, New York, NY, USA, May 17-20, 2004*. ACM, 2004.

Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors. *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*. ACM, 2004.

Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors. *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*. Springer, 2003.

Mariano Fernandez-Lopez, Asuncion Gomez-Perez, and Natalia Juristo. METHONTOLOGY: from Ontological Art towards Ontological Engineering. In *Proceedings of the AAAI97 Spring Symposium*, pages 33–40, Stanford, USA, March 1997.

RA Fisher. On the Interpretation of χ 2 from Contingency Tables, and the Calculation of P. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922.

William B. Frakes and Ricardo A. Baeza-Yates, editors. *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992.

Gaihua Fu, Christopher B. Jones, and Alia I. Abdelmoty. Building a Geographical Ontology for Intelligent Spatial Search on the Web. In Hamza (2005), pages 167–172.

Francis Fukuyama. *Trust: The Social Virtues and the Creation of Prosperity*. Free Press, 1996.

Aldo Gangemi and Jérôme Euzenat, editors. *Knowledge Engineering: Practice and Patterns, 16th International Conference, EKAW 2008, Acitrezza, Italy, September 29 - October 2, 2008. Proceedings*, volume 5268 of *Lecture Notes in Computer Science*. Springer, 2008.

Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening Ontologies with DOLCE. In Gómez-Pérez and Benjamins (2002), pages 166–181.

Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. Modelling Ontology Evaluation and Validation. In Sure and Domingue (2006), pages 140–154.

Aldo Gangemi, Jos Lehmann, Valentina Presutti, Malvina Nissim, and Carola Catenacci. C-ODO: an OWL Meta-model for Collaborative Ontology Design. In Noy et al. (2007).

Raúl García-Castro, Mari del Carmen Suárez-Figueroa, Mauricio Espinoza, Margherita Sini, Holger Lewen, and Eva Blomqvist. D5.6.1Experimentation with the NeOn methodologies and methods. Technical Report D5.6.1, Universidad Politécnica de Madrid, March 2008.

Yolanda Gil and Natasha Fridman Noy, editors. *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009), September 1-4, 2009, Redondo Beach, California, USA*. ACM, 2009.

Jennifer Golbeck and James A. Hendler. Accuracy of Metrics for Inferring Trust and Reputation in Semantic Web-Based Social Networks. In Motta et al. (2004), pages 116–131.

Jennifer Golbeck and Jim Hendler. Inferring Reputation on the Semantic Web. In *Proceedings of the 13th International World Wide Web Conference*, volume 316. Citeseer, 2004.

Jennifer Golbeck, Piero A. Bonatti, Wolfgang Nejdl, Daniel Olmedilla, and Marianne Winslett, editors. *Proceedings of the ISWC*04 Workshop on Trust, Security, and Reputation on the Semantic Web, Hiroshima, Japan, November 7, 2004*, volume 127 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

Jennifer Golbeck. *Computing and Applying Trust in Web-based Social Networks*. PhD thesis, University of Maryland. College Park, MD, 2005.

Jennifer Golbeck. Combining Provenance with Trust in Social Networks for Semantic Web Content Filtering. In Moreau and Foster (2006), pages 101–108.

Jennifer Golbeck. Trust on the World Wide Web: A Survey. *Foundations and Trends in Web Science*, 1(2):131–197, 2006.

Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors. *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 6-7, 2007*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

Asunción Gómez-Pérez and V. Richard Benjamins, editors. *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Proceedings*, volume 2473 of *Lecture Notes in Computer Science*. Springer, 2002.

Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering*. Advanced Information and Knowlege Processing. Springer, 2003.

Asunción Gómez-Pérez. Ontology Evaluation. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies, First Edition*, chapter 13, pages 251–274. Springer, 2004.

Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. A Logical Framework for Modularity of Ontologies. In Veloso (2007), pages 298–303.

Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Extracting Modules from Ontologies: A Logic-Based Approach. In Stuckenschmidt et al. (2009), pages 159–186.

Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Ontology Integration Using epsilon-Connections. In Stuckenschmidt et al. (2009), pages 293–320.

David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors. *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*. ACM, 2004.

