Improving Centruflow Using Semantic Web Technologies

A thesis presented in partial fulfillment of the requirements for
the degree of Master of Science in Computer Science at
Massey University, Palmerston North, New Zealand.

Jonathan Andrew Giles
2007

Abstract

Centruflow is an application that can be used to visualise structured data. It does this by drawing graphs, allowing for users to explore information relationships that may not be visible or easily understood otherwise. This helps users to gain a better understanding of their organisation and to communicate more effectively. In earlier versions of Centruflow, it was difficult to develop new functionality as it was built using a relatively unsupported and proprietary visualisation toolkit. In addition, there were major issues surrounding information currency and trust. Something had to be done, and this was a sub-project of this thesis.

The main purpose of this thesis however was to research and develop a set of mathematical algorithms to infer implicit relationships in Centruflow data sources. Once these implicit relationships were found, we could make them explicit by showing them within Centruflow. To enable this, relationships were to be calculated based on providing users with the ability to 'tag' resources with metadata. We believed that by using this tagging metadata, Centruflow could offer users far more insight into their own data.

Implementing this was not a straight-forward task, as it required a considerable amount of research and development to be undertaken to understand and appreciate technologies that could help us in our goal. Our focus was primarily on technologies and approaches common in the semantic web and 'Web 2.0' areas. By pursuing semantic web technologies, we ensured that Centruflow would be considerably more standards-compliant than it was previously. At the conclusion of our development period, Centruflow had been rather substantially 'retrofitted', with all proprietary technologies replaced with equivalent semantic web technologies. The result of this is that Centruflow is now positioned on the forefront of the semantic web wave, allowing for far more comprehensive and rapid visualisation of a far larger set of readily-available data than what was possible previously.

Having implemented all necessary functionality, we validated our approach and were pleased to find that our improvements led to a considerably more intelligent and useful Centruflow application than was previously available. This functionality is now available as part of 'Centruflow 3.0', which will be publicly released in March 2008. Finally, we conclude this thesis with a discussion on the future work that should be undertaken to improve on the current release.

Acknowledgements

"I would have written a shorter letter, but I did not have the time."
- Blaise Pascal

It is my pleasure to thank the people and institutions that made this thesis possible.

Firstly I want to thank my wife Julia, who has supported me throughout the year. Undertaking this thesis required considerable time, and I thank her for allowing me to work as hard as I needed to to complete this thesis.

Thanks to my parents Ian and Sue who have always supported me in whatever endeavour I set out on, and for buying me that Apple IIe back in the late 80's for $5000. It introduced me to the wonderful world of computing, and got me my first (unpaid) job teaching my friends and teachers at my local kindergarten (at the age of four).

I wish to acknowledge my supervisor Dr. Jens Dietrich from Massey University for his time, support, and ideas during this project. Without Jens, this project would not have been as successful as it was, particularly in terms of the formulation of the necessary mathematical functions. I am by no means a mathematician.

The Foundation for Research Science & Technology financially supported this project through the provision of a Technology in Industry Fellowship (TIF). Additionally, I appreciate the support of Massey University, through their provision of both the Massey Scholar scholarship and the Massey Masterate scholarship.

This project could not have been implemented without the support of a large number of open source projects, of which there are too many to mention. The key projects I want to thank include:

- The Jena semantic web framework by Hewlett-Packard Laboratories Bristol
- Chris Bizer for his D2R Server, and D2RQ mapping language.

Finally, thank you for taking the time to read this thesis. I appreciate any thoughts you may have on this research, and invite you to contact me via email at Jo@JoGiles.co.nz.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 The Big Picture

> "The dream behind the Web is of a common information space in which we communicate by sharing information. Its universality is essential: the fact that a hypertext link can point to anything, be it personal, local or global, be it draft or highly polished. There was a second part of the dream, too, dependent on the Web being so generally used that it became a realistic mirror (or in fact the primary embodiment) of the ways in which we work and play and socialize. That was that once the state of our interactions was on line, we could then use computers to help us analyse it, make sense of what we are doing, where we individually fit in, and how we can better work together."
>
> - Tim Berners-Lee [1]

This thesis explores the research and development of a new version of Centruflow [2], an application used to visualise data sources inside large organisations. Like Tim Berners-Lee in the quote above, Centruflow has been and continues to be a dream for us. The first two versions of Centruflow very much helped turn our dream into a reality and create a sturdy base framework, but it is the topic of this masters thesis that, it is hoped, will turn Centruflow into a tool that large companies will need to have. The goal of Centruflow 3.0 (and in part this thesis) is to enable the visualisation of more varied data sources, using standards and recommendations proposed by bodies such as the World Wide Web Consortium [3] (or W3C as it is more commonly known). On top of this, our goal is to enable increased socialisation and communication within Centruflow through the use of 'web 2.0' concepts such as tagging.

As with any research thesis, the overall goal is to publish an account of the creation of new knowledge and intellectual property. This thesis is no different, but its focus is split

between the creation of a set of algorithms to calculate trust, and the development of Centruflow 3.0, which uses these algorithms in a corporate sense. The reason for the second area of focus is due to the fact that this thesis is funded by both industry and the New Zealand Government, so a large emphasis has been placed on achieving results that lead to increased business value.

How do we plan to extend knowledge and understanding? For starters, Centruflow will be a proof that viable software applications can be built atop the semantic web vision. This has been a recurring question online, where people question the viability of the semantic web [4], and whether it is merely an academic pursuit in the same vein as the topic of artificial intelligence once was (and still largely is). In addition to this, Centruflow will make extensive use of very early[1] open source software libraries, and will be well-placed to test and develop these in conjunction with the open source community. From a business perspective, it is hoped that Centruflow will be able to visualise far more sources of data, and in far better ways, helping organisations to be more informed and able to make better decisions using the pictures generated by Centruflow.

What follows in the remainder of this chapter is an overview into the areas that this research was involved in (this is expanded in much more detail in chapter 2). Following this, section 1.4 outlines the goals and the scope of this thesis. We then discuss the business requirements of the software that were developed as part of this thesis. At the end of this chapter (section 1.6) is an outline for the remainder of this thesis.

1.2 Research Overview

1.2.1 The Internet

The Internet, as it stands today, is a huge mass of data (with educated guesses suggesting that there are a 100 trillion words on the Internet [5]). A subset of the Internet, the World Wide Web (WWW), is a human-browsable series of hyperlinked pages, and is the primary medium used by most users of the Internet. Up until recently, there was very little standardisation of the web, except for the use of HTML for presenting information to the user. It should be noted however that standardisation is a loaded word, as it merely means each vendor has adopted their own variation of each others standards [6].

There is presently very little scope for any more advanced use of the data on the Internet, for example for intelligent agents to use this information to assist human users. The most advanced technologies that support this kind of dream are 'screen scrapers' (which depend on the HTML output remaining largely constant) and, more recently, the proliferation

[1] Early in the sense that many of these software components are built to enable semantic web functionality, and are in alpha and beta stages of release.

of web services. This makes any ambition to create 'software agents', that is, intelligent software applications that may help users, far more difficult than necessary. Additionally, the integration and reuse of data from multiple sources is near-impossible without custom software development occurring for each unique scenario.

Despite this, the Internet has been an undeniably huge success for human users to connect and communicate in areas where Internet connectivity is available. This is evidenced in the World Internet Usage and Population Statistics website [7], which states that of the approximately 6,574,666,417 people in the world, 1,244,449,601 people have access to the Internet. This represents 18.9% of the worlds population. Notable by their lack of Internet penetration are Asia (at 12.4%) and Africa (at 4.7%). This represents a far larger social problem that must be resolved, but falls far outside the scope of this thesis.

Whilst the Internet has succeeded in connecting 18.9% of the world to each other, it has largely failed in supporting computer software agents to in turn support their users. This is because the Internet is simply too difficult to be understood by computers. One of the key goals of the semantic web is to improve on this situation by providing more metadata about information, allowing for software agents to become more informed and more connected. One of the goals of this thesis was to learn more about the semantic web and how it intends to improve upon this situation.

1.2.1.1 Standardisation and Web Services

Standardisation in the IT world is often fraught with difficulties. The range and number of standards is overwhelming, which leads to a situation where no one is really any better off than they were prior to the standards being developed. As the saying goes, the good thing about standards is that there are plenty to choose from - and if you don't like any, there is always next year's lot!

Early efforts to standardise on a means to allow software to remotely communicate has in particular been rather fractured, and in many circumstances, over-engineered. At any one time there have been multiple frameworks designed to support this requirement, but no one framework has gained the critical mass necessary to accelerate its rate of adoption to become a de facto or industry standard approach. This has lead to the situation where application developers are forced to carefully consider their design decisions, such that their software can interact with other important software. It has also lead to an increased number of enterprise messaging systems, whose job it is to route messages between applications safely and securely.

The most popular approach has always been the concept of a 'Remote Procedure Call' (RPC) in software, whereby an application may communicate with another through some form of previously agreed upon protocol. There have been attempts to standardise such

protocols, with prominent examples being the 'Common Object Request Broker Architecture' (CORBA)[**?**] and Java's Remote Method Invocation (RMI)[8]. These technologies have struggled to gain market acceptance, and the primary reason behind this is their complexity and limitations [9].

More recently the SOAP standard [10] has taken the headlines. SOAP uses the common and highly supported XML syntax, but still has a rather 'enterprise' feel to it, requiring the developer to jump through a number of hoops to deploy such services on the Internet [11].

A simpler alternative to SOAP called Representational State Transfer (REST)[12], whilst not a standard, has also captured a lot of attention from developers. REST simply uses the basic Internet infrastructure (the HTTP protocol) with no complex messaging layer atop it (as is the case with SOAP). The most important HTTP methods are PUT, GET, POST and DELETE, which are often compared to the CREATE, READ, UPDATE and DELETE (CRUD) methods used by databases. An important point of REST is that it is stateless, meaning that neither client or server needs to remember the interactions that have occurred. Using this definition, the web itself is mostly RESTful, with things like cookies and sessions introducing some statefulness.

1.2.2 Web 2.0 And Social Networking

The term 'Web 2.0', coined by Tim O'Reilly [13], is a confused term, insofar that it has no real definition. We deem the term to mean the proliferation of rapidly developed websites that make use of technologies such as asynchronous Javascript and XML (or 'AJAX', as it is commonly known - not to be mistaken for the cleaning liquid however). With Web 2.0 websites, interactions started becoming more social, as websites could be more fluid and user friendly, and communities could be more easily built. This is in part thanks to AJAX, but also due to the influx of big business (and thus money and venture capital), and at the same time, the increasing importance of web *design*, not just web development.

'Social networking' is the term applied to the recent development of websites that allow for its users to easily communicate, post pictures, form groups around shared ideals and promote themselves or their ideals. There are a number of well-known social network sites, each catering to a particular demographic, with examples including MySpace, Facebook, Bebo, and LinkedIn [14, 15, 16, 17]. In addition, the concept and subsequent adoption of web diaries (or 'blogs') has exploded. To sum up what social networking is, one could say that it is the enabling of people on the Internet to more easily communicate with those people who share common interests or friends.

A common attribute of social networking is that it is based around user profiles. This leads to users building up a form of 'public trust' based on, for example, the amount of positive

feedback they have received from other users (versus the amount of negative feedback). This is the common scenario used by most online auction sites, such as TradeMe [18] and eBay [19]. The important thing to note is that users are directly reviewing other users, which can lead to malicious abuse of the system. We consider this a weakness, which is something that we endeavoured to work around in this research.

1.2.3 The Semantic Web

> "I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize."
>
> - Tim Berners-Lee [20]

The semantic web is a vision backed by the World Wide Web Consortium (W3C)[3]. It essentially tries to pick up where the Internet left off, in relation to improving the ability to make information more machine-understandable. It has struggled to gain traction, as people outside academia have objected to the scale of the work involved, but it is gaining momentum in the academic realm. It is hoped that through improved tools, academia will eventually allow for the automatic generation and extraction of much of the metadata required by the semantic web. Projects such as GRDDL [21, 22] and RDFa [23] are evolving to help fill this hole.

The semantic web vision has existed for quite some time [24], and has been driven by the World Wide Web Consortium, who have been diligently developing the necessary standards to support this. The W3C is led by Sir Tim Berners-Lee, who is the original author of the HTTP protocol, which is used to link web pages together.

We cover the semantic web in much more detail in section 2.1, where we outline the research we undertook on a number of the key technologies.

1.2.4 Visualisation Technology

Whilst not within the scope of this thesis, we pride ourselves on providing the latest visualisation technology to our customers. As we identified that a replacement of the visualisation engine would be necessary to achieve our goals for this thesis, we had to identify how a new visualisation engine would integrate with the rest of the work that was within the scope of this masters.

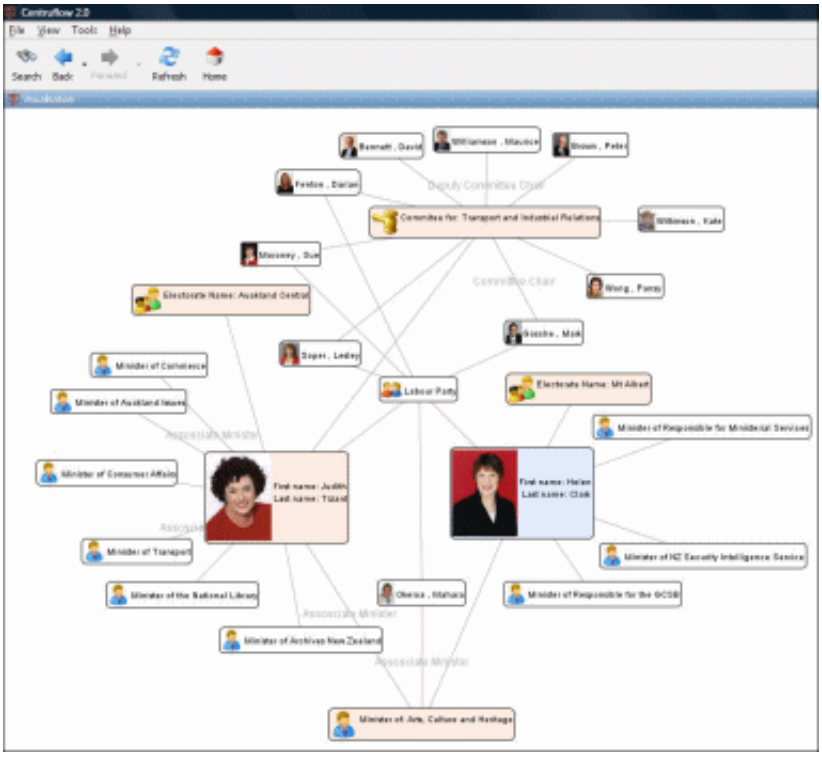


Figure 1.1: A screenshot of Centruflow 3.0's main visualisation perspective.

In summary, the chosen technology to replace our current visualisation technology is called prefuse [25], with the job of integration falling to Graham Jenson, a honours student within the College of Sciences at Massey University, New Zealand. The split between Graham's work and the work done as part of this thesis is essentially that we were responsible for getting the data all the way from the relevant data sources into the replacement visualisation engine, therefore requiring very little of the visualisation. To aid in understanding what Centruflow is, figure 1.1 is a screenshot of Centruflow 3.0 in action.

1.3 Centruflow

Centruflow [2] is a software application that is tasked with providing clarity to 'information overflow' inside companies. It does this by providing a visualisation that draws all information inside a company as a single graph, that can be navigated around by the user. In this regard, Centruflow allows for users to find information that may have been hidden inside their organisation. In addition to this, Centruflow allows for users to edit the information shown to them, ensuring that data is always up to date.

Centruflow works by connecting to an organisations databases directly, and visualises the information contained within these databases as a graph that may be navigated graphically. Centruflow was born from the experience of people in the IT industry, as there was a growing realisation that static images (such as those drawn in Microsoft Visio) and

spreadsheets were not a viable long-term communication medium. This problem is only exacerbated by the frequent changes that are necessary to this kind of document, and the ease in which data can be duplicated, leading to a loss in data currency, and thus, trust in the data. As Centruflow technology advanced through its first two versions, it became clear that there were some inherent weaknesses in the architecture, outlined in the remainder of this section.

1.3.1 Templates

To ensure that Centruflow provides users with a rich user experience, it is necessary that the client understand the data that it is visualising as much as possible. This is the role of 'templates' within Centruflow. A template is a collection of XML files that provide the client with all necessary configuration data relevant to the current graph. Template files include files that describe the metadata of each node and edge, the kinds of elements within a form to present to the user when they edit an entity, how to cluster and sort data, among other template files. A graph without a template lacks considerable functionality and presents only a minimal user interface.

1.3.2 Centruflow Release Timeline

1.3.2.1 Centruflow 1.0

Centruflow was conceived by Abstract Engineering Ltd [26] in mid-2004, with development of Centruflow starting in late 2004. Centruflow is structured using a client/server architecture. Development of Centruflow 1.0 largely led to a 'software island' - it depended on proprietary technologies and database schemas. In particular, the entire server-side component, and the visualisation software within the client, both belonged to a third-party software company called 'Thinkmap'. This meant that the cost of deploying a Centruflow-based solution was larger than it should have been, for two key reasons:

1. For every sale, a considerable proportion of our earnings were being paid to the third-party to allow us to license their visualisation and server software.

2. For every customer, there was need for considerable consultancy time to transfer the companies data to appropriate data structures. Many companies would not even consider the software until the consultancy work had been completed. This led to a number of occurrences where considerable time was spent on customers who did not then proceed to purchase licenses to use Centruflow.

In addition to these problems, the approach we took with Centruflow 1.0 was too prohibitive to enable the kind of software we wanted - we were too forced into a particular development methodology. For the Centruflow 1.x series, the primary weakness was that Centruflow was being developed within the proprietary Thinkmap SDK, which meant that should the 1.x series continue to exist and be developed, Centruflow would forever be dependent on Thinkmap software. Given the risks that this introduced to the business (not to mention the costs), it was decided that a major redevelopment effort would be necessary.

1.3.2.2 Centruflow 2.0

This redevelopment process took place throughout 2006 as an honours project at Massey University undertaken by Jonathan Giles [27]. It almost entirely rewrote Centruflow to be far more flexible (it become plugin-based), and far less dependent on any one software vendor. This resulted in the Centruflow 2.x series of releases. Centruflow 2.0 still made use of the Thinkmap visualisation technology however, as this provided the easiest means of visualising enterprise data sources. In addition, the Centruflow 2.x work did not attempt to resolve any issues with the server-side components, so we were still largely dependent on Thinkmap.

1.3.2.3 Centruflow 3.0

Once Centruflow 2.0 was completed, we were still faced with the same issues as previously mentioned: we were constrained by third party visualisation software, and the need to duplicate information. This situation simply could not last in the long-term, as many customers would balk at having to duplicate their data unnecessarily. This led to an inherent lack of trustability in the very system designed to improve trust and communications. Fortunately, at this time Centruflow was only lightly deployed to the public, and stopgap solutions (including editing Microsoft Excel spreadsheets which would then do a bulk dump into an empty database) were developed. With businesses beginning to take Centruflow more seriously, a better solution was needed, with this need prompting the beginning of the Centruflow 3.x series of releases.

Because of the improved software abstractions in Centruflow 2.0, it become far more feasible to consider improving Centruflow by replacing the visualisation software. These opportunities were carefully reviewed, and this was the driving force behind Centruflow 3.0, as well as a small portion of this thesis.

1.3.3 Configuring Centruflow

Setting up Centruflow to visualise a new data source is a two-step process, which is outlined below. It is hoped that by outlining this, the reader gets a better understanding of how

Centruflow operates.

The first step prior to being able to use Centruflow is to correctly configure a server-side component to understand the architecture of the databases, and the relationships inherent within them. In Centruflow 1.x and 2.x releases the server-side component was the 'Thinkmap Server', but in Centruflow 3.0 this was the 'Centruflow Server', which is outlined in more detail later in this thesis. In terms of Centruflow Server, this configuration is largely a form-filling exercise - the server is able to automatically scan and create the necessary configuration files for most properly configured databases.

The second step it to make available (either through creating or downloading) templates. If being developed afresh, this can be an iterative process, with the template becoming more refined through a number of iterations. This is possible as Centruflow does not need a template to display data visually.

1.4 Project Objective and Scope

The primary objective of this thesis was to develop the necessary algorithms and software to enable users to productively use tagging functionality as both a bookmarking and communication mechanism, as well as a means to improve the value of the data being shown to users by inferring new relationships. As mentioned, this required a lot of software development work, much of it on the cutting edge of computer science research, particularly in relation to the fields of the semantic web and social networking.

This project was not about developing a new visualisation technology, although it should make use of and extend available visualisation technology to convey tagging information to the user. Our plan was 'bottom-up' in nature, meaning that we focused on the fundamental issues prior to enabling access to this data in Centruflow. This meant that at whatever point of time we had to stop development due to time constraints, we would have a useful technology 'stack'.

To enable this tagging research, a number of sub-components to this objective were also considered to be within the scope of this masters. These are now outlined below.

Understand Business Use Cases: We needed to understand and develop the business use cases for enterprise social networking, in particular tagging. This includes understanding the limitations of the current Centruflow technology, in particular with reference to business requirements for improved datasource handling.

Algorithm Development: Explore, design and develop algorithms to calculate user trust, entity similarity and distance. These calculations are rather time consuming, as they must operate on a large amount of data. Identifying a means of performing these calculations efficiently was the goal of this stage.

User Interface Design: Prior to commencing coding, a short amount of time was to be dedicated to creating 'low-fi prototypes' of the user interface components that will be used to allow the use cases outlined in appendix A to be met. The purpose of this was to identify any unconsidered issues that may affect development.