Detlef Groth, Stefanie Hartmann, Georgia Panopoulou, Albert J. Poustka, and Steffen Hennig. GOblet: Annotation of anonymous sequence data with Gene Ontology and Pathway terms. *J. Integrative Bioinformatics*, 5(2), 2008.

Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

Nicola Guarino and Christopher A. Welty. Evaluating ontological decisions with OntoClean. *Commun. ACM*, 45(2):61–65, 2002.

Nicola Guarino and Christopher A. Welty. An Overview of OntoClean. In Staab and Studer (2004), pages 151–172.

Ramanathan V. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of Trust and Distrust. In Feldman et al. (2004b), pages 403–412.

Ramanathan Guha. Open Rating Systems. In *1st Workshop on Friend of a Friend, Social Networking and the Semantic Web*, 2004.

Minaxi Gupta, Paul Judge, and Mostafa H. Ammar. A Reputation System for Peer-to-Peer Networks. In Papadopoulos and Almeroth (2003), pages 144–152.

Zoltán Gyöngyi, Hector Garcia-Molina, and Jan O. Pedersen. Combating Web Spam with TrustRank. In Nascimento et al. (2004), pages 576–587.

Peter Haase, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu d'Aquin, and Enrico Motta. The NeOn Ontology Engineering Toolkit. In *WWW 2008 Developers Track*, April 2008.

M. H. Hamza, editor. *IASTED International Conference on Databases and Applications, part of the 23rd Multi-Conference on Applied Informatics, Innsbruck, Austria, February 14-16, 2005.* IASTED/ACTA Press, 2005.

Andreas Harth, Sheila Kinsella, and Stefan Decker. Using Naming Authority to Rank Data and Ontologies for Web Search. In Bernstein et al. (2009), pages 277–292.

Jens Hartmann, Raúl Palma, York Sure, María del Carmen Suárez-Figueroa, Peter Haase, Asunción Gómez-Pérez, and Rudi Studer. Ontology Metadata Vocabulary and Applications. In Meersman et al. (2005), pages 906–915.

Jens Hartmann, Raúl Palma, and Asunción Gómez-Pérez. Ontology Repositories. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks Information System, pages 551–571. Springer Berlin Heidelberg, 2009.

Jens Hartmann. ONTHOLOGY. An Ontology Metadata Repository. *Demo and Poster Proceedings of ESWC*, 2006.

Qi He and Tok Wang Ling. An ontology based approach to the integration of entity-relationship schemas. *Data Knowl. Eng.*, 58(3):299–326, 2006.

Martin Hepp, Pieter De Leenheer, Aldo de Moor, and York Sure, editors. *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications*, volume 7 of *Semantic Web And Beyond Computing for Human Experience.* Springer, 2008.

Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors. *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008.* ACM, 2008.

Aleks Jakulin and Dunja Mladenić. Ontology Grounding. In *Proceedings of 8th International multi-conference Information Society IS-2005*, pages 170–173, 2005.

Clement Jonquet, Mark A. Musen, and Nigam Shah. A System for Ontology-Based Annotation of Biomedical Data. In Bairoch et al. (2008), pages 144–152.

Audun Jøsang, Roslan Ismail, and Colin Boyd. A Survey of Trust and Reputation Systems for Online Service Provision. *Decision Support Systems*, 43(2):618–644, 2007.

Audun Jøsang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–212, 2001.

Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *WWW*, pages 640–651, 2003.

Donald E. Knuth. Big Omicron and big Omega and big Theta. *SIGACT News*, 8(2):18–24, 1976.

Charles W. Krueger. Software Reuse. *ACM Computing Surveys*, 24(2):131–183, 1992.

Alexander Kubias, Marko Babic, Holger Lewen, Martin Dzbor, Klaas Dellschaft, and Jose Manuel Gómez-Pérez. D4.3.1 Review of Trust Models as a Criterion for Ontology Customization. Technical Report D4.3.1, University of Koblenz-Landau, October 2007.

Chang-Shing Lee and Mei-Hui Wang. Ontology-based intelligent healthcare agent and its application to respiratory waveform recognition. *Expert Syst. Appl.*, 33(3):606–619, 2007.

Douglas B. Lenat. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

Jure Leskovec, Daniel P. Huttenlocher, and Jon M. Kleinberg. Predicting Positive and Negative Links in Online Social Networks. In Rappa et al. (2010), pages 641–650.

Ralph Levien. *Attack Resistant Trust Metrics.* PhD thesis, UC Berkeley, USA, 2003.

Holger Lewen and Mathieu d'Aquin. Extending Open Rating Systems for Ontology Ranking and Reuse. In Cimiano and Pinto (2010), pages 441–450.

Holger Lewen, Kaustubh Supekar, Natalya F. Noy, and Mark A. Musen. Topic-Specific Trust and Open Rating Systems: An Approach for Ontology Evaluation. In *Proc. of the 4th Int. Workshop on Evaluation of Ontologies for the Web (EON), 15th Int. World Wide Web Conference)*, Edinburgh, UK, MAY 2006.

Holger Lewen, Mathieu d'Aquin, Jérôme Euzenat, Chan Le Duc, and Raúl Palma. D1.4.3 Cupboard—Supporting Ontology Reuse by Combining a Semantic Web Gateway, Ontology Registry and Open Ratings Systems. Technical Report D1.4.3, Universität Karlsruhe (TH), February 2009.

Holger Lewen, Mathieu d'Aquin, and Salman Elahi. D1.4.6 Cupboard–Supporting Ontology Reuse by Combining a Semantic Web Gateway, Ontology Registry and Open Ratings Systems – Improved and Final Version. Technical Report D1.4.6, Universität Karlsruhe (TH), February 2010.

Holger Lewen. Introducing Topic-Specific Trust in Open Rating Systems. Diplomarbeit, Institute AIFB, Universität Karlsruhe (TH), December 2005.

Holger Lewen. Facilitating Ontology Reuse with a Topic-Specific Trust Open Rating System. Technical report, Universität Karlsruhe (TH), JUN 2009. http://people.aifb.kit.edu/hle/paper/TR3.pdf.

Holger Lewen. Implementation and Performance Evaluation of the Topic-Specific Trust Open Rating System. Technical report, Universität Karlsruhe (TH), JUN 2009. http://people.aifb.kit.edu/hle/paper/TR1.pdf.

Holger Lewen. Optimization and New Performance Evaluation of the Topic-Specific Trust Open Rating System. Technical report, Institut AIFB, KIT, DEC 2009. http://people.aifb.kit.edu/hle/paper/TR4.pdf.

Holger Lewen. Simulation-based Evaluation of the Topic-Specific Trust Open Rating System. Technical report, Universität Karlsruhe (TH), JUN 2009. http://people.aifb.kit.edu/hle/paper/TR2.pdf.

Chu Yee Liau, Xuan Zhou, Stéphane Bressan, and Kian-Lee Tan. Efficient Distributed Reputation Scheme for Peer-to-Peer Systems. In Chung et al. (2003), pages 54–63.

Cristiano Longo and Lorenzo Sciuto. A Lightweight Ontology for Rating Assessments. In Semeraro et al. (2008).

Ignac Lovrek, Robert J. Howlett, and Lakhmi C. Jain, editors. *Knowledge-Based Intelligent Information and Engineering Systems, 12th International Conference, KES 2008, Zagreb, Croatia, September 3-5, 2008, Proceedings, Part I*, volume 5177 of *Lecture Notes in Computer Science*. Springer, 2008.

Alexander Maedche and Steffen Staab. Measuring Similarity between Ontologies. In Gómez-Pérez and Benjamins (2002), pages 251–263.

Massimo Marchiori, Jeff Z. Pan, and Christian de Sainte Marie, editors. *Web Reasoning and Rule Systems, First International Conference, RR 2007, Innsbruck , Austria, June 7-8, 2007, Proceedings*, volume 4524 of *Lecture Notes in Computer Science*. Springer, 2007.

Stephen Paul Marsh. *Formalising Trust as a Computational Concept.* PhD thesis, University of Stirling, 1994.

Paolo Massa and Paolo Avesani. Controversial Users Demand Local Trust Metrics: An Experimental Study on Epinions.com Community. In Veloso and Kambhampati (2005), pages 121–126.