Datasource Creation: Suitable data sources were created to be used by Centruflow (and accessed by a 'Centruflow Server'). Appropriate data sources include two disjointed graph databases (one Centruflow database and a 'graph-based' data set) and a tag database.

Design and Development of a 'Centruflow Server': This server acts as the single means for a Centruflow application to read information from remote data sources. This server operates using both SPARQL and REST, and is discussed in considerably more detail in chapter 4.

Development of Centruflow 3.0: This stage had two primary goals:

1. To develop the necessary functionality to allow for Centruflow to communicate with the Centruflow Server.
2. To develop the necessary functionality and user interface components to allow for users to tag inside Centruflow.

1.5 Business Requirements

Centruflow 3.0 was not directly the focus of this thesis, but it formed a necessary foundation to the tagging research. The scope for Centruflow 3.0 was based on the development of a replacement Centruflow visualisation engine. In almost all other regards, Centruflow 3.0 was no different to Centruflow 2.0, and as such it was not overly important to do a fully-fledged requirements analysis process, as this had been undertaken during the development of Centruflow 2.0. However, developing Centruflow 3.0 required a considerable development effort, and so to gain the maximum value from any redevelopment effort, a brief requirements gathering exercise was undertaken.

A number of key people were interviewed during the early stages of this thesis to ascertain what considerations should be included in Centruflow 3.0. In general, companies expressed interest in the following areas:

1. Allowing staff to easily add metadata to resources.
2. Being able to easily modify data within the graph.
3. Being able to build graphs from an initially blank state from within Centruflow (see section 4.5 for more on this).
4. Being able to visualise non-structured data.

We used these requirements in guiding our research and development effort whilst planning for Centruflow 3.0.

1.6 Overview of Thesis

This chapter has briefly introduced the areas of research behind this thesis, and has introduced the business requirements and problem statement. These clearly detail what is both inside, and outside, of the scope of this thesis.

Researching the key areas relevant to this thesis are the topic of chapter 2. This chapter includes topics ranging from semantic similarity and trust systems through to semantic web technologies and even touches on user interface design.

Having identified our goals we move into chapter 3 where we explore the requirements for our mathematical algorithms. Following this, we proceed to develop the mathematical functions that meet these requirements.

Chapter 4 discusses the work we undertook to implement our mathematical algorithms in software. Following this, we discuss the implementation of both Centruflow 3.0 and our new Centruflow Server software.

Having implemented the software required for this thesis, we proceed to validate our approach in chapter 5. This validation tests the value gained by tagging resources within Centruflow by some prototypical user profiles, and checking what trust values we receive for each user. In addition, we look at what inferred relationships are created.

Finally, chapter 6 summarises the results of this thesis and recommends future work.

Chapter 2

Background

In undertaking this thesis, there was a number of areas that we felt we needed to research and appreciate in a greater depth. The range of topics that we researched was varied, ranging from data retrieval to user interface design. This chapter attempts to highlight and discuss these areas.

2.1 Semantic Web

> "Any enterprise CEO really ought to be able to ask a question that involves connecting data across the organization, be able to run a company effectively, and especially to be able to respond to unexpected events. Most organizations are missing this ability to connect all the data together."
>
> - Tim Berners-Lee [28]

The semantic web is a vision driven by the W3C, who have tasked themselves with the development of the necessary standards and technologies to enable this vision. These standards and technologies are all in various stages of their life cycles ranging from being the de-facto standard to being emerging recommendations. To indicate the relationships between these technologies, the W3C has developed a 'layer cake' diagram, which is shown in figure 2.1. This diagram has gone through a number of permutations since its inception, but in general they all portray the same concept, that being the inter-relatedness of a number of key technologies related to the semantic web. In terms of this thesis, our interests lie in a number of these layers, including XML, RDF, RDF-S, OWL, SPARQL, trust and user interfaces/applications. Of course, the user interface/applications layer is Centruflow itself, and our approach to the trust layer is implemented using our user trust calculations in section 3.2. The remainder of this section is dedicated to detailing the standards that we are most interested in based on the W3C layer cake diagram.

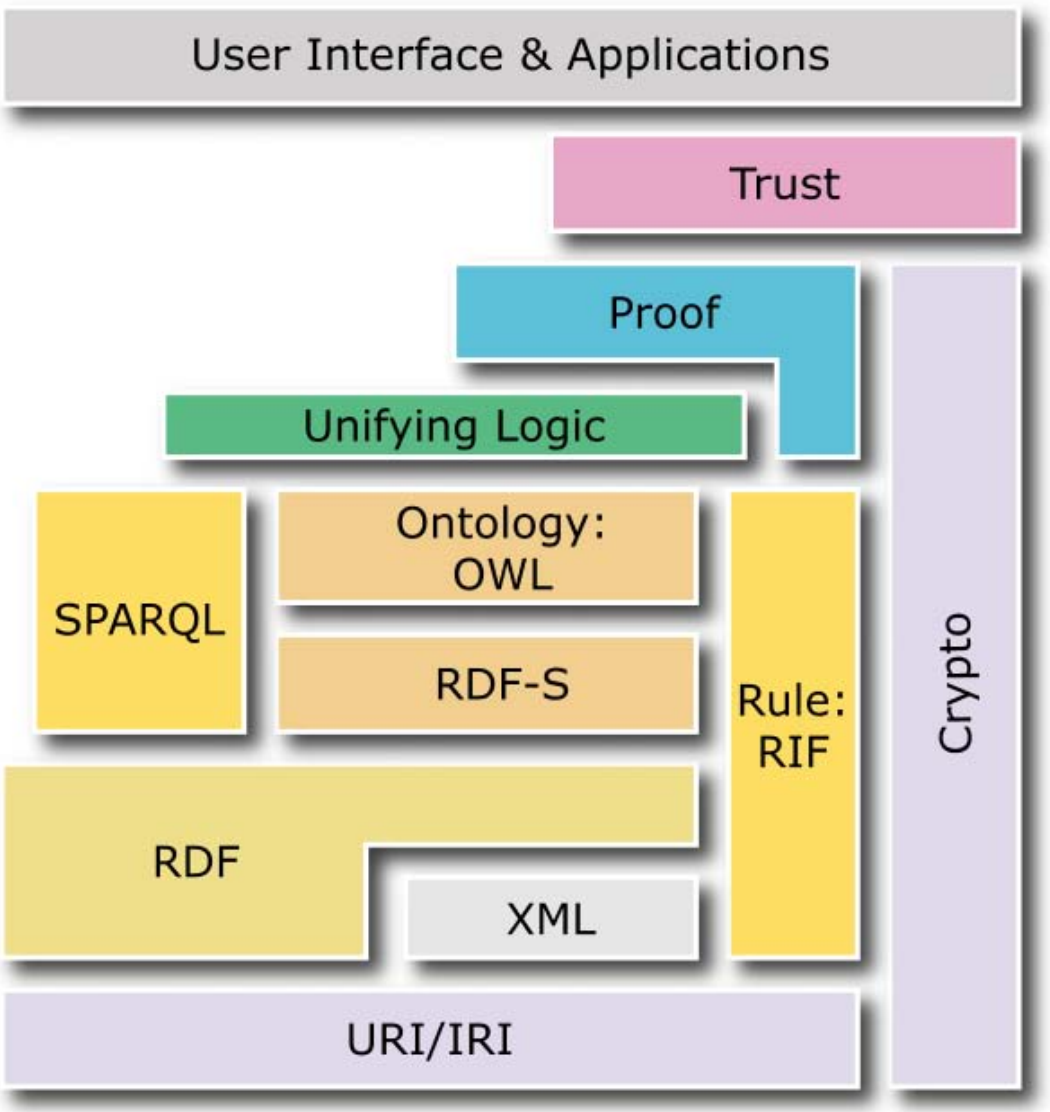


Figure 2.1: The semantic web layer cake [29].

2.1.1 XML

Extensible Markup Language (XML [30]) has been around since 1996, and is essentially a text format for representing data using tags that allows for hierarchical structures to be easily built. This allows for applications to exchange data in a syntactically common format, as opposed to each application creating its own syntax. This means that there is no need to write complex XML parsers, as these are no common 'library' functions provided by most programming languages. Despite this, it is still necessary to write the far more simpler 'importers' and 'exporters' for the specific XML syntax relevant to the application. XML has a number of benefits, including:

- It can be easily generated by a web server and pass through proxies (due to being plain text).
- It can easily be parsed as most programming languages now natively support XML.
- It can be easily transformed from one XML syntax to another using XSLT, which is essentially a 'stylesheet' for XML that can completely transform it.
- It can be easily queried using XPath, which is a specially designed query language able to retrieve data from an XML file using simple queries.

XML is shown near the bottom of the layer cake in figure 2.1, symbolising the fact that XML is critical to the semantic web as it provides a means of data interchange. To be fair, XML is not the only data interchange format, as RDF is a general purpose language for representing information. This is discussed in more detail shortly.

The minimum requirement for a file to be considered an XML file is that it be 'well-formed', which includes requirements [30, 31] such as:

- proper ordering of tags (i.e. closing a child tag before closing its parent tag),
- closing all tags (for example, `<tagName />` or `<tagName>text</tagName>`),
- using quotes around all attributes (for example, use `<tagName attr="north"/>`, not `<tagName attr=north />`).

A well-formed XML document can be sent between systems and parsed, but it can not be guaranteed that the contents of the XML file conforms to any particular syntax or structure. To enforce these requirements, it is necessary for an XML file to move to being a 'valid' file, the next step above well-formed XML. This means that the XML can be validated against an associated document type definition (DTD) or schema file. By being validated, it is possible to be certain that the data being provided in the XML meets the requirements of the parsing application.

2.1.2 RDF

The Resource Description Format (RDF [32]) is a "language for representing information about resources in the World Wide Web" [33]. It is commonly expressed using an XML syntax, but other formats do exist (such as Notation3 and Turtle [34, 35]). RDF represents knowledge by using triples, which consist of a subject, predicate, and an object (this is known as 'SPO form'). A set of triples results in a graph, where each triple is a single arc in the graph. There is a link between RDF and rule interchange formats (such as SWRL which is discussed in section 2.1.5), as whenever a triple is satisfied, the corresponding rule formula `s'[p' -> o']` is also satisfied.

A good description of RDF, taken from [33], is quoted below:

> "RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created."

As shown in figure 2.1, RDF forms a critical building block of the semantic web, given its ability to express relationships between resources (or knowledge) which is so fundamental to the semantic web. The reason for this is that the triples can be understood by software agents perhaps more easily than by humans, and this for the first time allows for software agents to intelligently use information found on the semantic web.

In developing Centruflow 3.0, we wanted a means to join together multiple data sources, and it is through RDF that we found the best possible means. Using RDF, we have a common format to express links between data, irrespective of its source.

2.1.3 Ontology Languages

While RDF allows for the expression of relationships between resources, it does not allow for the definition of classes and their related properties and instances. This is the role of ontology languages such as RDF Schema and the web ontology language (OWL). Ontology sits in the middle of the semantic web layer cake, and makes use of both RDF and (optionally) XML.

RDF Schema is the simplest ontology language developed by the W3C. RDF Schema (and all ontology languages) can be difficult to comprehend at first, which is highlighted in [36]:

> "If RDF is a way of describing data, then the RDF Schema can be considered a domain-neutral way of describing the metadata that can then be used to describe the data for a domain-specific vocabulary."

To try to draw a comparison, this is similar in concept to what happens in relational databases, in that the database schema is built to describe what information it can contain, and what relationships can exist between this information. As noted in [37], "RDF Schema defines classes and properties that may be used to describe classes, properties and other resources". Whilst RDF Schema appears to have been largely eclipsed by OWL, it is important to highlight RDF Schema as it does provide the first, and most minimal, ontology language for RDF. This can be very useful in many circumstances where a fully-fledged OWL ontology is overkill.

There are three versions of OWL, known as OWL-Lite, OWL-DL, and OWL-Full, where each of these versions is an extension of its predecessor, meaning that all valid OWL-Lite ontologies are also OWL-DL ontologies, and so on. In addition, OWL is an extension to RDF Schema. Because of this, each version offers an increasingly expressive language and more functionality which in turn leads to improved machine interpretation, but increases the potential for increased complexity and decreased decidability. In places where RDF Schema already has functionality, OWL tends not to implement it (as it is possible to use RDF Schema statements within an OWL ontology), but this is not always true (for example, both languages offer a separate means of defining classes, instead of OWL simply using RDF Schemas rdfs:Class).

We use OWL-Lite as a means to describe relationships between different tags, which we can then use to infer new tags based on the tags input by users. This is discussed in more detail in section 4.2.2.

2.1.4 SPARQL

SPARQL [38] is a recursive acronym for 'SPARQL Protocol And RDF Query Language'. SPARQL is a query language used to query RDF graphs, much in the same fashion that SQL is used to query relational databases. SPARQL has had a rocky start to life, and is still not yet a W3C standard for querying RDF graphs, but it is nearing this stage. It was released as a candidate recommendation in April 2006, but then was returned to being a working draft in October 2006 due to two unresolved issues [39]. As of June 2007 it has returned to being a candidate recommendation. As noted in [38]:

> "The SPARQL query language is based on matching graph patterns. Graph patterns contain triple patterns. Triple patterns are like RDF triples, but with the option of a query variables in place of RDF terms in the subject, predicate

or object positions. Combining triple patterns gives a basic graph pattern, where an exact match to a graph is needed."

SPARQL is a new addition to the semantic web layer cake, having been omitted from earlier versions. This suggests that SPARQL is about to take on the role of being the standard

2.1.4.1 SPARQL Query Forms

SPARQL has several query forms, unlike SQL with its SELECT query, which means the user is provided with a greater variety of means for querying data sources and, if necessary, transforming the results. The most analogous query in SPARQL to SQL is SPARQL's SELECT query, which returns all variables in a result set exactly how a SQL SELECT query returns a result set. SPARQL SELECT queries do look somewhat like SQL queries, so they can be read by a newcomer, but are different enough to be quite difficult to write. An example query is shown in listing 2.1, which is one of the main queries presently used by a Centruflow client as a result of the research undertaken in this thesis (except that the URI is not hardcoded as in this example, but is a variable that refers to a single resource in the visualisation).

```
SELECT ?property ?hasValue ?isValueOf
WHERE {
    { <http://www.centruflow.com/resource/1/2> ?property ?hasValue }
    UNION
    { ?isValueOf ?property <http://www.centruflow.com/resource/1/2> }
}
```

Listing 2.1: An example SPARQL SELECT query used by Centruflow to request all information for one specific URI.

Another SPARQL query form is the CONSTRUCT query form, which returns a single RDF graph based on a specified graph template. As noted in [38], "The result is an RDF graph formed by taking each query solution in the solution sequence, substituting for the variables in the graph template, and combining the triples into a single RDF graph by set union". This allows for queries to be performed, and the results transformed into a suitable graph structure prior to the client receiving the data, reducing the amount of overhead needed by the client. An example CONSTRUCT query, from [38], is shown in listing 2.2.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT  { <http://example.org/person#Alice> vcard:FN ?name }
```

```
WHERE      { ?x foaf:name ?name }
```

Listing 2.2: An example SPARQL CONSTRUCT query.

The SPARQL ASK query form allows for the caller to test if the provided query pattern has any solutions. This is a simple query form, simply returning a boolean value to represent whether any solutions exist. An example ASK query is shown in listing 2.3, once again taken from [38].

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name "Alice" }
```

Listing 2.3: An example SPARQL ASK query asking if there is any user with a foaf:name of "Alice".

The final query form available in SPARQL is the DESCRIBE query form, which returns "a single result RDF graph containing RDF data about resources" [38]. Two examples of DESCRIBE queries are shown below in listings 2.4 and 2.5.

```
DESCRIBE <http://example.org/>
```

Listing 2.4: The simplest SPARQL DESCRIBE query.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x WHERE
  { ?x foaf:mbox <mailto:alice@org> }
```

Listing 2.5: Another SPARQL DESCRIBE query asking for all information on a person with the email address "alice@org".

In developing Centruflow 3.0 as part of this thesis research, we only made use of the SELECT query, but we believe that for future development, these additional query forms could become incredibly useful, particularly CONSTRUCT.

2.1.4.2 SPARQL Query Functionality

SPARQL queries offer a considerable amount of expressiveness in terms of being able to easily retrieve desired information. SPARQL provides a large number of useful functions to modify the structure of results, known as solution sequence modifiers. If a SPARQL query has more than one modifier, then they are applied in the order shown below. Each modifier belongs to one of the following types:

Order modifiers The order modifier is the ORDER BY clause, and it operates in a fashion similar to that of SQLs ORDER BY clause. The order modifier changes the order of results returned to a SELECT query only - it does not have any effect on CONSTRUCT or DESCRIBE queries, as SELECT is the only query form to return a sequence of results. As with SQL, it is possible to adjust the ordering to be either ascending or descending (using ASC() and DESC() order modifiers), as well as modify the order by more than one variable. An example order modifier SPARQL query is shown in listing 2.6.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name }
ORDER BY DESC(?name)
```

Listing 2.6: A SPARQL query using an order modifier.

Projection modifiers Projection modifiers simply retrieve a subset of the variables returned in a query, ignoring all other variables. This is shown in listing 2.6 where only `?name` is requested, despite `?x` also being available.

Distinct modifiers As with the DISTINCT operator in SQL, the distinct modifier in SPARQL simply eliminates duplicate results from a solution.

Offset modifiers The offset modifier works in the same fashion as the SQL variety, causing the result set to start a specified number of rows after the first result of the result set. The offset modifier will not be useful unless an order modifier is used to make the order of results predictable. This is shown in listing 2.7.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE  { ?x foaf:name ?name }
ORDER BY ?name
LIMIT  5
OFFSET 10
```

Listing 2.7: A SPARQL query using an offset modifier a limit modifier and an order modifier.

Limit modifiers As shown in listing 2.7 above, the limit modifier sets an upper limit on the number of results returned in a query.

2.1.4.3 SPARQL/Update

At this point in time, SPARQL is a read-only query language, that is, it does not offer an equivalent to the SQL INSERT, UPDATE and DELETE statements. This is set to change in the near future with a new language extension known as SPARQL/Update designed to support this requirement [40]. It should be noted that RDF is more commonly used to view read-only data, so lacking these language features is not as critical as SQL lacking this functionality.

SPARQL/Update is a language based on SPARQL that can be used to update data sources. Two example statements (taken from [40]), are shown in listings 2.8 and 2.9.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
MODIFY <http://example/bookStore>
DELETE
 { <http://example/book3> dc:title "Fundamentals of Compiler Desing" }
INSERT
 { <http://example/book3> dc:title "Fundamentals of Compiler Design" }
```

Listing 2.8: An example SPARQL/Update query fixing a spelling mistake.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
DELETE
 { ?book ?p ?v }
WHERE
 { ?book dc:date ?date .
   FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
 }
```

Listing 2.9: An example SPARQL/Update query deleting all books with a date before the year 2000.

2.1.5 SWRL

To the right of the semantic web layer cake is 'rules', which presently is dominated by SWRL, or the Semantic Web Rule Language [41]. SWRL was proposed in May 2004, and is still in the proposal stage. It is based on OWL-Lite and OWL-DL as well as some parts of RuleML [42]. It has a rather convoluted XML syntax but an easily understood human-readable form. The human-readable form is shown below in listing 2.10.