Paolo Massa and Paolo Avesani. Trust Metrics in Recommender Systems. In *Computing with Social Trust*, HCI Series, pages 259–285. Springer London, 2009.

Paolo Massa and Conor Hayes. Page-reRank: Using Trusted Links to Re-Rank Authority. In Skowron et al. (2005), pages 614–617.

Ryusuke Masuoka, Yannis Labrou, Bijan Parsia, and Evren Sirin. Ontology-Enabled Pervasive Computing Applications. *IEEE Intelligent Systems*, 18(5):68–72, 2003.

Deborah L. McGuinness and Frank van Harmelen, editors. *OWL Web Ontology Language Overview*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors. *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, volume 3298 of *Lecture Notes in Computer Science*. Springer, 2004.

Doug M. McIlroy. Mass produced software components. In John Buxton, Peter Naur, and Brian Randell, editors, *Software Engineering Concepts and Techniques (Proc. 1968 NATO Conf. on Software Engineering)*, pages 88–98. Van Nostrand Reinhold, 1976.

Robert Meersman, Zahir Tari, Pilar Herrero, Gonzalo Méndez, Lawrence Cavedon, David Martin, Annika Hinze, George Buchanan, María S. Pérez, Víctor Robles, Jan Humble, Antonia Albani, Jan L. G. Dietz, Hervé Panetto, Monica Scannapieco, Terry A. Halpin, Peter Spyns, Johannes Maria Zaha, Esteban Zimányi, Emmanuel Stefanakis, Tharam S. Dillon, Ling Feng, Mustafa Jarrar, Jos Lehmann, Aldo de Moor, Erik Duval, and Lora Aroyo, editors. *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops, OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, GADA, MIOS+INTEROP, ORM, PhDS, SeBGIS, SWWS, and WOSE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings*, volume 3762 of *Lecture Notes in Computer Science*. Springer, 2005.

Stanley Milgram. The Small World Problem. *Psychology today*, 2(1):60–67, 1967.

Barbara A. Misztal. *Trust in Modern Societies: The Search for the Bases of Social Order*. Polity Press, 1996.

Luc Moreau and Ian T. Foster, editors. *Provenance and Annotation of Data, International Provenance and Annotation Workshop, IPAW 2006, Chicago, IL, USA, May 3-5, 2006, Revised Selected Papers*, volume 4145 of *Lecture Notes in Computer Science*. Springer, 2006.

Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors. *Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004, Whittlebury Hall, UK, October 5-8, 2004, Proceedings*, volume 3257 of *Lecture Notes in Computer Science*. Springer, 2004.

Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors. *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*. Morgan Kaufmann, 2004.

Ngoc Thanh Nguyen and Lakhmi C. Jain, editors. *Intelligent Agents in the Evolution of Web and Applications*, volume 167 of *Studies in Computational Intelligence*. Springer, 2009.

Ian Niles and Adam Pease. Towards a Standard Upper Ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9, 2001.

NIST. Secure Hash Standard, Federal Information Processing Standards Publication 180-2. *US Department of Commerce, National Institute of Standards and Technology (NIST)*, 2002.

Natalya F. Noy, Ramanathan Guha, and Mark A. Musen. User ratings of ontologies: Who will rate the raters. In *Proceedings of the AAAI 2005 Spring Symposium on Knowledge Collection from Volunteer Contributors*, 2005.

Natalya Fridman Noy, Harith Alani, Gerd Stumme, Peter Mika, York Sure, and Denny Vrandecic, editors. *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007) at the 16th International World Wide Web Conference (WWW2007) Banff, Canada, May 8, 2007*, volume 273 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

Natalya Fridman Noy, Nicholas Griffith, and Mark A. Musen. Collecting Community-Based Mappings in an Ontology Repository. In Sheth et al. (2008), pages 371–386.

Natalya F. Noy, Nigam H. Shah, Patricia L. Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L. Rubin, Margaret-Anne A. Storey, Christopher G. Chute, and Mark A. Musen. BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic acids research*, 37(Web Server issue):W170–173, July 2009.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University, CA, USA, 1998.

Raúl Palma, Peter Haase, and Asunción Gómez-Pérez. Oyster – Sharing and Re-using Ontologies in a Peer-to-Peer Community. In Carr et al. (2006), pages 1009–1010.

Raúl Palma, Peter Haase, Óscar Corcho, Asunción Gómez-Pérez, and Qiu Ji. An Editorial Workflow Approach For Collaborative Ontology Development. In *Proc. of the 3rd Asian Semantic Web Conference (ASWC 2008)*, volume 5367 of *LNCS*, pages 227–241. Springer, 2008.

Christos Papadopoulos and Kevin C. Almeroth, editors. *Network and Operating System Support for Digital Audio and Video, 13th International Workshop, NOSSDAV 2003, Monterey, CA, USA, June 1-3, 2003, Proceedings.* ACM, 2003.

Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL Ontologies. In Ellis and Hagino (2005a), pages 633–640.

Chintan Patel, Kaustubh Supekar, Yugyung Lee, and E. K. Park. OntoKhoj: A Semantic Web Portal for Ontology Searching, Ranking and Classification. In Chiang et al. (2003), pages 58–61.

Jyotishman Pathak, Thomas M. Johnson, and Christopher G. Chute. Survey of Modular Ontology Techniques and their Applications in the Biomedical Domain. *Integr. Comput.-Aided Eng.*, 16(3):225–242, 2009.

Silvio Peroni, Enrico Motta, and Mathieu d'Aquin. Identifying Key Concepts in an Ontology, through the Integration of Cognitive Principles with Statistical and Topological Measures. In Domingue and Anutariya (2008), pages 242–256.

Robert Porzel and Rainer Malaka. A Task-based Approach for Ontology Evaluation. In *ECAI Workshop on Ontology Learning and Population, Valencia, Spain.* Citeseer, 2004.

Roger S. Pressman and Darrel Ince. *Software Engineering: A Practitioner's Approach.* McGraw-Hill New York, 1987.

Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors. *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010.* ACM, 2010.

Paul Resnick and Hal R. Varian. Recommender Systems - Introduction to the Special Section. *Commun. ACM*, 40(3):56–58, 1997.

Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust Management for the Semantic Web. In Fensel et al. (2003), pages 351–368.

Jordi Sabater and Carles Sierra. Reputation and Social Network Analysis in Multi-Agent Systems. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 475–482. ACM, 2002.

Jordi Sabater and Carles Sierra. Social ReGreT, a reputation model based on social relations. *SIGecom Exchanges*, 3(1):44–56, 2002.

Marta Sabou, Sofia Angeletou, Mathieu d'Aquin, Jesus Barrasa, Klaas Dellschaft, Aldo Gangemi, Jos Lehmann, Holger Lewen, Diana Maynard, Dunja Mladenic, Malvina Nissim, Wim Peters, Valentina Presutti, and Boris Villazon. D2.2.1 Methods for Selection and Integration of Reusable Components from Formal or Informal User Specifications. Technical Report D2.2.1, Knowledge Media Institute, the Open University, May 2007.

Marta Sabou, Guadalupe Aguado de Cea, Mathieu d'Aquin, Enrico Daga, Holger Lewen, Elena Montiel, Valentina Presutti, and Mari del Carmen Suárez-Figueroa. D2.2.3 Methods and Tools for the Evaluation and Selection of Knowledge Components. Technical Report D2.2.3, Knowledge Media Institute, the Open University, February 2009.

Norman M. Sadeh, Mary Jo Dively, Robert J. Kauffman, Yannis Labrou, Onn Shehory, Rahul Telang, and Lorrie Faith Cranor, editors. *Proceedings of the 5th International Conference on Electronic Commerce, ICEC 2003, Pittsburgh, Pennsylvania, USA, September 30 - October 03, 2003*, volume 50 of *ACM International Conference Proceeding Series*. ACM, 2003.

Pierangela Samarati and Sabrina De Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms. *Foundations of Security Analysis and Design*, pages 137–196, 2001.

Abraham Sebastian, Natalya Fridman Noy, Tania Tudorache, and Mark A. Musen. A Generic Ontology for Collaborative Ontology-Development Workflows. In Gangemi and Euzenat (2008), pages 318–328.