```
hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)
```

Listing 2.10: A human readable SWRL rule.

The XML form of this rule is shown in listing 2.11.

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example1"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom swrlx:property="hasParent">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="hasBrother">
      <ruleml:var>x2</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

Listing 2.11: The XML form of the human-readable SWRL rule.

In this thesis we do not make use of SWRL, but it certainly is interesting in the longer term as a means of inferring additional information. The fact that we can consider SWRL at all is one of the major achievements of this thesis research, as prior to this it would have been considerably more difficult to make use of a rules engine, whereas SWRL almost comes 'for free' simply by including a semantic web framework in both Centruflow client and server software.

2.1.6 What Is The Semantic Web?

In our opinion and understanding of the semantic web, the goal of the semantic web is simple to describe, but complex to successfully implement. We believe the semantic web enables the intelligent and automated reuse of the masses of data that is available on our computers and networks for our benefit. Data is not valuable in and of itself, and it is through the conversion to *information* that we as humans can actually proceed to be better informed as to the availability of options and the consequences of our actions.

Before this is possible, the groundwork must be put into place, and that is the current state of the semantic web. RDF and OWL are slowly becoming more stable as standards definitions for the means of semantically marking up data, SPARQL is becoming the de facto means to query such data, and more frequently we are encountering situations where, visibly or not, applications (and this includes web applications) are using a solid semantic web foundation to not only give themselves a 'foot up', but to also be more interoperable in the long term.

Once this groundwork is in place, the varied ideas that have been floated by the visionaries behind the semantic web (notably Tim Berners-Lee) become a lot more realistic. We are of

the belief that this is no longer a matter of 'if', but 'when', and perhaps most importantly, depends now only on the uptake of the W3C standards by the large software vendors such as Microsoft, IBM, Oracle, and SAP. This has already begun.

To be clear, the semantic web will not make websites any prettier than they are now (thanks to the Web 2.0 emphasis on design), but it will allow them to be smarter, and more connected. People have reserved the term 'Web 3.0' for many things (including the semantic web faction), so we will avoid confusing this any further, but suffice to say, the semantic web will be the next (or the next next) big thing for the Internet.

Behind the scenes of the semantic web, there is a general trend towards enabling systems to use metadata to dynamically connect services and to syndicate new services. This will enable smart agents to more easily communicate between systems, but what is lacking presently is the standards that will enable this. The work on such standards will surely happen, but as with the work so far, it is important for researchers and the W3C to not get too far ahead of themselves.

2.1.7 Open and Closed-World Assumptions

The concept of open and closed worlds is based in the domain of logic: can we assume that by not knowing something, that it is true (or at least not false), or should it always definitely be false? The closed world assumption is the presumption that what is not currently known to be true must be false, whereas the open world assumption assumes that if something can not be proven to be true, then it does not automatically become false. In other words, knowledge that is unknown (i.e. not explicitly stated or that can not be inferred) is declared to be unknown, rather than false (i.e. wrong). Finally, the inverse of the closed world assumption is the concept of 'negation as failure', which states that every predicate that can not be proven to be false must be true.

Some examples include:

- Databases are open world, as they can not give definitive false answers should information not be present in the database.
- Similarly, RDF and OWL are also open world as it is not possible to include every bit of information in a RDF graph, and as such, tuples not explicitly defined in an ontology are assumed to represent a fact that is unknown, rather than false.

Therefore, we can say that closed-world assumptions make sense when we know that the information we have is complete (obviously in a small domain such as an employee database or product catalogue). The open world assumption makes more sense when dealing with incomplete datasets, such as those found in big databases and in the semantic web. For the

purposes of this research, we must assume then that our data belongs to an open world, and as such we can never be certain about a negation statement as we, in most cases, will not have a full set of data.

2.2 Trust Systems

Trust systems have been researched for a long time, with an array of approaches being taken by researchers. Early on in the project scoping phase of this masters project, it was identified that a trust system would need to be developed to enable Centruflow to effectively deal with user taggings, such that poor tags could be filtered out and popular tags 'float' to the top of any tag suggestions. Not only was this important in terms of displaying useful tags to the user, it was important also as a means of improving the quality of our resource distance algorithm. To understand our options, we reviewed a large amount of prior research, and summarise some of the more valuable findings in the remainder of this section.

In terms of the semantic web, trust is shown to be incredibly important by its frequent placement at the top of any 'layer cake' diagram, such as that shown earlier in figure 2.1. Despite this, the issue of trust on the semantic web has yet to have been thoroughly researched, with only minor progress as of the time of this thesis [43]. In one interesting discussion, [44] discusses how the semantic web could become more trusted without the "existence of a complex public key infrastructure". Outside of the semantic web, trust systems have been researched for a considerably longer time, and it is these systems which we briefly review now.

In [45], Matthew et al decide to not use a subjective approach, which would require users to be responsible for ranking other users. Instead, they rely on each user creating a 'web of trust', where each user only inputs trust details about a small number of people closest to them. Instead of calculating a single trust value for each user, they suggest using these trust values to create a personalised set of trusts allocated to each user. In this way, a users trust value is in fact a set of trust values (which presumably can then be averaged out to find a single value).

In [46, 47], Golbeck et al extend the FOAF[1] RDF schema to include the ability for users to specify the level of trust that a person holds in other people. They then develop algorithms that can infer the trust relationships between people and the trust ranking for individuals. This allows them to rank email based on the trust ranking of the sender, and their relationship to the recipient.

The term 'web of trust' originated in the IT industry with the introduction of 'Pretty Good Privacy' (PGP), a computer program that allows for people to encrypt and decrypt their

[1] FOAF, or Friend Of A Friend, is a means of describing your friends and their relationships to you using RDF. People commonly talk about having a FOAF file on their website, for example.

files/communications, and also authenticate themselves [48]. PGP is based on public-key encryption technology. PGP allowed for people to encrypt and sign their emails prior to submission, and required the email recipient to have the senders public key to decrypt the message. This approach was not perfect [49], but it did begin to place emphasis on the issue of trust. Unlike the papers discussed above, PGP has no choice but to be based on people manually maintaining their 'keyring' of trusted friends, as realistically this is the only approach applicable in this situation.

In these papers we feel that there is an inherent weakness in the system, in that users are forced to rate their belief of other peoples trustworthiness, which is always open for abuse. This could potentially be compounded in corporate environments where factors such as hierarchies and office politics play a greater role in interpersonal relationships. This issue will be especially prevalent in the bootstrapping phase of any trust system, where the number of user trust rankings is relatively small, providing more emphasis towards the malicious trust rankings.

2.3 Semantic Similarity

Semantic similarity is the concept of being able to measure the distance between two terms based on a metric which determines the likeness/similarity of the terms. Being able to easily and efficiently calculate the similarity of two objects has long been a topic of research. Modules exist in many languages to enable easy reuse of these metrics [50], and there are web-based systems that are available to perform these calculations [51].

What we found when researching semantic similarity is that the most commonly used approach is to use tools such as WordNet and GeneOntology [52, 53]. With these tools, it is possible to calculate the shortest distance between two words by performing edge counting [54, 55]. This works by counting the number of edges that exist in the shortest path between two words in a hierarchical taxonomy. This count of edges is often referred to as the 'distance' between the terms, but is prone to errors such as synonyms and homonyms confusing the algorithm.

Whilst semantic similarity is incredibly useful in many circumstances, it quickly become apparent that it would not be feasible to use research in this area to help our cause. This is because, from the perspective of user tagging, we are not interested in the single tag-tag relationships and their respective distances, but rather the degree of similarity of one resources set of tags with another resources set of tags. In effect, we are interested in the degree of overlap between the tag sets of two resources. This was our goal in this thesis, as it allows us to calculate the similarity between two resources based on the tags they have each received.

2.4 Representation State Transfer (REST)

Representational State Transfer (REST) is the name given by Roy Fielding in his PhD thesis [12] to describe an architectural style of networked systems. Roy Fieldings comments on REST are:

> "Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

As mentioned, REST is an architectural style, and as such does not have a standard, in a similar fashion to the client/server architectural style. Despite not being a standard, REST does indeed use standards, including HTTP, URL's, and XML/HTML. In fact, many web services that have existed on the web have been REST-based without being aware of this fact - in this way, REST is a series of 'best-practices' that many developers have for a long time known about and used. In addition to this, there are projects attempting to standardise how REST works in particular programming languages, for example the Java Community Process (JCP) is presently working on JSR-311 which is titled 'JAX-RS: The Java API for RESTful Web Services'. This Java Specification Request (JSR) also has a reference implementation called Jersey [56].

REST can be considered to be very lightweight and more readily implementable than other web service architectures such as SOAP. This makes REST interesting to us as it offers a simple means to expose the tags database (which as mentioned in section 4.2.1 contains a considerable amount of useful information).

This trend towards more lightweight frameworks has increased in recent years. This can be put down to the increasingly complex and heavyweight frameworks that are being developed. Examples of such frameworks include the previously mentioned SOAP, the Java enterprise edition (Java EE) and enterprise Java beans (EJB) frameworks, the Open Management Groups CORBA, among many other complex frameworks. From a developers perspective, lightweight approaches allow for more rapid development with less hurdles to jump.

2.5 Tagging Systems

2.5.1 Tagging

Tagging allows for users to label a resource with a short textual 'tag'. Commonly, a tag is a single word, or a few words joined to remove any spaces, alternatively using the underscore

character or 'camel case'[2]. Any discrete chunk of information can be tagged. Tagging in this (digital) sense has been around since at least 1988, where it was used in Lotus Magellan to allow annotations of documents on a persons hard drive [57]. This made it easier for users to subsequently find their documents, but it lacked the social aspect that tagging has now.

Tagging really exploded along with the web 2.0 movement. Two examples of web 2.0 tagging include Del.icio.us and Flickr [58, 59]. Del.icio.us is an online bookmarking site for websites, allowing users to tag their bookmarks for easy retrieval, whereas Flickr is an online picture sharing site where people may tag pictures to make them part of a group, allowing other people to browse using their interested tags. The benefits of tagging in these instances include:

- Tags can be used as a means of bookmarking information for later retrieval.
- Communities can form around a set of tags.
- 'Tag clouds' can be generated through the linking of all user tags. This allows for people to browse from one tag to the next, discovering potentially useful information.

Tagging is explored in much more depth in subsequent chapters, as it forms the fundamental basis of this thesis.

2.5.2 Folksonomy

> "Part of the allure of classifying things by assigning tags to them is that the user can give free reign to sloppiness. There is no authority - human or computational - passing judgment on the appropriateness or validity of tags, because tags have to make sense first and foremost to the individual who assigns and uses them. And yet, the whole point of distributed classification systems (DCSs) such as Del.icio.us and Flickr is that the aggregation of inherently private goods (tags and what they describe) has public value: When people use the same tag to point to different resources they are organizing knowledge in a manner, commonly referred to as a folksonomy, that makes sense to them and to others like them. In other words, the tag is the object that brings a resource and a social group together via the shared meaning of a word."
>
> - Ulises Ali Mejías [60]

[2]Camel case is the compounding of words with each word having its first letter capitalised. It is very common in programming as it enables for easier reading of code.

The term folksonomy comes from the merging of 'folks' and 'taxonomy', and is commonly attributed to Thomas Vander Wal [57]. Folksonomy is the concept otherwise known as socialised tagging, where individuals are free to tag resources in whatever means they see fit - there is no rigid taxonomy which is required to be used. Folksonomies become popular with the advent of the web 2.0 movement, as it enabled users to far more easily and rapidly add tags. Both Del.icio.us and Flickr are examples of folksonomies, as they both enable users to tag information, and as more tagging data is added, it makes the tagged data more easy to search, discover and navigate over time.

Applications built around folksonomies often allow users to explore other users tags, to allow for people to understand the interests of these users. This is very useful when there is no trust model in place, as when a trust model exists, knowing who has created tags can be an avenue for abuse. In the case of Centruflow and this thesis, we had to carefully weigh up the benefits of exposing the people behind the tags versus the viability of our trust model.

Despite the benefits of folksonomies, they are not the solution to the meta-classification problem for everyone, but they certainly can help in a number of circumstances. There are a number of areas where folksonomies are criticised [61], including:

- Their lack of control for synonyms and homonyms [62]. We propose a partial solution to this problem later in section 4.2.3.
- Users being uninformed in how best to apply tags (i.e. through lack of a set of 'best practices'). This includes issues such as the use of lower case and upper case characters, singular versus plural terms, and how to group multiple words (i.e. camel case or underscores).
- Lack of any spell checking prevents useful but misspelt tags from being found again.
- Personal tags that have no value to other users. These are known as subjective tags, and are discussed in more detail in section 2.5.6. In addition, we developed a prototypical user type that exhibits this behaviour in section 3.2.1; we call them 'bookmarkers'.

A folksonomy that does not have solutions to these problems is not actively engaging in the process of folksonomy minimisation, and as such the value of the folksonomy to all users decreases, as more 'noise' enters users search results. We tried our best to minimise these problems, and others, such that we could claim to be actively minimising our folksonomy. This is detailed in later sections.

Figure 2.2: The 'MySQLicious' tag database schema.

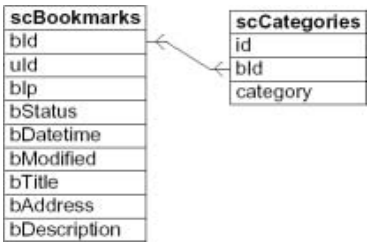


Figure 2.3: The 'Scuttle' tag database schema.

2.5.3 Tagging Architectures

With the advent of the 'Web 2.0' movement, tagging exploded onto a number of well-established websites, leading to tagging becoming available to a large number of web users. Unfortunately, whilst the tags were indeed useful for users as a means of bookmarking and aggregating information, each tagging system was a proprietary implementation. This meant that tags were unable to transition between different systems (such as using Del.icio.us tags in Flickr). This means that no single tagging folksonomy exists - each website is effectively a tagging island. As of the time this thesis was being written, this was still sadly the case. There is however growing interest in defining a common means for sharing tag data. This is the subject of this section.

2.5.3.1 Early and Most Common Tag Database Schemas

Much discussion exists on the Internet regarding proper database structures to enable tagging functions [63, 64]. The three most common database schemas are shown in figures 2.2, 2.3 and 2.4 (all taken from [63]). What these figures show are three of the more common tag database schemas as of 2005.

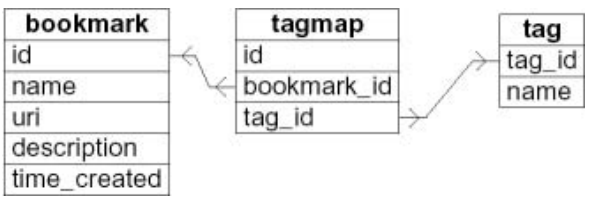


Figure 2.4: The 'Toxi' tag database schema.

The database schema suggestions put forth by these websites vary in query complexity, memory requirements, and degree of normalisation. We did not particularly agree with any of the solutions offered, as they offered more functionality than was necessary in our situation, and this extra functionality would have the cost of extra database query time. We did however take the discussions into account when we were developing our own tags database schema in section 4.2.1. We now discuss the three main schemas outlined.

The first schema, shown in figure 2.2, is known as the MySQLicious schema as it is the schema used by MySQLicious [65] to import Del.icio.us bookmarks into a MySQL database. Our main issue with this database schema is that it is fully denormalised, requiring all tags that a user inputs about a particular URL to all be in one 'tags' field. This means that there is increased processing requirements as the tags field will need to be tokenised in order to deal with individual tags. In addition, the number of tags able to be stored in the tags field is constrained to either the size of a varchar (256 bytes), or is hobbled by the overhead of a text data type. Despite this, the MySQLicious database schema is the closest to our final design, with the exception that each tagging only contains one tag - in other words the users input is tokenised once prior to submitting to the database. In this way, we are substituting memory requirements for processing requirements.

The second schema, known as the Scuttle tag database schema (figure 2.3) uses two tables to organise its tags. The 'scCategories' table is used to hold the tags, with all the other metadata contained in the 'scBookmarks' table. This is a far more normalised database schema compared to the MySQLicious schema, and becomes increasingly more valuable as the amount of metadata recorded increases. Another benefit is that this schema does not impose any kind of limit on the number of tags that can be applied to a resource. The downside to this schema is that it requires a join for any useful query to be performed. We would consider moving to this schema should our schema begin to gain more metadata, and once tests regarding the efficiency of joins were performed.

The Toxi schema, shown in figure 2.4, is simply a more normalised version of the Scuttle schema, trading processing requirements for more efficient memory use.

2.5.3.2 Ontologies

The next step after storing tag data is to be able to map the tagging information to resources, which is where projects such as the TagCommons project and the Tag Ontology project come in [66?]. These projects are the leading projects in the area of sharing tag data over multiple web applications. They both attempt to define a common ontology to share tagging information between systems, and in the case of the TagCommons project, investigate the wider needs of such a solution. The goal is to mitigate the issue of differing database schemas discussed above by providing an abstraction layer using ontologies. What

this means is that these projects are trying to come up with ways for tags to be used between tagging systems, linked together over the Internet.

Unfortunately, both projects have not seen updates in at least six months. For this reason unfortunately neither project was pursued any further, and instead we defined our own, simpler architecture. The downside to this is that the Centruflow approach also lacks the ability to connect with external tagging systems. We hope that in the future such projects can successfully develop ontologies to allow for the easy interchange of tagging metadata.

2.5.4 Tagging User Interface

Tagging is a very social mechanism, and is something that should be simple and fast to perform: there should be minimal hurdles for the user. Because tagging is generally used on the Internet[3], it made sense to try and emulate the user experience of the web within Centruflow. To fully understand how to approach a tagging user interface, we therefore turned to Del.icio.us as a source of inspiration. This section discusses the main user interface components of Del.icio.us. The results of this research can be seen in section 4.4.6, where we implement a solution.

As mentioned earlier, Del.icio.us is a web-based website bookmarking tool. It allows for people to easily add new website URLs by categorising them with zero or more tags. By tagging a URL, the user aggregates their bookmarks into a number of subcategories. These subcategories can be browsed by both the user and any other user. Del.icio.us uses all user tags to create related tag lists, allowing users to find other potentially interesting URLs by browsing these suggested tags. Del.icio.us requires users to sign up for a free account prior to allowing them to store any information, but any user can browse Del.icio.us. Finally, Del.icio.us allows users to mark a bookmark as private, meaning that no other users can see the URL that the user has bookmarked.

2.5.4.1 Adding a Bookmark

Figure 2.5 shows the popup window that allows users to add bookmarks to their profile. The most interesting aspect from a Centruflow perspective is the 'recommended tags' and 'popular tags' list at the bottom. Clicking on any of these tags automatically adds them to the 'tags' text field, allowing for users to easily copy and use other users tags for this particular URL. The primary advantage behind this is the ability to keep the tagging folksonomy small, which as mentioned earlier helps to increase the quality of the folksonomy when searched or browsed by users. In particular, this helps with the resource distance calculations (see section 3.3).

[3] Despite this statement, tagging is beginning to make inroads into the desktop, see for example Microsoft Windows Vista and Microsoft Office 2007. Also, Lotus Magellan, mentioned earlier, was a DOS-based application.

Figure 2.5: The Del.icio.us 'add bookmark' screen.

2.5.4.2 Browsing Bookmarks

Having enabled users to input their bookmarks and to categorise them into any number of categories using tags, the next obvious step is to provide the user with a means to easily find their bookmarks at a later date. This functionality is shown in figures 2.6 and 2.7. This approach differs from our approach in this thesis, as we do not want to simply use a list to represent tags when we have a powerful visualisation engine available to us. For this reason, our approach was to instead visualise the tags as a separate 'data layer' that a user may switch on to see the tags as related to the presently visible resources. This is described in more detail in section 4.4.6.4.

2.5.4.3 Inferring Tag Relationships

Based on the combinations of tags that users input into Del.icio.us, Del.icio.us is able to infer relationships between tags, as shown in figure 2.8. This is a useful function of Del.icio.us, as it provides a means to discover other relevant web pages related to a particular topic. This is where the value of tag suggestion comes into play: the smaller the number of unique tags within the tag folksonomy, the tighter the inferred relationships between tags can be.

2.5.4.4 Browsing Other Peoples Bookmarks

It is possible to look at anyones bookmarks with Del.icio.us. This has considerable use in the context of social networking (including within Centruflow), but as discussed, we actively

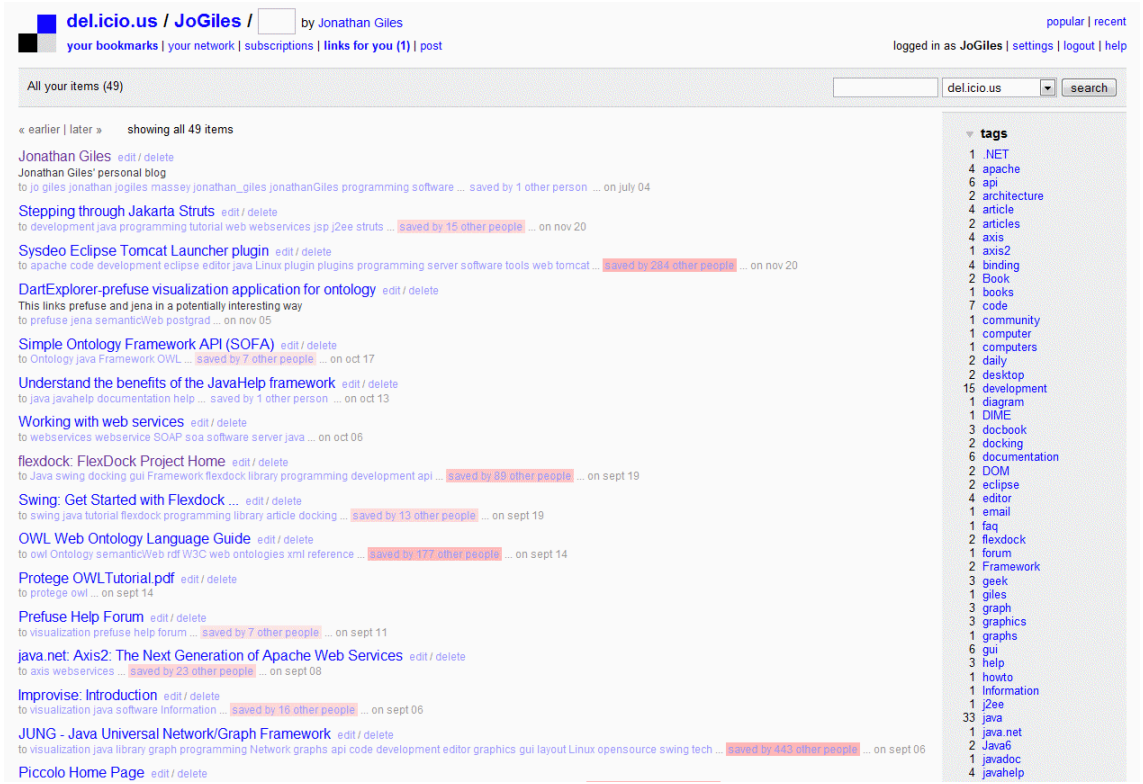


Figure 2.6: The main Del.icio.us page

Figure 2.7: Viewing tags as a tag cloud.

Figure 2.8: Del.icio.us can infer relationships between tags based on peoples taggings.

discouraged introducing any form of personality into our system for fear of abuse of the trust system. This meant that users were only able to review information based on the relationships inferred by Centruflow, and not based on the people behind the information. We felt that this was both important and novel, but once again, this will be discussed in more detail later on.

2.5.4.5 Tag Recommendation

Recommending tags to the user helps in reducing the size of the folksonomy, as users are carefully guided to use particular tags. There are two approaches that must be pursued to reach this goal, which are:

Popular tag suggestion When a user selects a resource that they wish to tag, they should ideally be presented with a list of other tags that have been applied to this particular resource already. This encourages the user to reuse tags, rather than create their own. The user is however not forced to use these suggestions if they do not wish. This means that the user should feel empowered to tag as they wish.

The implementation of this approach is rather simple, as it simply equates to a SQL query against the tags database (see section 4.2.1 for more information).

On-the-fly tag suggestion As the user types tags into the text area, they would ideally be presented with tags that match (i.e. begin with) the text that the user has already typed. This would rank the tags so that the more popular tags are shown at the top of the list. Once again implementing this feature is essentially an SQL query, but the user interface for this is much more difficult.

2.5.5 Inferred Relationships

Over and above the first-order benefits of tagging such as ease of bookmarking, collaboration and aggregation of information, a second-order benefit exists that we wanted to expose in Centruflow. This benefit has to do with the linking and recommending of information based on the co-occurrence of tags in the tag sets belonging to each resource. This is partially adopted by Del.icio.us (as shown in figure 2.8), in that a user may find additional interesting information by browsing the related tags, as calculated by the Del.icio.us algorithms.

Our intention was to take this one step further, by visually showing inferred relationships between resources as part of our visualisation (for clarity we differentiate inferred edges with dashed lines). This would allow people to directly explore to other areas of information where an explicit datasource relationship may not exist, but where an implicit relationship may actually exist. This is the focus of our resource distance algorithm, detailed in far more depth in section 3.3.

2.5.6 Public and Private Tags

A very early consideration when developing this system was whether all taggings in Centruflow would be public, or whether a user could choose to mark some of their tags as private. Initially we dismissed this need to differentiate tags as not necessary due to the increased development complexity, but in hindsight, and after reading considerable literature, we deemed that such a piece of functionality would be incredibly useful for a number of reasons, including:

- The folksonomy of public tags can be minimised to only those tags that people deem most valuable to share.
- The 'bookmarker' user type (discussed later in section 3.2.1) would not be unfairly penalised for creating what others deemed to be poor quality tags.
- The amount of computation time required for calculating user trust and resource distance will decrease, as these calculations would only need to be run on the public tags.

- Personal tags will not interfere with the resource distance algorithm, for example, if many people tag different resources as 'mine', then inferred relationships would be created where no such inference should be made.

Once these benefits were identified, we made the necessary modifications to our plan to incorporate the concept of public and private tags. Ideally, personal tags would be the only tags that users would mark as private, with subjective tags (e.g. 'cool') and factual tags (e.g. 'tutorial') being marked as public, so that they may be seen and used by others.

In [67, 68], folksonomies are analysed using three classification categories (personal, factual, and subjective). They claim that

> "Users have their own perspective when describing web resources. They may describe a resource by its type, discipline, content or they may even add new contextual dimensions to it to visualize its application or relation to other neighboring domains."

In [67], they extend the work done previously by adding some additional heuristics, which include:

- Filtering out all tags that are misspelt.
- Counting the number of tag occurrences to indicate the agreed meaning of it (which is what we do). In addition to this, they suggest that lower tag occurrence indicates personal use.
  - As a subset of this point, they say that "compound tags and vague abbreviations are considered personal, since no one knows what they mean, or why they were formed in this way". Because of this, the tag counts of these tags will be at a minimum, meaning that they will be made private automatically. Of course, one point not mentioned in this paper is that should other people begin to use this tag, then the tag should become public.

From the analysis of a very large folksonomy retrieved from Del.icio.us, their findings were that 34% of all tags were personal, 62% were factual, and 4% were subjective. What this suggests is that there will be a large amount of potential 'noise' in any inferred relationships should personal tags be allowed to influence the resource distance calculations. For this reason we included the concept of private tags in our tagging system.

2.6 Reflections

In conducting this background research, we read a considerable number of papers in the areas discussed above. We also joined a number of communities involved in particular research areas, so that we may directly ask the leaders of these projects particular questions. Whilst the research is summarised relatively succinctly above, there is a considerable amount of research underway in these areas. We can only report on our findings at the current time (early to mid 2007), with some understanding of what the future will hold. Looking at older papers we note that they were trapped by the same reality, meaning that they now are outdated and incorrect, as they based their approaches on the likely standards of the time. This is particularly the case in relation to languages used to query RDF data sources. When reviewing these papers, it was important for us to separate the concept from any implementation specified.

Despite the previous statement (and we are tempting Murphy's Law here), we identified that standards are finally starting to fall into place (such as RDF, OWL, SPARQL, and SWRL). The only gap where no standard (and only minimal discussion) exists is in the area of updating RDF stores, which was covered earlier in section 2.1.4. This is definitely a critical issue for Centruflow, as it is always a goal to offer to the user a means to update a subset of the data. Whilst this is important for Centruflow, it does fortunately fall outside the scope of this thesis.

Another issue is in finding software libraries that are still actively being developed: a large amount of useful code exists for embedding into Centruflow, but a lot of it is effectively 'abandonware'. Additionally, some of the more interesting technologies are in licenses not viable for use in Centruflow, such as the GPL. By the end of this research however we did find a suitable collection of libraries to aid us during development, these included Jena, Joseki, Jetty, D2R Server and D2RQ, all of which are discussed later.

Chapter 3

Algorithm Theory And Implementation

In this chapter, we develop a set of algorithms that allow us to calculate the relationship between resources in our visualisation. This calculation is not based on explicit links between these resources, but rather based only on the tag metadata provided by the user. In addition to this, we develop algorithms to calculate the trustworthiness of a user based on their taggings. Finally, we introduce the novel concept of untaggings, whereby users can disagree with a tag, with such a disagreement being recorded and used in subsequent calculations.

Based on our background research, we believe we are introducing a novel approach for these algorithms based around the concept of the inherent value in the co-occurrence of taggings between resource pairs in an RDF graph.

3.1 Introduction

To allow for the requirements outlined in chapter 1.4, we identified two core algorithms that would need to be developed. These algorithms were what would ultimately be responsible for the calculation of the inferred relationship between any two resources. In this chapter we examine the requirements for each of these algorithms, and then proceed to formally define them.

3.2 User Trust

Centruflow users have the ability to tag resources, but not all tags are created equal: many are unsuited to the resource for which they have been applied. There needs to be a

mechanism that can identify the less valuable tags, and to filter them out. This helps to minimise the size of the tags folksonomy, as users will not be given these tags as suggestions. On the other hand, when it is apparent that a tag is largely popular with regard to a single resource, we should multiply the tag to give it added strength, and to once again work towards the minimisation of a tags folksonomy. It is the user trust algorithm, the topic of this section, that attempts to achieve this goal.

In this thesis we have approached this problem in a different fashion than those discussed in section 2.2, by instead requesting that users rate the information presented to them without the knowledge of which other users have also been involved with it, either in creating, agreeing, or disagreeing with it. We do this as we believe humans can not always be truthful about their thoughts on another person, particularly when office politics and personalities come into play. We therefore choose to work around this issue by separating the personalities from the information being presented on screen. With users only able to rate information based on its quality, we can deduce which users are more trustworthy than others based on how they dealt with this information. In effect, we are simply reversing the direction of traditional user trust systems, whilst still ensuring users receive the most useful information with as little 'noise' as possible.

3.2.1 User Profiles

When using a program that allows for users to tag and untag information, it is important to consider the stereotypical types of users that could exist within a company, and to ensure that any trust algorithm can deal with the effect of these people. To create our user trust algorithm, we must take into consideration all of the user profiles and identify means through which we can mitigate any negative effects to user trust values. We will tackle this issue for each prototypical user below.

3.2.1.1 The Guru

The guru is someone in a company that has expert knowledge which may not be widely known by other users. Because of this, the guru is in the position of applying tags that may not necessarily be understood by other users, and at the same time, may find the tags applied by some people to be incorrect due to his advanced understanding. Other users, not understanding the tags applied by the guru, and not knowing that they were applied by a guru, will either choose to ignore or untag the tag. This leaves the guru in a bad position, as he can not possibly get a good trust ranking if users do not understand his tags. If his trust does not increase, then he is likely to find his tags filtered out from being used in relevancy calculations, meaning that overall the system does not appear to be learning from the guru. This could demotivate the guru from inputting further information, and this of course is to the detriment of all users. In summary, the guru tends to apply a lot of

taggings that may or may not be agreed with, and occasionally untags taggings that are not accurate in his opinion.

To help the guru receive the trust value that he deserves, we can either encourage new and unique taggings to be applied, or instead focus on supporting users who engage in the critical process of untagging. Realistically there is only one option that can be taken, and that is the second option, as encouraging users to create new and unique tags will simply lead to an explosion in the size of the folksonomy, and the subsequent relatedness calculations becoming largely worthless. This is one of the main reasons why we wish to restrict the size of a folksonomy (and why the Centruflow application provides tag suggestions to users).