Giovanni Semeraro, Eugenio Di Sciascio, Christian Morbidoni, and Heiko Stoermer, editors. *Proceedings of the 4th Italian Semantic Web Workshop, Dipartimento di Informatica - Universita' degli Studi di Bari - Italy, 18-20 December, 2007*, volume 314 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors. *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*. Springer, 2008.

Pavel Shvaiko, Jérôme Euzenat, Fausto Giunchiglia, Heiner Stuckenschmidt, Natalya Fridman Noy, and Arnon Rosenthal, editors. *Proceedings of the 4th International Workshop on Ontology Matching (OM-2009) collocated with the 8th International Semantic Web Conference (ISWC-2009) Chantilly, USA, October 25, 2009*, volume 551 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

Elena Paslaru Bontas Simperl, Christoph Tempich, and York Sure. ONTOCOM: A Cost Estimation Model for Ontology Engineering. In Cruz et al. (2006), pages 625–639.

Elena Paslaru Bontas Simperl, Igor O. Popov, and Tobias Bürger. ONTOCOM Revisited: Towards Accurate Cost Predictions for Ontology Development Projects. In Aroyo et al. (2009), pages 248–262.

Elena Paslaru Bontas Simperl. Reusing ontologies on the Semantic Web: A feasibility study. *Data Knowl. Eng.*, 68(10):905–925, 2009.

Andrzej Skowron, Rakesh Agrawal, Michael Luck, Takahira Yamaguchi, Pierre Morizet-Mahoudeaux, Jiming Liu, and Ning Zhong, editors. *2005 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2005), 19-22 September 2005, Compiegne, France*. IEEE Computer Society, 2005.

Barry Smith and Pierre Grenon. The Cornucopia of Formal-Ontological Relations. *Dialectica*, 58(3):279–296, 2004.

Barry Smith, Waclaw Kusnierczyk, Daniel Schober, and Werner Ceusters. Towards a Reference Terminology for Ontology Research and Development in the Biomedical Domain. In *Proceedings of KR-MED*, volume 2006, pages 57–65. Citeseer, 2006.

Barry Smith. Ontology (Science). In Eschenbach and Grüninger (2008), pages 21–35.

Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

Steffen Staab, Rudi Studer, Hans-Peter Schnurr, and York Sure. Knowledge Processes and Ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

Katherine J. Stewart and Yali Zhang. Effects of Hypertext Links on Trust Transfer. In Sadeh et al. (2003), pages 235–239.

Katherine J. Stewart. Transference as a means of building trust in World Wide Web sites. In *ICIS*, pages 459–464, 1999.

Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*. Springer, 2009.

Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge Engineering: Principles and Methods. *Data Knowl. Eng.*, 25(1-2):161–197, 1998.

Kaustubh Supekar, Daniel Rubin, Natalya F. Noy, and Mark A. Musen. Knowledge Zone: A Public Repository of Peer-Reviewed Biomedical Ontologies. *Studies in health technology and informatics*, 129(1):812, 2007.

York Sure and John Domingue, editors. *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings*, volume 4011 of *Lecture Notes in Computer Science*. Springer, 2006.

Zhihong Tan, Weiling Liu, Libing Liu, and Zeqing Yang. The Application of Ontology Model in Intelligent Tutoring System. In *International Conference on Computer Science and Software Engineering, CSSE 2008, Volume 5: E-learning and Knowledge Management / Socially Informed and Instructinal Design / Learning Systems Platforms and Architectures / Modeling and Representation / Other Applications , December 12-14, 2008, Wuhan, China*, pages 1176–1179. IEEE Computer Society, 2008.

Adolfo Lozano Tello and Asunción Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *J. Database Manag.*, 15(2):1–18, 2004.

Christoph Tempich, Elena Paslaru Bontas Simperl, Markus Luczak, Rudi Studer, and Helena Sofia Pinto. Argumentation-Based Ontology Engineering. *IEEE Intelligent Systems*, 22(6):52–59, 2007.

Edward Thomas, Jeff Z. Pan, and Derek H. Sleeman. ONTOSEARCH2: Searching Ontologies Semantically. In Golbreich et al. (2007).