Therefore we can identify the fact that gurus will tend to receive agreement from other users for their taggings and untaggings in the long term. This suggests that recording the time a user performs a tagging may be useful, as we may then delay the use of such taggings in relatedness calculations until such a time that there has been the ability for people to see these taggings, and agree or disagree with them. Additionally, to reward gurus, we should ensure that our trust algorithm pays attention to untagging.

3.2.1.2 The Hypercritical

The hypercritical is a user who finds fault with most information in a system, and therefore spends the vast majority of his time untagging information. Because the hypercritical finds fault in most tags, he ends up applying his own tags, and so in this way the hypercritical resembles the guru. Because the hypercritical does not put in many tags that other people agree with, his tags largely languish as ignored by users, or more accordingly, they are likely to be untagged by other users. In summary, the hypercritical applies many untaggings, and his taggings are more often than not untagged or simply not agreed with.

Because of the note above regarding rewarding gurus based on their untaggings, we have a situation where hypercritical users can abuse this fact, to artificially inflate their own trust value. To negate this abuse, we can identify the fact that hypercritical users tend to create a vast number more untaggings than they would taggings, and that their untaggings and taggings would likely have a small amount of agreement from other users. This differs from a guru in that they would likely have a relatively even number of taggings and untaggings, and also would more likely have a larger number of people agreeing with their untaggings and taggings.

3.2.1.3 The Consensus Seeker

The consensus seeker is someone who tries to reinforce already created taggings, by simply applying taggings that already exist within the system. This means that the consensus

seeker very rarely creates new taggings. This user therefore does not bother to apply untaggings very frequently, as he does not have enough information to know which tags are 'popular to untag'. He may however untag whenever a tag is obviously incorrect.

The consensus seeker is a largely 'untapped' resource for knowledge sharing within an organisation, and somehow needs to be motivated to share any knowledge he may have by applying appropriate taggings. The consensus seeker is not a negative influence on any trust or relatedness calculations by any means, so we do not need to negate their influence.

3.2.1.4 The Bookmarker

The bookmarker is a person who does not care about the social aspect of tagging, and instead simply uses tags to bookmark and easily find information that is pertinent to him. For this reason, the bookmarker creates tags that may not make sense to anyone else - they are comprised of cryptic alphanumeric combinations and shortened words. Because the bookmarker does not care for any social aspects of tagging, he never tags or untags other tags in the system. However, because other users can see the bookmarkers tags, and they make no sense to anyone but the bookmarker, users are likely to untag the tags.

The bookmarker is perhaps the most special case, and as discussed in section 2.5.6, really requires the concept of private tags. This is important for bookmarkers when their tags clearly do not, and never will, allow for increased knowledge within the companies knowledge domain.

3.2.1.5 The Inactive

The inactive is a person who does not choose to partake in the tagging that Centruflow supports. An inactive user that does not contribute anything will not even be considered as part of any relevancy calculations, so this user profile is of no concern.

3.2.1.6 The Common User

The common user is simply the user that uses tagging in the intended way, that is, they seek to either create useful 'bookmarks' for themselves and their coworkers or they wish to improve the companies shared knowledge. In either case, they wish to tag with understandable tags. The common user has no alternative motive to undermine the system and artificially improve their trust value. It is hoped that most users will fall into this category.

Users in this category simply need to be rewarded when they interact with the tagging system, and particularly when they reuse tags already in the system (i.e. when they are

part of the consensus rather than creating their own tags) and when they untag other tags. Of course, they should not be harshly treated for creating new tags at all, but neither can we assume that they are more trustworthy simply by inputting new tags.

3.2.2 User Profile Findings

Based on these user profiles, we have identified some useful criteria that can be used in defining our user trust algorithm. This is shown in table 3.1. Based on our prototypical users, the two most valuable findings were:

1. Untagging should be valued more so than tagging, as it is more of a 'critical review' process than tagging is.
2. We should expect that a user that has created a lot of untaggings to also tend to agree with other taggings, and not simply creating their own taggings (otherwise they are likely a hypercritical).

With these key findings, as well as the prototypical user profiles developed here, we can now proceed to implement our user trust algorithm.

3.2.3 Algorithm Implementation

The input to our algorithm is the 'raw tags' database (discussed in section 4.2.1), where each row is a 'tagging', consisting of a unique combination of a resource (a URI), a user ID, and a tag, as well as other useful metadata. In the case of our user trust algorithm, one of these metadata values is particularly useful, that being the timestamp in which the tagging or untagging was actually applied. In implementing the concept of tags, we treated each tag as an OWL class, as discussed later in section 4.2.2. This approach is based on Peter Mika's paper [69], which defines actors (users), concepts (tags), and objects (resources) as a tripartite graph with hyperedges.

Formally, let R, T, U and D be disjoint sets. R is the set of all resources, T is the set of all tags, U is the set of all users, and D is the set of all timestamps for which a tagging occurs. This is of course an infinite number of values, but we are bound by limited memory and space issues in implementation, so by default we treat this as finite, where timestamps are no earlier than January 1st, 1970 and in fact only exist on days for which a tagging has been applied.

We define the term *taggings* to represent the set $R \times T \times U \times D$. A tagging is a single tag ($t \in T$) applied by a single user ($u \in U$) belonging to a single resource ($r \in R$) applied with

	Quality Tags	Tags Often	Untags Often	Tags Agreed With	Tags Disagreed With
Guru	Yes	Yes	Yes	Uncertain	Moderately
Hypercritical	No	Yes	Yes	No	Infrequently
Consensus Seeker	Yes	Uncertain	No	Yes	Yes
Bookmarker	No (if public)	Uncertain	No	No (if public)	No
Inactive	Uncertain	No	No	N/A	N/A
Common User	Yes	Uncertain	Uncertain	Yes	Yes

Table 3.1: Summary of prototypical users.

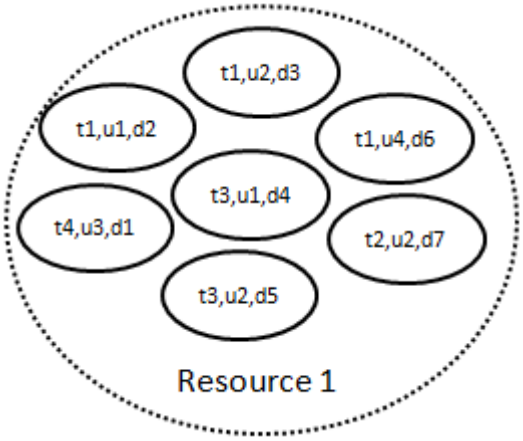


Figure 3.1: An example resource with the tags that belong to it (as well as the associated users and the timestamps at which they were applied).

a particular timestamp ($d \in D$). A tagging is unique. Figure 3.1 shows a single resource with all the taggings that are associated with it. We use this venn diagram metaphor throughout this chapter to more clearly explain the purpose of our mathematics. Our goal is to calculate a value between zero and one for each user who has applied at least one tagging. A trust of zero represents a completely untrustworthy user, whereas a trust of one represents a completely trustworthy person.

We define the term *untaggings* to represent a subset of $R \times T \times U \times D$. An untagging is a single untag ($ut \in T$) applied by a single user ($u \in U$) belonging to a single resource ($r \in R$) applied with a particular timestamp ($d \in D$). Additionally, the set of untaggings belongs to the subset of all taggings, as an untag can only be applied against an existing tag ($t \in T$).

The complete user trust algorithm pulls together two equations which are defined below, and it uses different weightings to emphasise/deemphasise the output from these equations. The actual values of α and β will be determined later, once testing has been undertaken, and so for now α and β are defined to be equal to 0.5.

$$trust(u) : U \rightarrow [0, 1]$$
$$\alpha, \beta \in [0, 1]$$
$$\alpha + \beta = 1$$
$$trust(u) = \alpha \times normalisedUserConsensus(u) + \beta \times normalisedTaggingActivity(u)$$

With this function defined, we now delve into the explanation and development of *normalisedUserConsensus* and *normalisedTaggingActivity*.

3.2.3.1 User Consensus

The user consensus function tackles the question of how often a user is part of the overall consensus, i.e. how often does the user agree with the popular tags and not have their tags untagged by other users. We approach this problem by firstly creating some useful sub-functions prior to creating the actual user consensus function.

Tag agreement is calculated for each user, and is the sum of the number of instances where people have applied the same tag as the user on the same resource. To calculate tag agreement we must iterate over all tags that a user has applied, and for each tag, collect all other equivalent tags that exist for the same resource. The result is therefore a sum of all such 'equivalent taggings' (i.e. where only the user or timestamp differs). This number represents the number of people who have 'agreed with the user'[1].

$$taggingAgreement : R \times T \times U \times D \to \mathbb{N}$$
$$taggingAgreement(r,t,u,d) = \left|\left\{u' \mid \exists_{u' \in U \wedge u' \neq u} \exists_{d' \in D} : (r,t,u',d') \in Taggings\right\}\right|$$

$$tagAgreement : U \to \mathbb{N}$$
$$tagAgreement(u) = \sum_{\exists_{r \in R} \exists_{t \in T} \exists_{d \in D} : (r,t,u,d) \in Taggings} taggingAgreement(r,t,u,d)$$

In summary, *tagAgreement(u)* is the number of people who have agreed with user u by tagging a resource that user u has also tagged, with a tag that user u has used. *taggingAgreement(r,t,u,d)* is the number of people who have applied the same tag t to the same resource r.

Dealing with tag disagreement is equivalent to the above two functions, replacing the *Taggings* set with the *Untaggings* set in all instances. We will therefore not formally define the *tagDisagreement(u)* and *tagDisagreement(r,t,u,d)* functions. In summary, the *tagDisagreement(u)* function is the number of people who have disagreed with this user by applying an untag to a tag on a resource that user u has created or agreed with. *TaggingDisagreement(r,t,u,d)* is the number of people who have applied the same untag t to the same resource r.

We now proceed to define the *normalisedUserConsensus* function, one of the two functions used to determine a users trust value. The user consensus function calculates how good the user is at agreeing with other users (or being agreed with), versus how often they are

[1] Note that this is despite the fact that the user may not have been the tag originator and in reality other users are only agreeing/disagreeing with the tag, not the user behind the tag

tagAgreement	tagDisagreement	userConsensus	normalisedUserConsensus
0	0	0	0
0	1	0	0
1	0	0	0
1	2	0.5	0.025
2	1	2	0.1
1	20	0.05	0.0025
20	*1*	*20*	*1*

Table 3.2: Examples of the user consensus function. Note that the maximum user consensus value is 20.

disagreed with.

$$userConsensus(u) : U \rightarrow \mathbb{R}$$

$$userConsensus(u) = \begin{cases} if\ tagDisagreement(u) = 0 & 0 \\ otherwise & \frac{tagAgreement(u)}{tagDisagreement(u)} \end{cases}$$

Finally, we normalise the user consensus values for each user to enable us to easily compare the consensus values for each user.

$$maximumUserConsensus : \mathbb{R}$$

$$maximumUserConsensus = max(\{userConsensus(u) \mid \exists_{r \in R}\, \exists_{t \in T}\, \exists_{u \in U}\, \exists_{d \in D} : (r, t, u, d) \in (Untaggings \cup Taggings)\})$$

$$normalisedUserConsensus(u) : U \rightarrow [0, 1]$$

$$normalisedUserConsensus(u) = \frac{userConsensus(u)}{maximumUserConsensus}$$

In tables 3.2 and 3.3 we review the results of the user consensus function in a few scenarios, as well as our consideration as to how this function relates to the user profiles developed earlier in this chapter.

3.2.3.2 User Tagging Activity

The user tagging activity sub-component of the trust algorithm allows for us to see how active a particular user has been during a particular time period. We can use this functionality to reward frequent contributors with a greater trust value over those people who do not contribute as much. In addition to this, we can encourage frequent contribution by 'expiring' old taggings[2], and we can help the guru personality type by giving their taggings 'time to breath' by ignoring taggings and untaggings until they reach a certain age.

[2]This should not be confused with deletion however. All taggings are kept for historical purposes.

User Profile	Treated Correctly?	Comments
Guru	No (possibly yes)	Could be unfairly penalised as their taggings may not be obvious to other users. This may lead to other users untagging the tags, reducing the gurus user consensus value. Over time however, the gurus user consensus value should improve as other users begin to understand the information input by the guru.
Hypercritical	No	A hypercritical user can tarnish other users trust values by disagreeing with their tags, however, other users are also likely to untag a hypercritical users tags. This means that a hypercritical user does not increase their own trust, but they can still damage other users trust values.
Consensus Seeker	No	A smart consensus seeker will receive a very high trust value.
Bookmarker	Yes	A bookmarker will find that they receive a large amount of tag disagreement if they don't mark their tags as private. This will result in their trust tending towards zero.
Inactive	Yes	
Common	Yes	

Table 3.3: Algorithm evaluation table for user consensus metric

We have to define a number of equations that we can shortly pull together to calculate the users tagging activity.

withinTimeWindow(d) This function tests whether a timestamp falls within a certain date range. To define the valid date range, we take the current date (hereafter referred to as *now*). We then modify this date with a natural number that represents a number of weeks.

$$earliestAllowableDate, latestAllowableDate : Date$$

$$\sigma \geq \tau$$

$$\sigma, \tau \geq 0$$

$$earliestAllowableDate = now - \sigma$$

$$latestAllowableDate = now - \tau$$

The *withinTimeWindow* function then simply tests whether the given timestamp is within these bounds.

$$withinTimeWindow(d) : Date \rightarrow Boolean$$

$$withinTimeWindow(d) = d \geq earliestAllowableDate \wedge d \leq latestAllowableDate$$

numberOfTags*(u)* This is the total number of tags input into the system by user u, where tags are only used if they were created within the valid time window. There is an equivalent function for *numberOfUntags* where the only difference is that *Untaggings* is used in place of *Taggings*. We define the functions *isValidTag* and *isValidUntag* here, as they will be reused later on also.

$$
\begin{aligned}
& isValidTag : R \times T \times U \times D \rightarrow bool \\
isValidTag(r,t,u,d) &= (r,t,u,d) \in Taggings \wedge withinTimeWindow(d)
\end{aligned}
$$

$$
\begin{aligned}
& isValidUntag : R \times T \times U \times D \rightarrow bool \\
isValidUntag(r,t,u,d) &= (r,t,u,d) \in Untaggings \wedge withinTimeWindow(d)
\end{aligned}
$$

$$
\begin{aligned}
& numberOfTags : U \rightarrow \mathbb{N} \\
numberOfTags(u) &= |\{r \mid \exists_{r \in R}\, \exists_{t \in T}\, \exists_{d \in D} \, : \, isValidTag(r,t,u,d)\}|
\end{aligned}
$$

$$
\begin{aligned}
& numberOfUntags : U \rightarrow \mathbb{N} \\
numberOfUntags(u) &= |\{r \mid \exists_{r \in R}\, \exists_{t \in T}\, \exists_{d \in D} \, : \, isValidUntag(r,t,u,d)\}|
\end{aligned}
$$

averageUserTagCount and averageUserUntagCount *AverageUserTagCount* divides the sum of all tags applied within a certain time window by the number of users who have applied one or more tags within this time window. *AverageUseUntagCount* is the equivalent function for untaggings. Both functions are shown below.

$$
averageUserTagCount : \mathbb{R}
$$

$$
averageUserTagCount = \frac{\sum_{(r,t,u,d) \in Taggings} numberOfTags(u)}{|\{t \mid \exists_{r \in R}\, \exists_{t \in T}\, \exists_{d \in D} \, : \, isValidTag(r,t,u,d)\}|}
$$

$$
averageUserUntagCount : \mathbb{R}
$$

$$
averageUserUntagCount = \frac{\sum_{(r,t,u,d) \in Untaggings} numberOfUntags(u)}{|\{t \mid \exists_{r \in R}\, \exists_{t \in T}\, \exists_{d \in D} \, : \, isValidUntag(r,t,u,d)\}|}
$$

Tagging and Untagging Averages The next two equations compare an individual user to the averages calculated in the last step. This can show us how the user relates to other users. It is preferable for a user to be above one in both of the equations below, as this

indicates that they are more active than most users.

$$userTaggingAverage(u) : U \rightarrow \mathbb{R}$$
$$userTaggingAverage(u) = \frac{numberOfTags(u)}{averageUserTagCount}$$

$$userUntaggingAverage(u) : U \rightarrow \mathbb{R}$$
$$userUntaggingAverage(u) = \frac{numberOfUntags(u)}{averageUserUntagCount}$$

3.2.3.3 Tagging Activity

Pulling together the above functions, we get the *taggingActivity* function which returns a value that, if it exceeds one, represents a person who tends to be more active than normal. Conversely a value of less than one represents a less trust worthy person, as they are less active than normal. Whilst this is not necessarily true, it is a good indicator of a users trustworthiness. As noted in table 3.5, the prototypical user profile that is treated most incorrectly in this instance is the hypercritical, who is likely to be rewarded even though their taggings and untaggings are of less value than those belonging to other user profiles.

$$taggingActivity(u) : U \rightarrow \mathbb{R}$$
$$taggingActivity(u) = userTaggingAverage(u) + userUntaggingAverage(u)$$

Finally, we normalise the tagging activity values for each user to enable us to easily compare the tagging activity for each user. We do this simply by getting the maximum taggingActivity value for all users, and use this to divide the results of each taggingActivity calculation. This means that we get a nice normalised set of values ranging between zero and one.

$$maximumTaggingActivity : \mathbb{R}$$
$$maximumTaggingActivity = max(\{taggingActivity(u) \mid \exists_{r \in R} \exists_{t \in T} \exists_{d \in D} \exists_{u \in U} : (r, t, u, d) \in (Untaggings \cup Taggings)\})$$

$$normalisedTaggingActivity(u) : U \rightarrow [0, 1]$$
$$normalisedTaggingActivity(u) = \begin{cases} if\ maximumTaggingActivity = 0 & 0 \\ otherwise & \frac{taggingActivity(u)}{maximumTaggingActivity} \end{cases}$$

userTaggingAvg	userUntaggingAvg	taggingActivity	normalisedTaggingActivity
0	0	0	0
1	0	1	0.1
10	0	10	1
0	1	1	0.1
0	10	10	1
5	2	7	0.7

Table 3.4: Examples of the tagging activity algorithm.

User Profile	Treated Correctly?	Comments
Guru	Yes	
Hypercritical	No	A hypercritical user will do very well on this equation.
Consensus Seeker	Mostly yes	A consensus seeker will score well in userTaggingAverage, but will not in userUntaggingsAverage.
Bookmarker	Yes	
Inactive	Yes	
Common	Yes	

Table 3.5: Algorithm evaluation table for user tagging activity metric

3.2.4 Algorithm Summary

What we have successfully developed (as validated later in chapter 5) is a function that determines the trustworthiness of a person, not based on their personal relationships with other people, but based solely on the quality of the information they input into the system. This removes any potential abuse of the system to artificially inflate or deflate a persons trust value. In addition to this, our system treats fairly all types of users, as we identified in section 3.2.1.

Where the tagging activity function falters is in its inability to calculate the quality of the tags and untags input by the user. This is fortunately the area in which the user consensus function helps, and thus by merging these two functions we get a good measure of both the activity of a user (which is constantly decreasing due to the reliance on a specific time window) and the agreeability of the user.

We defined the user trust algorithm formally in section 3.2.3, but as a brief reminder, it was formally defined as the following:

$$trust(u) : U \rightarrow [0, 1]$$

$$\alpha, \beta \in [0, 1]$$

$$\alpha + \beta = 1$$

$$trust(u) = \alpha \times normalisedUserConsensus(u) + \beta \times normalisedTaggingActivity(u)$$

We now proceed to use this user trust functionality as a component of the resource distance algorithm, defined in the next section.

3.3 Resource Distance

Using the tagging information that users input into Centruflow, we want to extract as much value as possible. Of course, there is a number of first-order benefits to tagging, discussed in section 2.5.4, but what about second-order benefits? One potentially valuable benefit is inferring relationships between resources where an explicit relationship does not exist. Our hypothesis was that in business environments there exists much implicit information that is not recorded due to various reasons, and ideally there would be a mechanism to record this information within Centruflow, with any inferred knowledge being shown within the Centruflow visualisation. We believe that tagging is an ideal way of attaching small bits of information to a resource, but we could not identify any algorithm to determine relationships between resources. We therefore decided to develop an algorithm to support this.

Our approach is primarily based around the concept of *taggings* and *untaggings*, where a single tag is applied to a single resource by a single user. For each resource we take into account all such taggings and untaggings and calculate relationships to all other resources. Including all this information into a calculation allows for Centruflow to give users a display of 'related resources'.

As with the user trust algorithm, let R, T and U be disjoint sets. R is the set of all resources, T is the set of all 'tag concepts', and U is the set of all users. For the resource distance algorithm, we do not concern ourselves with the timestamp of a tagging, therefore taggings is defined to represent the set $R \times T \times U$.

We define the term *tag concepts* to refer to the potential amalgamation of tags based on the fact that they are assumed to be synonyms. This is done in a preprocessing stage prior to this resource algorithm being run, where essentially each tag is checked to see if it has a 'preferred synonym', and if it does, we simply replace the tag with its preferred tag. This has the effect of increasing the count of the preferred tag, at the cost of completely removing all synonyms that are not preferred. It should be noted that this data is not removed from the database - it remains in the 'raw tags' taggings database table for historical purposes (for example, should the synonyms change). The updated taggings are stored in a 'refined tags' taggings database table, and it is these that are used within Centruflow for all tagging related functionality.

Prior to developing our resource distance algorithm, we briefly introduce the Jaccard distance metric which we loosely based our own metrics on. Following this, we develop a series of functions that, used together, allow us to calculate one measure of the distance between any two resources, based solely on tagging and untagging information.

3.3.1 Metrics

A metric is a mathematical function that defines a distance between any two elements of a set. In creating our algorithms in this section, we were effectively attempting to create suitable metric functions such that we may measure the distance between any two resources. We could then use this distance to determine which relationships to infer and show to the user. In summary, the four requirements for a function to be considered a metric are:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0$ if and only if $x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangular inequality)

In our algorithms below, the first three requirements can be proven trivially, with the fourth requirement proving difficult. To help with this, we researched a number of distance metrics that have been proven, and whilst we couldn't find one that fully met our requirements, we did come across the Jaccard distance metric, which is by far the closest metric to our needs.

3.3.1.1 Jaccard Distance Metric

The 'Jaccard Index', also known as the 'Jaccard similarity coefficient', is a metric used to measure the similarity between two sets of items. It is defined as the size of the intersection of the two sets divided by the size of the union of the two sets:

$$J_S(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

In the case of our calculations, we want to know the distance 'between' two sets, which is defined simply as $\mathit{distance}(A, B) = 1 - \mathit{similarity}(A, B)$. Therefore, to get the Jaccard Distance metric, we simply do the following:

$$J_D(A, B) = 1 - J_S(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

A point of difference between our approach above and the Jaccard distance metric is that instead of taking the cardinality of the number of elements in the intersection and union sets, we instead take the sum of these two sets. This is because then we may sum the trust/tag values, as opposed to simply counting the number of instances. This means that the algorithms defined in the next section can not be claimed to be proven by the Jaccard distance proofs [70, 71], but we believe that these proofs could potentially be adapted to prove that our algorithms are also metrics.

3.3.2 Distance

Using a similar approach to the user trust algorithm, our distance metric pulls together the tag distance and trust distance functions defined in the following sections. We once again use undefined parameters to specify the strength of each of these metric functions (γ, δ), but have not yet specified what values these parameters should have. For now it is assumed that they are both 0.5.

$$\begin{aligned} distance: R \times R &\rightarrow [0,1] \\ \gamma + \delta &= 1 \\ distance(r_1, r_2) &= \gamma \times tagDistance(r_1, r_2) + \delta \times trustDistance(r_1, r_2) \end{aligned}$$

We now proceed to implement the *tagDistance* and *trustDistance* metrics, starting with the *trustDistance* metric below. Whilst these two metrics become increasingly complex, it must be remembered that all we are trying to do is develop a means to measure how related two resources are, and that this is being achieved solely through the comparison of the tags belonging to each resource. If you don't have the stomach for maths, please skip through to chapter 4.

3.3.3 Metric Algorithm Basics

To aid in the explanation of these metrics, we introduce figure 3.2 which shows three resources, which between them have a number of taggings. We can see that there is some amount of 'overlap' between these resources, and hence there is also some amount of similarity. We will refer back to this figure throughout the following sections.

In addition to this, we introduce two functions that return the set of all taggings and all untaggings for a given resource. We do not care what tag or user is associated with the resource, as long as it is a unique tagging/untagging. These two functions are used in our two metrics, hence their introduction here.

$$taggings(r): R \rightarrow 2^{R \times T \times U}$$
$$taggings(r) = \{(t,u) \mid (r,t,u) \in Taggings\}$$

$$untaggings(r): R \rightarrow 2^{R \times T \times U}$$
$$untaggings(r) = \{(t,u) \mid (r,t,u) \in Untaggings\}$$

Based on figure 3.2, we would get the results shown in table 3.6. As there are no untaggings shown in figure 3.2, the result of calling *untaggings*(r) will be the empty set.

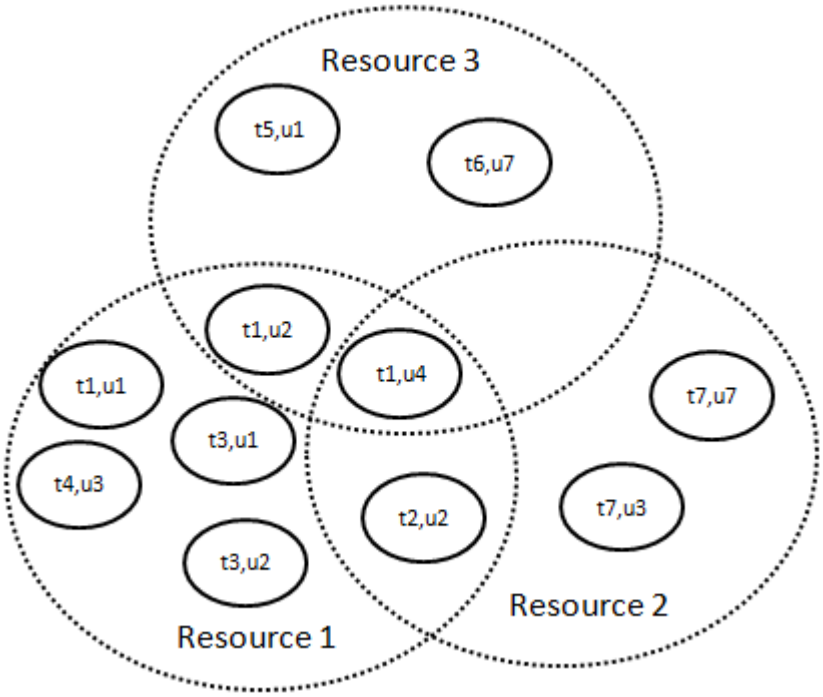


Figure 3.2: An example of three resources with some tagging overlap (but with the timestamps stripped to reduce the number of taggings shown).

	taggings
resource 1	$\{(r_1, t_1, u_1), (r_1, t_1, u_2), (r_1, t_1, u_4), (r_1, t_2, u_2), (r_1, t_3, u_1), (r_1, t_3, u_2), (r_1, t_4, u_3)\}$
resource 2	$\{(r_2, t_1, u_4), (r_2, t_2, u_2), (r_2, t_7, u_3), (r_2, t_7, u_7)\}$
resource 3	$\{(r_3, t_1, u_2), (r_3, t_1, u_4), (r_3, t_5, u_1), (r_3, t_6, u_7)\}$

Table 3.6: The result of calling *taggings(r)* on the resources shown in figure 3.2.

3.3.4 Trust Similarity and Distance

The first metric we implement as part of our resource distance metric makes use of our trust function developed earlier in this chapter. We divide the sum of the trust of all users for each tagging belonging to both resources by the sum of the trust for all taggings belonging to either resource. As a special case, we state that if there are no taggings for $r_1$and r_2, that the *trustSimilarity* is simply one. This makes sense as if two resources both have no resources, then they are completely similar, which means that their distance should be zero.

The *trustSimilarity* function essentially takes the 'overlap' between two resources tag sets and sums the trust value for all given users. It then divides this by the summed trust values for all users who have tagged either resource. This results in a number between 0 and 1, as the denominator is always greater than the numerator.

$$trustSimilarityFraction: R \times R \rightarrow [0,1]$$

$$trustSimilarityFraction(r_1, r_2) = \frac{\sum_{(r,t,u) \in (taggings(r_1) \cap taggings(r_2))} trust(u)}{\sum_{(r,t,u) \in (taggings(r_1) \cup taggings(r_2))} trust(u)}$$

$$trustSimilarity: R \times R \rightarrow [0,1]$$

$$trustSimilarity(r_1, r_2) = \begin{cases} if\ taggings(r_1) = taggings(r_2) = \emptyset & 1 \\ otherwise & trustSimilarityFraction(r_1, r_2) \end{cases}$$

$$trustDistance: R \times R \rightarrow [0,1]$$

$$trustDistance(r_1, r_2) = 1 - trustSimilarity(r_1, r_2)$$

3.3.5 Tag Similarity and Distance

The second and more complex metric that we implement is the tag distance metric. Conceptually the tag distance metric is similar to the trust distance metric above, as they are both based on the Jaccard distance. What follows is a number of functions that build up into the *tagDistance* metric function.

Tag and Untag Count We now define two functions that, given a tag and a resource, will tell us how many people have applied the same tag to the same resource. This makes

	tag 1	tag 2	tag 3	tag 4	tag 5	tag 6	tag 7
resource 1	3	1	2	1	0	0	0
resource 2	1	1	0	0	0	0	2
resource 3	2	0	0	0	1	1	0

Table 3.7: Example *tagCount* based on figure 3.2 and table 3.6.

use of the taggings function defined above.

$$tagCount : R \to \mathbb{N}$$
$$tagCount(t, r) = |\{u \mid \exists_{u \in U} : (r, t, u) \in taggings(r)\}|$$

$$untagCount : R \to \mathbb{N}$$
$$untagCount(t, r) = |\{u \mid \exists_{u \in U} : (r, t, u) \in untaggings(r)\}|$$

Continuing the example shown in figure 3.2, the result of the *tagCount* function is shown in table 3.7. It should be noted that from this point on we dismiss the concept of a user from our calculations, instead we aggregate simply the number of each unique tag belonging to a particular resource, and refer to it as the tagCount(t,r).

Effective Tag Count We treat untags as 'subtractive tags', meaning that the effective tag count for a given tag is a value less than the tagCount. We define the following function to calculate the effective tag count for a tag on a given resource, and we introduce a parameter γ that we can use to tweak the strength of untags. Additionally, we limit the minimum value of the effective tag count to zero, such that we do not end up with a negative number.

$$effectiveTagCount : T \times R \to \mathbb{R}$$
$$\gamma \in [0, 1]$$
$$effectiveTagCount(t, r) = \max(0, tagCount(t, r) - \gamma \times untagCount(t, r))$$

In figure 3.2 we do not have any untaggings, so the effective tag count is unmodified.

Overlap Ratio The overlap ratio calculates the amount of overlap there is between the tag sets of two resources. As an example, imagine two resources a and b, each with the same tag t. If a has 50 taggings of t, and b has 25, then the overlap is $\frac{25}{50}$. In other words, it is the minimum of the two *tagCounts* divided by the maximum value. To make the *overlapRatio* more readable, we have split it into a number of sub-functions below.

$$overlapDenominator : T \times R \times R \rightarrow [0,1]$$
$$overlapDenominator(t, r_1, r_2) = max(effectiveTagCount(t, r_1),\ effectiveTagCount(t, r_2))$$

$$overlapRatioFraction : T \times R \times R \rightarrow [0,1]$$
$$overlapFraction(t, r_1, r_2) = \frac{max(1,\ min(effectiveTagCount(t, r_1),\ effectiveTagCount(t, r_2)))}{overlapDenominator(t, r_1, r_2)}$$

$$overlapRatio : T \times R \times R \rightarrow [0,1]$$
$$overlapRatio(t, r_1, r_2) = \begin{cases} if\ overlapDenominator(t, r_1, r_2) = 0 & 0 \\ otherwise & overlapFraction(r_1, r_2) \end{cases}$$

As an example of the overlap function, take just 'tag 1' from table 3.7. The overlap ratio for resource 1 and resource 3 is then calculated as follows:

$$\begin{aligned} overlapRatio(t, r_1, r_3) &= \frac{max(1,\ min(effectiveTagCount(t, r_1),\ effectiveTagCount(t, r_3)))}{max(effectiveTagCount(t, r_1),\ effectiveTagCount(t, r_3))} \\ &= \frac{max(1,\ min(3, 2))}{max(3, 2)} \\ &= \frac{2}{3} \end{aligned}$$

Tag Sum The tag sum function is used to calculate the number of effective taggings belonging to two resources (r_1, r_2) where the tag (t) is fixed. In addition to summing the effective tag count of the two resources, we also multiply by the overlap ratio, as then we can encourage tag similarity to be greatest when the two resources have a relatively equal number of taggings for the tag t. Should one resource have a significantly different number of taggings of the tag t than the other, we should actually reduce the similarity, and this is what this part of *tagSum* does.

$$tagSum : T \times R \times R \rightarrow \mathbb{R}$$
$$tagSum(t, r_1, r_2) = effectiveTagCount(t, r_1) + effectiveTagCount(t, r_2)$$

$$overlapTagSum : T \times R \times R \rightarrow \mathbb{R}$$
$$overlapTagSum(t, r_1, r_2) = tagSum(t, r_1, r_2) \times overlapRatio(t, r_1, r_2)^2$$

Tag Similarity Pulling together all the functionality above, we get the tag similarity function which can tell us how similar two resources are based on the tags that they have.

$$tagSimilarityFraction: R \times R \rightarrow [0,1]$$

$$tagSimilarityFraction(r1, r2) = \frac{\Sigma_{(r,t,u) \in (taggings(r_1) \cap taggings(r_2))} overlapTagSum(t, r_1, r_2)}{\Sigma_{(r,t,u) \in (taggings(r_1) \cup taggings(r_2))} tagSum(t, r_1, r_2)}$$

$$tagSimilarity: R \times R \rightarrow [0,1]$$

$$tagSimilarity(r_1, r_2) = \begin{cases} if\ taggings(r_1) = taggings(r_2) = \emptyset & 1 \\ otherwise & tagSimilarityFraction(r_1, r_2) \end{cases}$$

Tag Distance The distance between two resources is simply one minus the similarity.

$$tagDistance: R \times R \rightarrow [0,1]$$

$$tagDistance(r_1, r_2) = 1 - tagSimilarity(r_1, r_2)$$

3.3.6 Resource Distance Algorithm Conclusion

The previous sections in this chapter have outlined a distance function that meets all criteria of a metric (as proven in [70, 71]. We provide examples of this algorithm in appendix B.

3.4 Relation Disagreement

Having created an algorithm for inferring relationships, we of course allow for such relationships to be shown within Centruflow. It would be remiss of us however if we did not offer some mechanism to allow for users to disagree with an inferred relationship, as without such a mechanism, there would be no easy means to 'manually correct' the system. This section therefore develops an additional function that we use to take into account the users input that they disagree with an inferred relationship. We have not integrated this function with our distance metric as it is not presently a metric itself, meaning that the distance metric would no longer be a metric either.

Relation disagreement deals with the number of times an *inferred* relationship is disagreed with. The greater this number, the more the distance should be decreased, as the inferred relationship is deemed incorrect. More formally, let R and U be disjoint sets. R is the set of all resources, and U is the set of all users. We define the term *relationshipDisagreement* to represent the subset $R \times R \times U$. A relationship disagreement is between two resources(r1,r2∈R and $r1 \neq r2$) applied by a single user (u∈U).

MaximumDisagreement can be defined as the largest disagreementCount value for all combinations of (r1,r2∈R).

$$maximumDisagreement : \mathbb{N}$$
$$maximumDisagreement = max(\{disagreementCount(r_1, r_2) \mid r_1 \in R \wedge r_2 \in R\})$$

$$disagreementCount(r_1, r_2) : R \times R \to \mathbb{N}$$
$$disagreementCount(r_1, r_2) = |\{u \mid \exists_{u \in U} : (r_1, r_2, u) \in RelationshipDisagreements\}|$$

$$relationDisagreement(r_1, r_2) : R \times R \to [0, 1]$$
$$relationDisagreement(r_1, r_2) = \frac{disagreementCount(r_1, r_2)}{maximumDisagreement}$$

The value calculated by this function therefore is based solely on the number of disagreements input by users. A user may only disagree with each edge once, so this may not be abused by users.

When a company is only bootstrapping its tagging system, this relationDisagreement function is likely to be rather unfair, but over time it should become fairer.

Chapter 4

Software Implementation

As previously discussed, this thesis was focused on the development of algorithms to calculate user trust and the inferred distance between nodes. Having just developed these algorithms in the previous chapter, we were then required to implement these in our programming language of choice, Java.

Once these algorithms were implemented, our focus shifted to the extension of the Centruflow framework, as well as the development of a Centruflow Server. In addition to this, there was of course the tagging architecture to implement, which spans both client and server. The remainder of this chapter discusses the implementation details at a relatively high-level.

4.1 Implementation of Similarity and Trust Functions

With the mathematical algorithms developed in chapter 3, it was time to implement these in code. Our plan was to embed this functionality inside the Centruflow Server, as our primary goal was to quickly enable a Centruflow client to find out which nodes have the strongest inferred relationships with other nodes. The two main approaches put forth as to how these functions could be developed were brute-force calculation and on-demand calculation, which are discussed below.

4.1.1 Brute-Force Computation

The brute-force approach simply suggests that all trust and resource distance (similarity) functions operate at a set schedule - perhaps once a day at 3:00am. All possible calculations should be run at this time, with the calculated data exported to a relational database. This would allow for queries from the Centruflow client related to user trust and inferred

relationships to be handled by simple SQL queries to a relational database (and cached in memory within the Centruflow Server).

This is only a good approach for small tagging systems, as the downside of such a solution is its lack of scalability. Centruflow is primarily aimed at enterprises, and there is no cap on the number of potential users (and hence tags). In particular, there are companies wanting to expose their Centruflow installations over the Internet, which could potentially lead to an explosion of users and taggings. We do not believe that this brute-force approach can handle these scenarios, and as such we would only recommend our tagging system to small enterprises. To enable universal use of the tagging system, an alternative approach is needed, such as that explained in the next section.

4.1.2 On-Demand Calculation

The on-demand solution attempts to allow for more 'up to date' trust and resource distance values by calculating the necessary information only when relevant. This does not mean however that this approach does not perform any caching/pre-computation - it is simply that the data expires more frequently and is not calculated in a batch process that runs nightly.

There is however a problem with this approach also: the resource distance algorithm makes use of the user trust calculation as part of its calculation, meaning that to calculate resource distance, the server must know the trust values for all users who have tagged the resources in question. Depending on the number of users who have tagged a particular resource, this could be a limiting factor in ensuring a good response time.

One way around this issue is to pre-compute the user trust values as part of a nightly batch process, and then to calculate resource distances on-demand. This approach helps when considering that our ultimate goal is to respond to the Centruflow client request in a minimal amount of time.

4.1.3 Our Approach

We chose to design our software such that we could easily switch implementations should a better idea be suggested, but for the purpose of this thesis, we implemented the brute-force implementation. This choice was made primarily due to its simplicity of implementation, but as noted in section 6.3, we need to more thoroughly plan and implement an on-demand approach in the future. In addition, implementing the weaker brute-force approach had no influence on any of our results, as both approaches would be expected to produce the same result for any given query.

In terms of implementation, our approach would not cause a 'nightly calculation downtime', as all calculations would be performed in memory or in a temporary database, with only the last action being to overwrite all data in the production refined tags database table. This would limit the possibility of users not retrieving useful tagging data, but to be fair, does expose a time window where such a problem will be possibly encountered by users.

4.2 Tagging Architecture

The tagging architecture spans both the client and the server, as the client allows for the input and visualisation of tags, and it is up to the server to handle storage and querying of tags. As discussed in section 2.5.3, there is an overwhelming number of ways to approach the tagging issue, and for a long time much consideration was spent on deciding the correct approach to take. Our architecture was by no means unique, it consisted of the need to apply a tagging t to a resource r, by one user u. In addition to this, we were interested in recording a small amount of metadata, such as the timestamp of the tagging, the template the tagging was applied to[1], and whether the tagging was public or private. In the future, additional metadata may be deemed valuable and be collected.

4.2.1 Taggings Relational Database

All tagging information is stored in a simple relational database, consisting of seven main tables, and a number of tables with lesser importance. There are no relationships between the tables, which considerably simplifies the queries (in terms of the required SQL, and more importantly the complexity of the data operations performed within the database). We will very briefly detail each of the most important tables below.

4.2.1.1 Raw Tags Table

The raw tags database table stores every tag ever input by a user, including when a tag has been untagged by users. The only exception is when the user has deleted a tag that they themselves have created. The raw tags table records the users ID, the resources URI, the tag, and the important metadata such as the template ID and the timestamp of creation.

4.2.1.2 Refined Tags Table

The refined tags database table is where all tags progress should they be deemed valuable. This table is only temporary however, as it is completely emptied as part of the resource

[1] In Centruflow, each template represents a different visualisation, and a separate graph of data.

distance algorithm. The means through which tags progress from the raw tags database to the refined tags database is detailed in section 4.2.3. This table is somewhat simpler than the raw tags table, as it does not bother to record timestamp metadata, as it is not yet useful to have this information available to the Centruflow client, however this may change in the future.

4.2.1.3 Untaggings Table

All untaggings are recorded in this table, in much the same fashion as the raw tags table.

4.2.1.4 Tag Synonyms Table

This records a mapping of words to their synonyms. This informs the resource distance algorithm that should it come across a word that is not the preferred synonym, that it should change it to the preferred synonym. This happens as part of the transformation process when raw tags are transformed into refined tags. More information on this process can be found in section 4.2.3.

4.2.1.5 Relation Disagreements Table

This table is used to hold any disagreement information submitted by users related to inferred relationships between nodes. This information is used as part of the resource distance algorithm.

4.2.1.6 Resource Distances Table

This table holds the precomputed distances between resources. This enables quick query times for the Centruflow client.

4.2.1.7 User Trust

This table simply records the trust of each user, on a per-template basis[2]. Trust is recorded as a value between zero and one, where zero is completely untrustworthy, and one completely trustworthy.

[2]As noted earlier, a template is simply a collection of XML descriptors that allow for the decoration and advanced visualisation tools offered by Centruflow for one particular data source/graph.

4.2.2 Tags as OWL Classes

In implementing our tagging architecture, and as briefly discussed in section 2.5.3.2, we treated each unique tag string in our system as if it were its own class. This is somewhat similar to the approach proposed in other projects [72]. This meant that, for example, the tag 'Actor' would be an OWL class, and as such could exist in a formal taxonomy (as opposed to the folksonomies discussed earlier). The benefit of having tags exist within a formal taxonomy is that then we may infer increased knowledge by using other tags in the taxonomy. Examples include:

- Making use of all super classes (i.e. classes from which a tag class inherits - and in OWL multiple inheritance is allowed). What is meant by this is that, given a tag class, we can find all 'more-generic' terms. An example may be the tag 'Dog', which may have the super classes 'canine' and 'mammal'. By using this hierarchical information, we enrich our data with further metadata, facilitating the realisation of further inferred relationships.
- Retrieving all equivalent classes for a given tag (i.e. synonyms). In other words, our ontology can define equivalent terms, for example the terms 'theatre', 'movie theatre' and 'cinema' all represent the same concept, but different people will choose to use different terms. If there isn't a means to suggest synonyms, the implicit relationships that exist with these words will not be understood by the algorithms.
- Measuring distances between tag classes based on the number of edges between classes in an OWL ontology.

The important issue to resolve in this situation is to choose which ontology language to use that is expressive enough to allow for the representation of tags. At the same time, we do not simply want to choose OWL-Full, as this provides us with too much functionality, and the possibility of undecidability if misused. In our case, we want to declare inheritance and equivalence. Inheritance is a feature of the lowest ontology language RDF-Schema, but equivalence is introduced in OWL-Lite. For this reason, we chose to use OWL-Lite for representing tags.

It should be noted however that despite tags being represented as OWL classes, it is not necessary that they belong in a formal taxonomy - this just provides additional benefits. If no OWL ontology is provided, then it is assumed that each tag class is completely unrelated to any other tag class, and so by default no additional inferencing can be performed.

4.2.2.1 Untaggings and OWL classes

In our current implementation of untagging, we do not treat untags as their own class. In fact (as noted earlier), untaggings are treated merely as subtractive tags, meaning that they

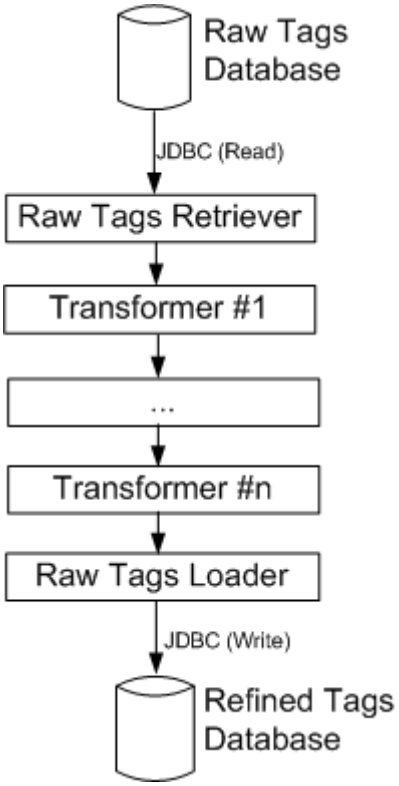


Figure 4.1: The tag transformation pipeline.

simply reduce the count for the given tag. We do this as it is a computationally simpler approach than the alternatives, which allows for our brute-force algorithm implementation to proceed more rapidly.

An alternative approach is to create untags as classes also, and to add these to the resource class with 'disjointWith' statements. This creates a more realistic representation of the user perspective of the taggings applied to the resources, and provides the ability for more reasoning to be applied on a higher level (i.e. developed in SWRL, for example). This approach requires considerable more research into the area of representing negative information, which was discussed earlier in section 2.1.7.

4.2.3 The Tagging Pipeline

Whenever a user inputs a tag into their Centruflow client, it is submitted via a RESTful protocol (see sections 2.4 and 4.3.4) to the Centruflow Server, where it is immediately inserted into the raw taggings database table. The raw taggings table holds every tag ever input by users (apart from those which have been deleted by their owner). We have developed a transformation pipeline prior to the running of the resource distance algorithm to 'clean up' the raw tags database. The end result of this transformation is the refined tags database. This is shown in figure 4.1.

The transformation pipeline is a series of chained iterators that operate on the output of the previous iterator. In the case of the first iterator, it retrieves the data from the raw tags database. We have implemented a few 'transformers', as they have come to be known, but the architecture has been left open such that future requirements can easily include more (or less) of these transformers. The transformers have the ability to add and remove information from the iterator (which means that they change the data going to the refined tags table - they do not modify the raw tags table). Some of the transformers developed so far include:

1. A user trust transformer, which removes the tags of users who are deemed untrustworthy (i.e. their trust value is less than a predetermined value), and duplicates the tags of those users who are deemed overly trustworthy. This enables Centruflow to encourage users to use the tags believed to be most valuable, and hide from users the tags deemed to be least valuable.

2. A tag count transformer, which requires there to be a minimum number of taggings for a particular tag in the raw tags database before it is allowed to graduate into the refined tags database. This encourages only the most popular tags, and filters out tags that are likely to be personal or subjective, and not factual, as discussed in section 2.5.6.

3. A tag synonym transformer. As mentioned above, we have a table that administrators can load with synonyms that they wish to replace with a preferred synonym. This enables the system to minimise the folksonomy such that the suggestions that users are presented with are of the highest quality. This transformer replaces any synonym with its preferred synonym. We achieve this by using the OWL equivalentClass feature, and as mentioned in section 4.2.2, this is the reason why we chose to use OWL-Lite.

4. A tag inferencing transformer. This plugin loads an OWL ontology which describes a hierarchy of terms. Each tag string is treated as if it were an OWL class, and should an OWL class exist in the provided ontology, we can infer that other terms in this ontology should also belong in the refined tags database. This transformer helps to make connections between tags, strengthening the folksonomy.

4.2.3.1 Weakness Of The Tagging Pipeline

Unfortunately, there is one serious weakness to using such an architecture, and that is related to the user experience of Centruflow. The problem is that a user expects to instantly see their tags appear when they add them to a resource within Centruflow. Of course, there is a problem with this, as any tag added to the system will only be shown to the user when it graduates to the refined tags table (and even then, there is absolutely no guarantee that

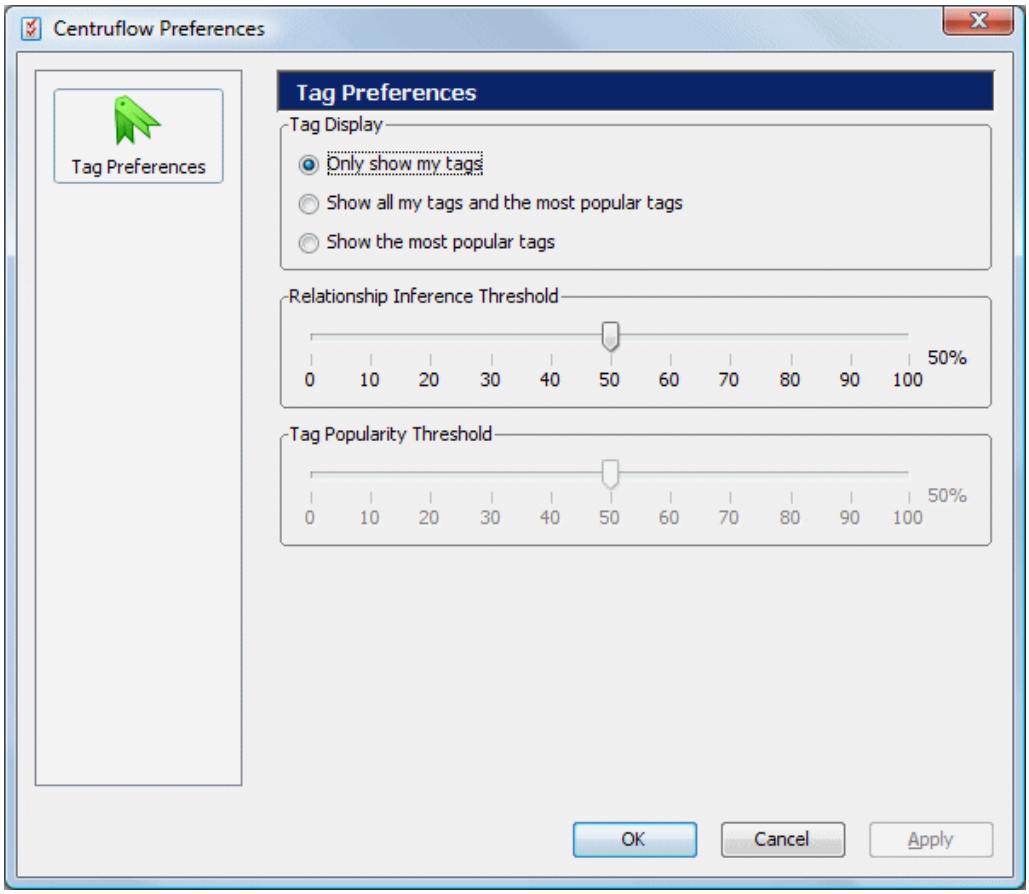


Figure 4.2: The tag preferences dialog.

the tagging will forever remain in the refined tags table). This means that, if Centruflow acted properly, the user may never see the tags that they have input into Centruflow!

In implementing the tagging system, we considered this problem and decided the best approach would be to offer to the user a configuration dialog that would enable them to specify how they wish to see the tags. This dialog is shown in figure 4.2. The three options the user has are for the user to just see their tags, to just see the most popular tags, or to see both. Based on this selection, the tagging plugin can choose to request information from either the raw taggings table (specific to one user), from the refined taggings table, or from both.

4.2.4 Querying The Tagging System

The Centruflow Server provides a number of access options to the tags database, all of which are exposed through RESTful functions[3]. What this means is that all queries and updates to the database are handled using the HTTP protocol, and in particular its GET and POST methods. Almost all responses for GET queries are encoded as RSS 2.0 formatted

[3]There is more information on REST and the Centruflow Server in section 4.3.4.

XML[4], meaning that queries can be subscribed to in RSS feed readers. The benefit of this is that people do not need to be constantly running Centruflow to receive information relevant to them. It is hoped that in the near future these RSS feeds will provide links to users to instantly load their Centruflow application and go directly to the information outlined in the RSS feed.

4.3 Centruflow Server

Whilst Centruflow is a client-side application, it necessarily requires some form of server-side component to enable easy access to relevant data sources. In removing Thinkmap's visualisation toolkit, we also had to remove the Thinkmap Server component. Replacing Thinkmap's visualisation toolkit was necessary to enable our goal of visually supporting tagging functions. Unfortunately, these two Thinkmap components are tightly bound to one-another, and being proprietary meant that it was not overly feasible to build a visualisation that used the Thinkmap Server. This meant that to replace the Thinkmap visualisation, we also needed to develop a Centruflow Server component that would act as an adapter onto data sources.

The role of the Thinkmap Server is to receive requests from the Thinkmap client, and to provide the client with information pertaining to the visualisation. This provided a layer of caching above the database server, and may have simplified the coding required within the Thinkmap client (we can not be certain due to it being closed source). The Thinkmap server was a simple server, leaving the majority of the 'heavy lifting' to the administrator, as it was necessary to develop a complex configuration file for each data source. It was our intention to attempt to minimise this headache in our Centruflow Server.

Prior to this software being developed, considerable research was necessary to understand the problems that businesses face, and what technological solutions presently exist to help achieve these requirements.

4.3.1 Requirements of a Centruflow Server

A key distinction from the outset of this thesis was the focus on 'data sources', and not only 'databases' (i.e. relational databases). We thought that companies would be keen to be able to easily visualise all their information within Centruflow, including their databases, LDAP directories, Intranet documents (Microsoft Word and Excel files), XML files, RDF files, etc. After meeting with companies, it turned out that instead they had an alternative feature that they would like to see: the ability to easily browse interlinked data that exists

[4]There are however one or two GET queries that simply return a plain text response, where this is appropriate.

within multiple databases, all within one visualisation. We added this request to the list of desired functionality.

What follows is some of the core requirements that we found would be necessary for a Centruflow Server product. These requirements were elicited from companies that Centruflow deals with, from our research of the current state of the market, and our own personal feelings having been involved in the development of previous versions of Centruflow.

4.3.1.1 Graph-like Data Retrieval

On a fundamental level, a Centruflow Server needed to offer a means to access graph-like data. This necessitated a means to query the server, which would then in turn handle the querying of the underlying data sources. Following the W3C standards at the time, SPARQL was the most attractive query language, given that it was inherently graph-based. In addition to this, there really was no alternative to SPARQL that made sense (and this is still the case now).

4.3.1.2 Server Management

Another major requirement was the ability to easily control the server through a web interface, as opposed to the more common approach of servers requiring manual editing of configuration files. Functionality that was highly desired included:

1. The ability to easily add and remove data sources.
2. The ability to 'browse' data sources within the web browser (not necessarily using a Centruflow visualisation to achieve this).
3. The ability to link together data sources to create combined graphs.

By the time this thesis was complete, points one and two were both implemented, with the final point being delayed until further development could be performed.

4.3.2 Alternative Software

Prior to developing this server, we set about researching the competitive landscape for such a product. This was necessary as there was no point creating this server if a better server application existed that could perform the necessary functions. As should be clear, a server component was not within the key business plans of Centruflow Ltd. Finding an alternative server application to handle this task would relieve the small development team

from having to develop such a complex piece of software, allowing them to instead focus on the Centruflow client.

Of course, an alternative point of view is to begin to argue that the development of a powerful server such as the one proposed in this thesis could instead become a new revenue stream for Centruflow Ltd, with or without the added value of the Centruflow client. Such a point of view has yet to be discussed within Centruflow Ltd, and as such the Centruflow Server is simply perceived as a necessary component for Centruflow client installations.

OpenLink Universal Server [73], and its open source version, OpenLink Virtuoso, are middleware applications capable of connecting to a number of different data sources. It is developed in C++, and appears to be actively developed (although as of October 2007, the last release was a year ago). It supports a large number of query protocols, including our preferred query language of SPARQL. It appears to be the closest match to what the Centruflow Server proposes to be, except that the OpenLink software is far more comprehensive, supporting a far wider array of functionality than is intended for the medium-term future of Centruflow Server. In this respect, the OpenLink software goes against the 'lightweight' ambitions of the Centruflow Server. Finally, and no doubt due to being so comprehensive, the OpenLink software is prohibitively expensive, insofar that Centruflow would need to double its cost to integrate such a solution. Integrating the open source version of OpenLink would not be an option either, given the requirement that Centruflow Server be a privately owned application.

Aduna Software provide a commercial and open source version of their Auto-Focus product [74]. This product provides a client and server side component, where the server component acts as an information 'crawler'. It effectively scans all data and metadata that it is permitted to see, allowing for the Auto-Focus client to visualise the relationships between information in much the same fashion as Centruflow does presently. The key difference between the Aduna Software product and the proposed Centruflow Server is that the Aduna Software Auto-Focus server is primarily focused on crawling data on the file system, as opposed to providing access to data sources such as relational databases, which is the main goal of the Centruflow Server. The Auto-Focus server uses the Aperture library [75], which is an open source framework for "extracting and querying full-text content and metadata from various information systems (e.g. file systems, web sites, mail boxes) and the file formats (e.g. documents, images) occurring in these systems". This unfortunately does not include databases. Despite this, Auto-Focus server and/or Aperture are interesting projects that may be used in future versions of the Centruflow Server.

Finally, Thetus Publisher, developed by Thetus Corporation [76] is a system that appears to follow the same approach as the software above, but it also has much more focus on the client-side visualisation of data sources. Unfortunately, their website does a very poor job at communicating the main goals of the application, and we failed in our attempts at communicating with the company to find out more about their software. Regardless, their

software does deserve a mention, as it does appear to quite closely resemble the goals of the Centruflow Server.

None of this software matches our requirements fully, with the biggest obstacles being cost and complexity. These software applications are designed to be run on powerful servers with much memory and processing speed. In addition, to require such software to operate Centruflow would price Centruflow out of the market, which is something the company management want to avoid. Therefore, a case exists for a simple and lightweight server component to be developed. This will initially be specifically for Centruflow 3.0 and later releases, but may gain traction as a separate product in the long-term.

Based on this research, there was a niche appearing for a software product that would be a lightweight middleware server that could handle the necessary querying and virtual merging of data sources. All other technical solutions on the market were far too 'heavyweight', in that their feature set (and hence their price) was far too considerable to be used as a sub-component to the Centruflow solution. We did not require these features, and so we decided to investigate the development of our own server, with three clear goals:

1. Keep it simple and lightweight.
2. Use open source components where feasible (ignoring components that were GPL'ed however).
3. Use open standards where feasible.

The purpose of the Centruflow Server was to replace the Thinkmap Server used in Centruflow versions 1.x and 2.x, which was the server-side component used to adapt a database to be visualised within Centruflow. How Thinkmap implemented this functionality is unknown, as their system is proprietary, but it in general receives requests from a Thinkmap client, and converts these requests to SQL before submitting them to the database. This is the same approach taken in the development of the Centruflow Server, but instead we used semantic web technologies such as RDF, OWL and SPARQL. It should be noted that the Centruflow Server did not exist prior to this masters thesis at all, and as such was designed from the ground up to work specifically for the needs of this research.

The justification for using these semantic web technologies is that they are standards (or soon to be in the case of SPARQL). This ensures increased interoperability with other systems in the future, assuming that other software applications come to standardise on these standards (which is obviously very apprehensive at this early stage of the semantic web lifecycle [77]).

The server is split into two functional subunits, those being graph-based data handling and non-graph-based data handling. Each unit is handled through a separate server interface (i.e. accessed using different URLs), and will be discussed separately below.

4.3.3 Handling Graph-Based Data

Graph-based data is the lifeblood of a Centruflow visualisation, and as such providing this functionality was the main goal of Centruflow Server. Unsurprisingly, most data inside companies is stored in relational databases, where relationships are declared through the use of primary and foreign keys. Despite this, it would be rather short-sighted to only consider relational databases as the sole repository for graph-based data, with other examples including LDAP directories, RDF files, SPARQL endpoints, and other proprietary systems.

4.3.3.1 High-Level Architecture

The graph-based aspect of the Centruflow Server application is a combination of Java servlet and Java Server Pages (JSP), backed with a considerable semantic web framework (making use of the Jena semantic web framework [78]). It exposes each distinct data source as a distinct SPARQL endpoint, allowing for any data source to be queried using the SPARQL query language. Whenever a SPARQL query is received by an endpoint, Centruflow Server converts it into the appropriate SQL queries[5], that can then be directly submitted to the necessary databases. When information is returned back from the database, it is converted into an RDF graph prior to being returned to the Centruflow client.

In this way, all data sources are treated as RDF graphs with a SPARQL endpoint. What this means is that the Centruflow Client can now talk to the Centruflow Server using the SPARQL query language, as opposed to the Centruflow Client using the proprietary Thinkmap protocol (and the necessary database duplication). This bodes well for future connectivity to systems that begin to expose data as RDF. The architecture for this aspect of the server is shown in figure 4.3. The diagram on the left shows the logical architecture, and the diagram on the right the physical architecture. In particular, the diagram on the right correctly shows that each data source adapter has its own SPARQL endpoint that forwards the SPARQL query directly to the data source adapter. This means that each Centruflow client query is specifically targeted at one SPARQL endpoint.

4.3.3.2 Implementation-Level Architecture

As noted earlier, we wanted to build the Centruflow server using open standards and open source components, and as shown in figure 4.4, we largely succeeded. The Centruflow Server product encompasses the entire top rectangle, but only a small portion is custom

[5]Whilst in theory the Centruflow Server can query any data source, we have only implemented support for databases at this stage. This was because businesses expressed little interest in connecting other data sources. Instead, as mentioned in section 4.3.1, other requirements were deemed more important.

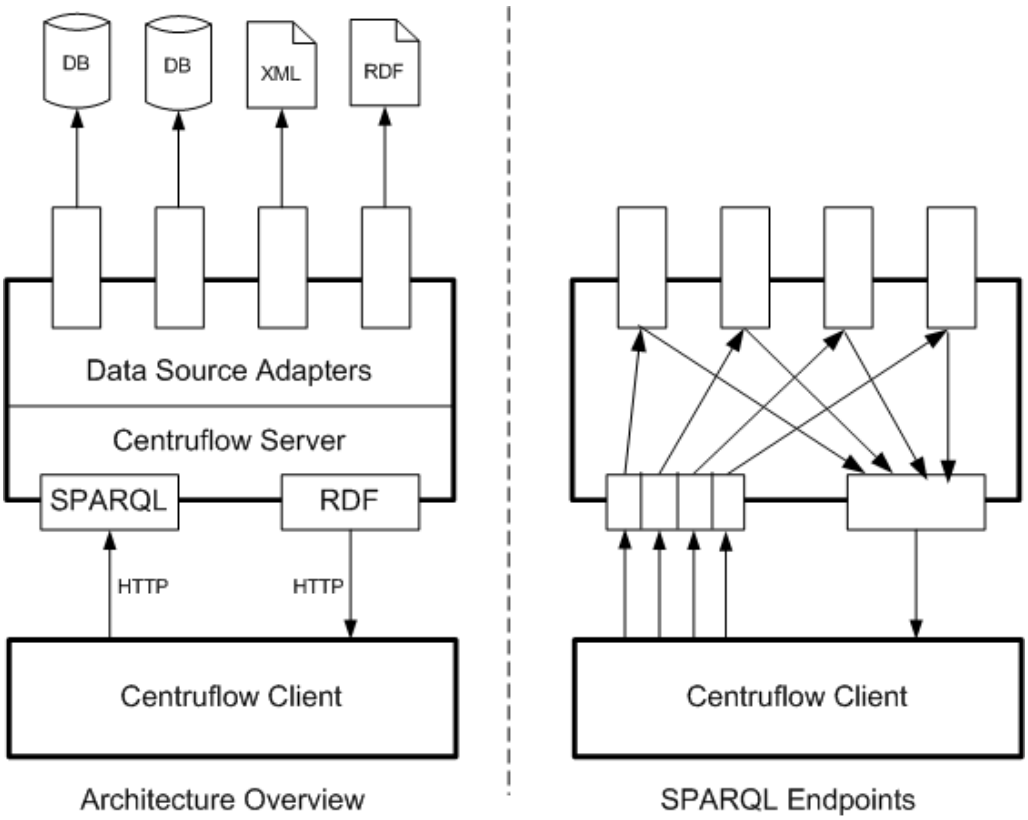


Figure 4.3: The Centruflow Server SPARQL Architecture.

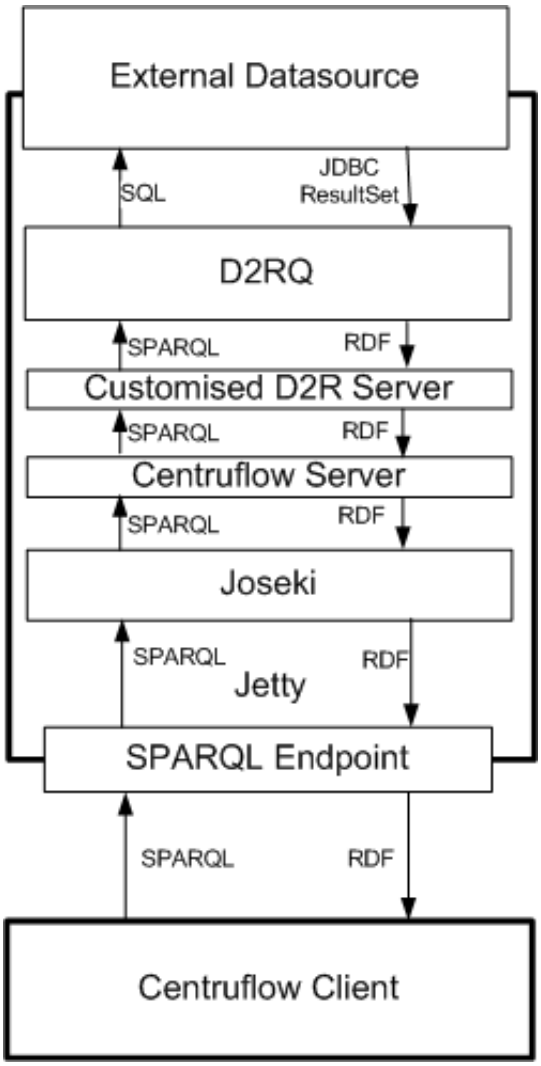


Figure 4.4: The Centruflow Server implementation using open standards and open source components.

code, with the vast majority of code belonging to open source projects. There are only two rectangles inside the bold Centruflow Server rectangle that we even touched with our own code, those of course being the Centruflow Server box, as well as the D2R Server box. Most of our effort was spent in configuring these software components to work appropriately, and providing the necessary 'glue' code between them.

To put this into context, Centruflow Server requires 35 Java 'JAR' files, totaling 13.9MB. The custom Centruflow Server code is itself a separate JAR file, that weighs in at 0.44MB. This is the reason why, in the acknowledgements at the start of this thesis, we say we owe a great deal of gratitude to the open source developers of the world.

What follows is a quick overview of what the other components shown in figure 4.4 are used for.

Jetty

Jetty [79] is an "open-source, standards-based, full-featured web server implemented entirely in Java". Jetty is used to provide the necessary HTTP protocol handling to allow for incoming and outgoing communications. In addition, Jetty acts as a fully capable Java servlet container, meaning that it can compile Java Server Pages (JSP) and servlets. We use Jetty to deploy Joseki.

Joseki

Joseki is deployed as a web application within the Jetty server and provides the ability to handle incoming SPARQL queries by accessing pluggable data sources. In our case, we have adapted Joseki to instead pass all SPARQL queries on to the D2R Server.

D2R Server

D2R Server is an application developed by Dr Christian Bizer which makes use of D2RQ to translate SPARQL queries into SQL, and in addition D2R Server provides a SPARQL endpoint that allows for SPARQL queries to be received. This is accomplished by pulling together Joseki and Jetty. We have adapted D2R Server considerably, to allow it to handle multiple data sources at once, as opposed to its original architecture of only supporting one data source.

Importantly, we have embedded the D2R Server within our Centruflow Server, despite it being released under the GPL. We were able to receive a commercial contract from Dr Christian Bizer that allowed for Centruflow Server to not be treated as a GPL'ed application. This also applies to D2RQ discussed below.

D2RQ

D2RQ is a library, also developed by Dr Christian Bizer, that translates SPARQL into SQL. It is no longer developed, but is still immensely useful in the context of Centruflow Server. As quoted from [80]:

> "As Semantic Web technologies are getting mature, there is a growing need for RDF applications to access the content of non-RDF, legacy databases without having to replicate the whole database into RDF. D2RQ is a declarative language to describe mappings between relational database schemata and OWL/RDFS ontologies. The mappings allow RDF applications to access the content of huge, non-RDF databases using Semantic Web query languages like SPARQL."

4.3.4 Handling Non-Graph-Based Data

To treat all data as an RDF graph does however make life difficult in some circumstances, in particular when the data is not graph-based or where there is a need for a more complex two-way communication between client and server. For this reason, we decided not to place all our eggs in one basket, and to instead offer an alternative means of retrieving data from the Centruflow Server. The primary driver for this requirement was the tagging system, as we needed a means for taggings to be submitted to the server, and for the server to send various information to the client related to taggings. Integrating this requirement into the SPARQL queries was feasible, but represented considerable overhead (both in terms of development and CPU time for each query). In addition to this, it was imagined that in the future there would need to be more functionality where communication would be two-way, and not graph-based (for example, allowing users to modify information from Centruflow client).

As discussed in section 2.4, in the area of two-way client-server communication, there are a number of technical possibilities as to how this may be implemented. Two popular approaches these days are using SOAP and REST. For the purposes of developing this server functionality, it was determined that REST was the better option because of its low overheads and ease of extension. REST essentially works by using standard HTTP requests to submit information to the server (primarily using the GET and POST protocols). In RESTful software approaches, the accepted approach is to use GET for retrieving information, and POST for modifying information. This is the approach followed in Centruflow.

To more succinctly explain the REST architecture of our server, we have created figure 4.5. In essence, we have created a servlet that receives all incoming REST queries, and then passes control to the relevant REST function based on the information it may parse

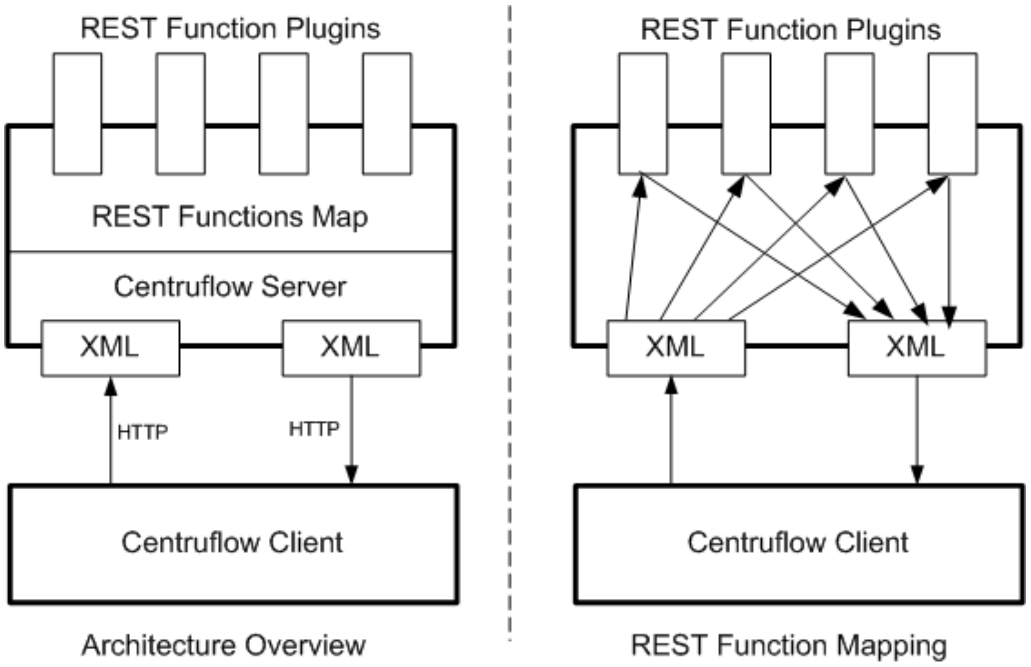


Figure 4.5: The Centruflow Server REST Architecture.

from the URL, using a defined syntax for communications between client and server. The diagram on the left shows the logical architecture, and the diagram on the right the physical architecture. In particular, the diagram on the right correctly shows that each REST function is its own plugin, with the server determining which plugin to call based on the URL received.

4.3.5 Features Overview

There are a few nice features of the Centruflow Server that have not been detailed, so this section will briefly highlight these features.

4.3.5.1 Database Installation Tool

With the Centruflow Server, it is possible in many circumstances to simply fill in and submit a form to have the Centruflow Server discover the database, analyse it and create all necessary configuration files. The server then only needs to be rebooted to make available all the data contained in the database. There is still plenty of improvements that can be done in this area, for example, making it easy for administrators to configure which database tables and columns should be made available, and how to link tables where a foreign key does not exist.

4.3.5.2 RDF Dump Tool

Users can dump a database out into an RDF file such that it may then be visualised by the standalone version of Centruflow or shared with people who do not have access to a Centruflow Server. Once again this could be improved in the future by allowing users to specify which data they wish to export to the RDF file.

4.4 Centruflow Client

Development of version 3.0 of the Centruflow Client was rather involved, requiring a number of different layers of focus. These included:

- An efficient means of retrieving data from the Centruflow Server.
- An efficient means to translate retrieved data into the internal data structures used by Centruflow (which are visualisation independent).
- Developing additional layers on 'top' of the Centruflow client through the development of plugins related to user tagging. This development effort was sped up through the use of Centruflows plugin architecture, the result of our research in 2006 on software plugin architectures [27].

The remainder of this section will now proceed to discuss the implementation details at a high-level.

4.4.1 Important Centruflow Concepts

There are two concepts that are very important within Centruflow, and to help in the understanding of the remainder of this section, will be discussed now. As mentioned earlier, Centruflow 2.0 was developed to help abstract away the underlying visualisation technology such that it could be more easily replaced. Two important concepts were developed during this time, those being the concept of a 'CentruflowEntity' and the 'EntityManager'.

The CentruflowEntity interface is essentially a wrapper around an underlying graph entity (node or edge), and it therefore provides a consistent interface that may be used throughout Centruflow. It wraps all important functionality without exposing any implementation details, making it an ideal abstraction for any underlying visualisation. Centruflow is hugely dependent on the CentruflowEntity, so it was paramount that any RDF data could be easily transformed into CentruflowEntity objects prior to the rest of Centruflow having to deal with any data. In Centruflow 2.0 the CentruflowEntity wrapped five distinct entity

types belonging to Thinkmap, and in Centruflow 3.0 stores all information from one RDF entity. We did have to extend the CentruflowEntity interface as part of Centruflow 3.0, but this was simply to improve functionality.

The EntityManager interface is the manager of all CentruflowEntity's within Centruflow, and should be implemented by the visualisation engine. The EntityManager can be communicated with by any other part of Centruflow, and provides a consistent interface to the underlying visualisation engine. This EntityManager interface allows for other aspects of Centruflow to get data such as the current 'centre' node, all the selected nodes, a node with a particular distinct identifier, and various other aspects of functionality. Once again, this interface was extended in Centruflow 3.0 to offer more functionality to the rest of Centruflow.

4.4.2 Data Retrieval Using SPARQL

We use SPARQL to query the Centruflow Server which in turn queries our remote data sources. The data returned is an RDF result set represented in XML. To reduce our development time we used Jena [78], a Java-based semantic web framework, to connect to Centruflow Server, send our SPARQL queries, and to handle the parsing of the responses. This left us with three key areas of focus:

1. Crafting relevant SPARQL queries to request the required information.
2. Transforming the RDF result sets into useful CentruflowEntity's.
3. Efficiently performing our queries without creating too much overhead on either the client or the server.

After making sense of SPARQL, we developed the query shown in listing 4.1. This query returns all information about a single URI (which represents a single resource), as well as details of all other resources that the requested resource connects to. It is still used by Centruflow to query remote data sources via the Centruflow Server, but its days are limited, as this query retrieves all information related to the node, even if it is not desired by Centruflow immediately (e.g. it is only shown if the user selects a 'show details' button). It makes sense to only retrieve this information on demand, as it would considerably lighten the amount of data being retrieved for each query. It is our belief that by doing this the Centruflow client will start up considerably faster, and be considerably more responsive, as currently there is an obvious slowdown as the initial set of data is being downloaded and parsed.