Andreas Tolk and Lakhmi C. Jain, editors. *Complex Systems in Knowledge-based Environments: Theory, Models and Applications*, volume 168 of *Studies in Computational Intelligence*. Springer, 2009.

Thanh Tran, Philipp Cimiano, Sebastian Rudolph, and Rudi Studer. Ontology-Based Interpretation of Keywords for Semantic Search. In Aberer et al. (2007), pages 523–536.

Thanh Tran, Peter Haase, Holger Lewen, Óscar Muñoz-García, Asunción Gómez-Pérez, and Rudi Studer. Lifecycle-Support in Architectures for Ontology-Based Information Systems. In Aberer et al. (2007), pages 508–522.

Thanh Tran, Holger Lewen, and Peter Haase. On the Role and Application of Ontologies in Information Systems. In *2007 IEEE International Conference on Research, Innovation and Vision for the Future in Computing & Communication Technologies, RIVF 2007, Hanoi, Vietnam, 5-9 March 2007*, pages 14–21. IEEE, 2007.

Tania Tudorache, Natalya Fridman Noy, Samson W. Tu, and Mark A. Musen. Supporting Collaborative Ontology Development in Protégé. In Sheth et al. (2008), pages 17–32.

Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice.com: Weaving the Open Linked Data. In *Proc. of the 6th Int. Semantic Web Conference, 2nd Asian Semantic Web Conference*, volume 4825 of *LNCS*, pages 552–565. Springer, 2007.

Amos Tversky and Daniel Kahneman. Judgment under Uncertainty: Heuristics and Biases. *Science*, 185(4157):1124–1131, 1974.

Mike Uschold and Michael Gruninger. Ontologies: Principles, methods and applications. *The Knowledge Engineering Review*, 11(02):93–136, 1996.

Mike Uschold, Mike Healy, Keith Williamson, Peter Clark, and Steven Woods. Ontology Reuse and Application. In *Proc. of the International Conference on Formal Ontology and Information Systems FOIS'98*, pages 179–192, 1998.

Manuela M. Veloso and Subbarao Kambhampati, editors. *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. AAAI Press / The MIT Press, 2005.

Manuela M. Veloso, editor. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007.

Kim Viljanen, Jouni Tuominen, and Eero Hyvönen. Ontology Libraries for Production Use: The Finnish Ontology Library Service ONKI. In Aroyo et al. (2009), pages 781–795.

Luis von Ahn. Games with a Purpose. *IEEE Computer*, 39(6):92–94, 2006.

Zdenko Vrandecic. *Ontology evaluation.* PhD thesis, Fakultät für Wirtschaftswissenschaften, Universität Karlsruhe (TH), 2010.

W3C OWL Working Group, editor. *OWL 2 Web Ontology Language Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

Yimin Wang, Jie Bao, Peter Haase, and Guilin Qi. Evaluating Formalisms for Modular Ontologies in Distributed Information Systems. In Marchiori et al. (2007), pages 178–193.

Baoning Wu and Brian D. Davison. Identifying Link Farm Spam Pages. In Ellis and Hagino (2005b), pages 820–829.

Frank Yates. Contingency Tables Involving Small Numbers and the χ 2 Test. *Supplement to the Journal of the Royal Statistical Society*, pages 217–235, 1934.

Giorgos Zacharia and Pattie Maes. Trust Management through Reputation Mechanisms. *Applied Artificial Intelligence*, 14(9):881–907, 2000.

Paul J. Zak, Robert Kurzban, and William T. Matzner. The Neurobiology of Trust. *Annals of the New York Academy of Sciences*, 1032(Biobehavioral Stress Response: Protective and Damaging Effects):224–227, 2004.

Qing Zhang and Ting Yu. A Classification Scheme for Trust Functions in Reputation-Based Trust Management. In Golbeck et al. (2004).

Cai-Nicolas Ziegler and Georg Lausen. Propagation Models for Trust and Distrust in Social Networks. *Information Systems Frontiers*, 7(4-5):337–358, 2005.

Philip R. Zimmermann. The ¿fficial PGP User's Guide, 1995.

Full Table of Contents