```
SELECT ?property ?hasValue ?isValueOf
```

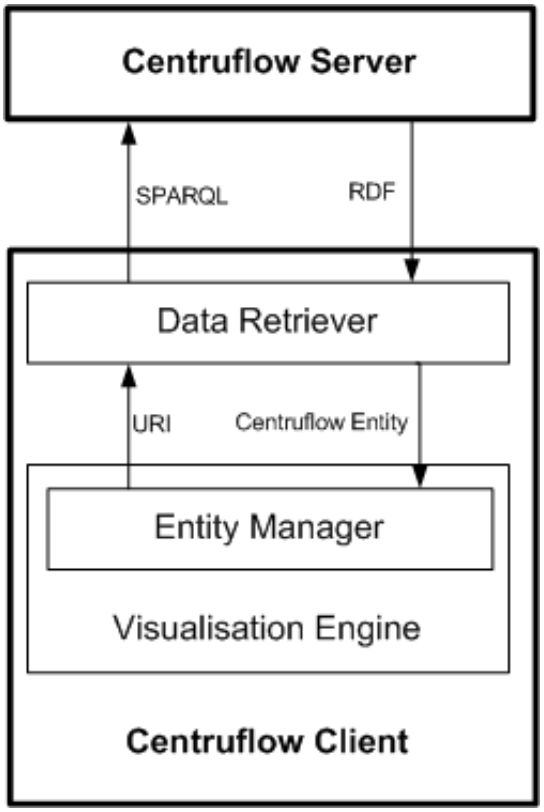


Figure 4.6: How the Centruflow client and server communicate.

```
WHERE {
    { <uri> ?property ?hasValue } UNION { ?isValueOf ?property <uri> }
}
```

Listing 4.1: The default SPARQL query used by Centruflow to request all information for one specific URI.

The architecture that we finally developed is shown in figure 4.6. We once again integrated the entity manager with the visualisation engine, and made it such that the visualisation controls the entity manager (e.g. based on user interaction). When the visualisation needs more data, it asks the entity manager to either return it from its cache, or retrieve it from the Centruflow Server. Therefore, the visualisation has no concept of SPARQL or semantic web technologies - it simply submits URIs for which it wants to visualise, and receives CentruflowEntity objects.

4.4.3 Data Translation

Directly after receiving the graph data from Centruflow Server, it is stored internally as RDF triples. Whilst this would have been an ideal means to store such data, Centruflow has a legacy which required for the data to be represented by the CentruflowEntity interface. This interface, as well as the EntityManager interface, was introduced in the Centruflow 2.0

release based on an improved understanding of the underlying technologies from building Centruflow 1.0. Initially it was considered to simply provide an adapter such that internally the triples be used, but this turned out to be relatively slow due to the way in which Jena works.

The alternative approach, which is the one we finally took, was to simply strip all relevant data from the triples as they are retrieved, and manually build CentruflowEntity objects. This approach works well, as it reduces the query time when Centruflow wants to retrieve data from the entity, and also slightly reduces memory requirements as all data retrieved from the server had a number of superfluous triples.

4.4.4 Data Integration

There are two important aspects to retrieving data from the Centruflow Server to show within a Centruflow client, these being:

1. Retrieving required information as close to 'Just In Time' as possible.
2. Minimising the memory requirements once the data is retrieved (including expiring old data).

4.4.4.1 Just In Time Querying

Centruflow is user-driven, in the same fashion as the world wide web (and pretty much every application ever made). It makes no sense for the application (be it Centruflow or the web browser) to begin to retrieve too much data 'off in the distance', as it becomes less likely that the user will actively browse to this information. On the other hand, pre-caching information prior to the user requesting it improves the perceived performance of the application.

In developing Centruflow 3.0, it took us a considerable amount of time to develop an approach that worked well, however we still believe that there is scope in the future to more properly handle this (see section 6.3). What we eventually settled on was requesting two times the depth setting that the user can see. In other words, if the user has the depth set to two (meaning that from the 'centre node' they can see at most all nodes that have two or less edges between them and the centre node), then we will download all nodes to a depth of four. Whenever the nodes on screen change (due to a user interaction), the appropriate data will be queried for. This provides for a good middle-ground between responsiveness and memory requirements.

4.4.4.2 Memory Management

Whilst the trivial data sets we used to develop Centruflow 3.0 with are useful, and act in a similar fashion to real data sets, customers have inquired about the feasibility of visualising considerably larger data sets (consisting of more than 800,000 nodes). It is this kind of situation that proper memory management becomes an issue.

Our approach to representing data in Centruflow 3.0 continues to revolve around the concept of a *CentruflowEntity*, which contains all information about a single resource. Data retrieved from the Centruflow Server is immediately converted into a CentruflowEntity, and given to the visualisation system to manage. This means that within Centruflow there is one authoritative repository for in-memory resources.

There are two problems that have not yet been addressed, which are:

1. All data about a single resource is requested, even if the user can not immediately see or use it. This links back to improved 'just in time' querying as mentioned above.
2. No data is expired after it does not seem valuable to the user anymore, meaning that continued browsing increases the memory requirements of Centruflow with no chance of the memory usage decreasing.

These two problems will be addressed further in section 6.3 as future work.

4.4.5 Search

For most users, a search component embedded within Centruflow is absolutely critical. In previous versions of Centruflow, search was handled by the Thinkmap visualisation components, and so for this reason alone, a replacement was needed for Centruflow 3.0. What was developed was a search user interface, shown in figure 4.7, as well as an extension-point[6] for search providers to interface with the search user interface. Two such search providers were then developed:

1. A datasource searcher, where the data source is queried using a SPARQL query. The query used is shown in listing 4.2.
2. A taggings searcher, which searches for taggings that have the desired search text contained within them.

[6] An extension-point is basically a slot for extensions to insert additional functionality. Centruflow consists entirely of plugins, where a plugin is simply zero or more extension points and zero or more extensions.

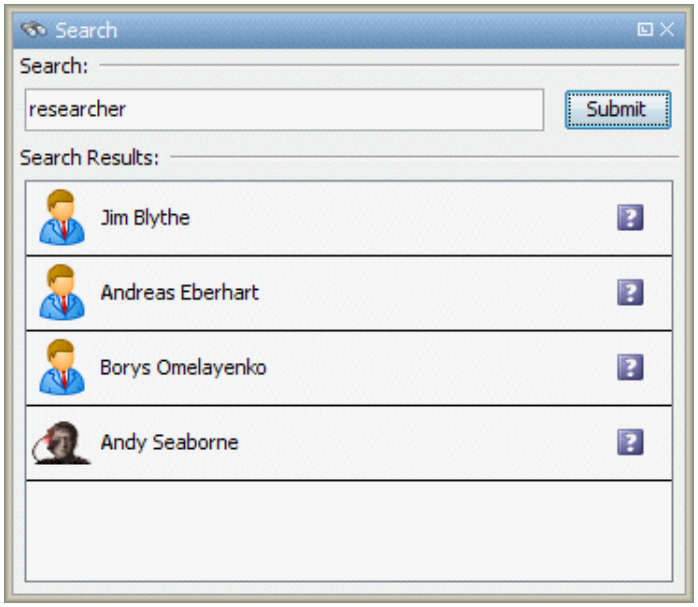


Figure 4.7: The search dialog.

```
prefix fn:<http://www.w3.org/2005/xpath-functions#>
SELECT DISTINCT ?subject ?predicate ?property
WHERE {
  ?subject ?predicate ?property .
  FILTER fn:contains(fn:upper-case(str(?property)),fn:upper-case("<search- ↩
      token-1>") .
  FILTER fn:contains(fn:upper-case(str(?property)),fn:upper-case("<search- ↩
      token-2>") .
  ...
  FILTER fn:contains(fn:upper-case(str(?property)),fn:upper-case("<search- ↩
      token-N>") .
}
```

Listing 4.2: An example of the SPARQL query used to search the Centruflow Server. The <search-token-n> fields should be replaced with a single search paramter.

4.4.6 Tagging User Interface

Built as a traditional plugin for Centruflow 3.0, the tagging functionality provides users with a number of features related to user tagging. These are detailed in the next few sections of this report.

4.4.6.1 Main Tags Dialog

The main tags dialog went through a number of user interface implementations prior to the current design. Shown in figure 4.8 is the user interface as of late September 2007. It

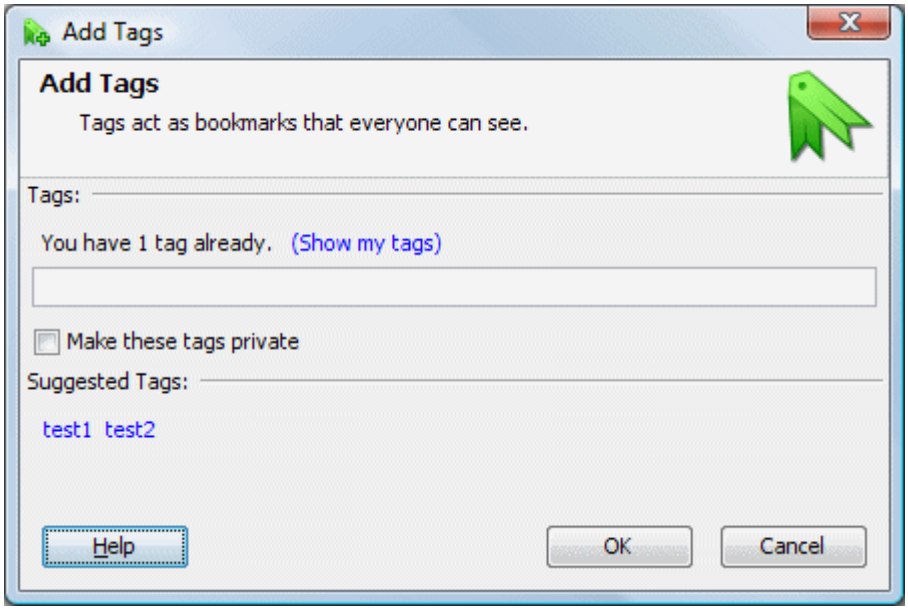


Figure 4.8: The add tags dialog.

provides a lot of functionality, some of which is not clearly visible. This functionality is outlined below.

Adding Tags The user may enter new tags by typing into the text field. Internally, these tags are tokenised based on where the user inserts commas or spaces (as tags can not contain these characters).

Tag Suggestion There are two forms of tag suggestion:

1. Node-based Popular Tags: This is when the user first loads the dialog, and is presented with a list of hyperlinked tags at the bottom of the dialog. These tags are the most popular tags belonging to the selected node, that have not already been added by the user to this particular node. Clicking any of these tags will have the effect of automatically adding them to the text field (the tag will be added only once, unless the user clears the text field).
2. Text-based Popular Tags: As the user types, a popup appears directly beneath their cursor, suggesting tags that start with the text that have already been typed by the user. The user may select these tags, and upon doing so, will see the tag text added to the text field.

Creating Private Tags As discussed in section 2.5.6, users may make tags private to prevent other users (and the trust and resource distance algorithms) from seeing them. Checking the checkbox in the dialog will force all tags entered into the text field to be

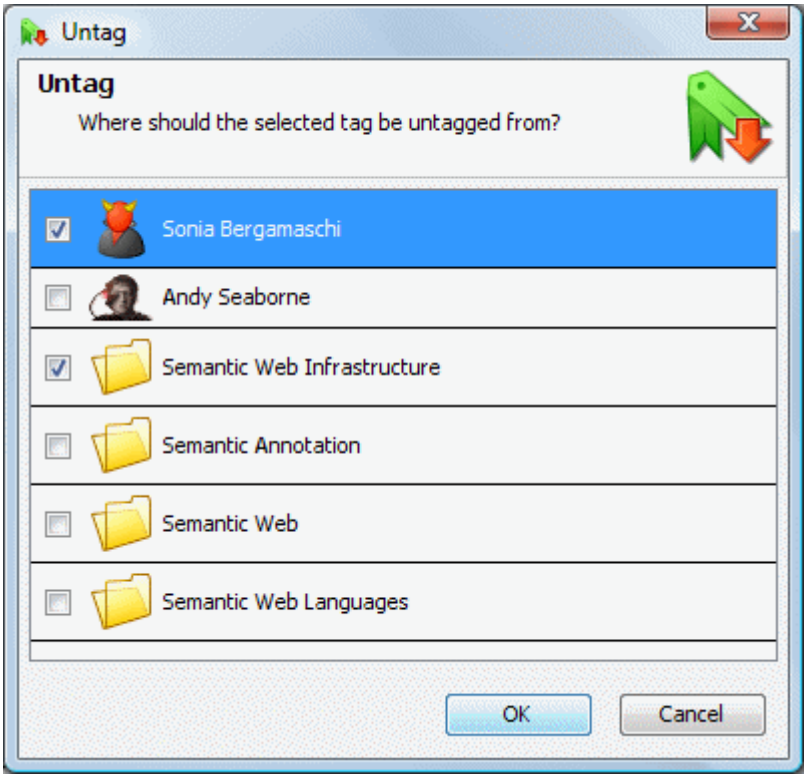


Figure 4.9: The dialog shown to users when they select a tag they wish to untag. A similar dialog is shown when a user disagrees or wishes to delete a tag.

marked as private. Unfortunately, there is not yet any means to be more granular than this, to specify individual tags as either public or private, but this is something that will be resolved at some point in the future. The main hurdle in attempting this is to conceive of a user interaction that did not overly complicate the dialog or confuse the user.

Deleting Tags It is possible for a user to delete their own tags from a resource from within this dialog by simply selecting the 'Show my tags' link, and then editing them in the text box. Clicking 'OK' then has the effect of adding and deleting tags as necessary.

4.4.6.2 Untagging, Deleting and Agreeing With Tags

When users select a tag inside the visualisation, what they are really selecting is the set of all taggings where the tag is equal to the selected tag. This means that when a user wishes to untag, delete or agree with a tag, they must specify for which resources they want this action to occur on. This functionality is shown in figure 4.9 for untagging. This dialog lets the user select the resources for which they wish to apply the desired action to (in this case untagging, but equivalent dialogs exist for deleting and agreeing with tags).

4.4.6.3 Tag Preferences

The tag preferences dialog is the area for users to change persistent settings for the tagging plugin. It was deemed the least important of the new functionality to actually implement, and as it turned out, there was unfortunately not enough time to finalise this piece of functionality. The only section implemented is the top-most, that is, the section pertaining to users controlling what tags they want to see.

The controls shown in figure 4.2 allow the user to change what tags they see, how many they see, and also how many inferred relationships to show. Each of these sections is discussed below.

Tag Display As has been discussed in section 2.5.6, there is a refining process before tags are actually able to be seen by Centruflow. This means that in reality, a tagging added by a user should not be instantly visible within Centruflow, and in fact may never be visible[7]. This is somewhat of a tradeoff, as we do not want all taggings, regardless of their quality, to be visible in the visualisation, but nor do we want to hide users tags from themselves, especially as tags can act as a users personal bookmarks also. For this reason, we allow the user to make an informed decision on which tags they see, as shown in the first box in figure 4.2. The three options a user is given is to only see their tags, to see all their tags and the most popular tags, or just the most popular tags. In almost all cases, it is anticipated that the user will choose either the first or second options, but we leave that choice to the user, providing them with the default choice being option two.

Relationship Inference Threshold The relationship inference threshold slider allows the user to specify how many inferred relationships to display (assuming they have the data layer visible - see the next section). The way this slider controls the number of inferred relationships is simply by sending to the Centruflow Server the percentage of inferred relationships it is interested in receiving. The Centruflow Server will then only return the required percentage, ranked in order of distance, such that the closest inferred relationships are to be returned.

Tag Popularity Threshold The tag popularity threshold works in much the same way as the relationship inference threshold described above: it limits the number of tags displayed to the user such that only the desired percentage of total tags are returned.

4.4.6.4 Data Layers

Data layers is a new concept that was introduced in Centruflow 3.0 after we went to see a particular Governmental department and observed how they use their information systems.

[7] Visible in the sense that the tag is visible on screen, but also that it appears in searches.

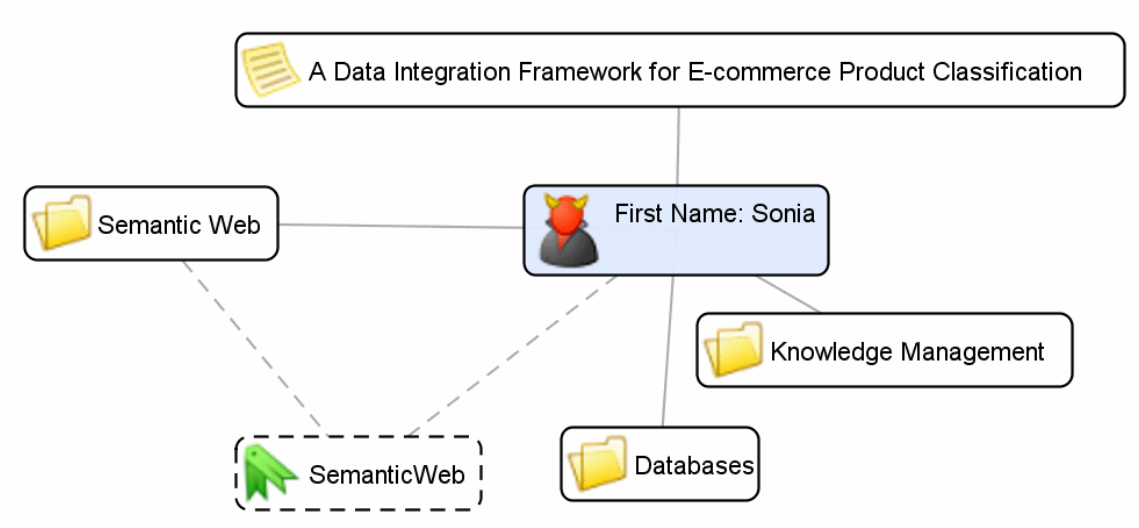


Figure 4.10: A tag as it is shown to the user inside Centruflow 3.0. It has the text 'SemanticWeb', and has been applied to the 'Semantic Web' topic, and to a person called 'Sonia'.

In one particular system, they had huge amounts of information, and the only way they could use it was to look at subsets of the information at any one time. This closely resembles the functionality presently offered in Centruflow called filters, where node types (computer, person,etc) can be shown or hidden, but is slightly different as data layers could consist of a number of different node types and/or be inferred (i.e. not actual data).

In the long term, we developed the concept of data layers under this premise, but it was actually a short term need that prompted this feature to be developed. The particular need was based on the desire to somehow show to the user two things:

1. How tags relate to nodes.
2. How nodes are inferred to be related, based on our resource distance algorithm.

In other words, it was through data layers that we made available tagging and inferred relationship information to users. Figures 4.10 and 4.11 show the result of enabling these data layers. Note that in both cases, the dashed lines or boxes represent the information contained within that data layer. In general, the 'nodes' represent a node coming from the database, and the 'edges' are the links between the data in the database. Clicking on a node shows the connections coming from the node. In this way, users can navigate the data in their database graphically and more intuitively.

4.5 Centruflow Standalone Version

As has been discussed, Centruflow follows the traditional client/server software model, where all data is stored on a remote server, and is merely downloaded by the client on an

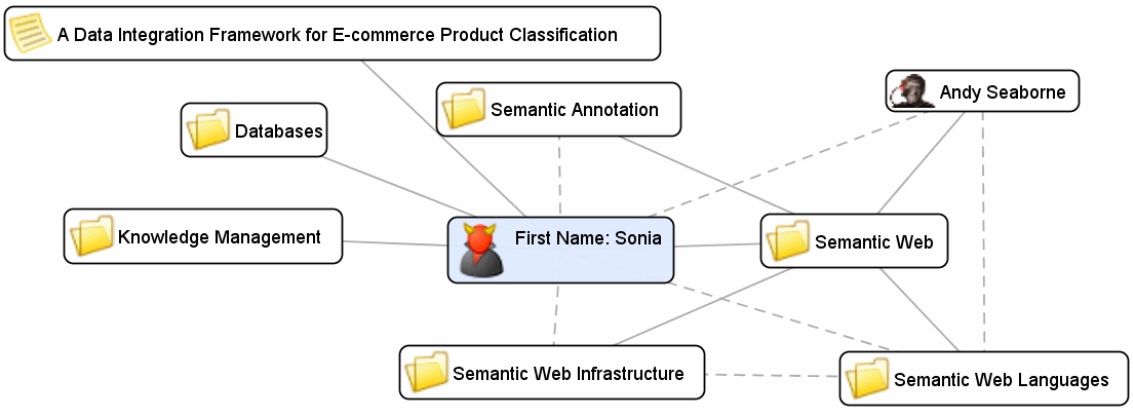


Figure 4.11: A number of inferred relationships shown inside Centruflow 3.0 (all dashed lines are inferred). The inferred relationships are the visible result of the algorithms developed in chapter 3.

'as-needed' basis. One of the findings from our research with customers and consultants using Centruflow was that there would be immense value in being able to model information within Centruflow when consultants are out in customer engagements. This recommendation comes despite the availability of tools such as Microsoft's Visio. The main reasoning behind this is that Centruflow allows for information to be more easily understood due to the ability to filter, cluster, and browse around information.

We identified two possible versions of a standalone client: a free viewer version and an 'editor' version. The free viewer version could be downloaded by anyone from the Centruflow website, and could be used to visualise RDF graphs (such as those generated by Centruflow). The 'editor' version provided all functionality of the free viewer version, but also allowed for users to modify the data by creating, editing, and deleting resources. This data could be saved as RDF and shared with other interested parties.

Because this customer requirement was not discovered until August 2007, our initial software architecture had been developed and implemented. This meant additional research had to be undertaken to investigate the means through which a standalone Centruflow client could be developed and integrated without sacrificing the integrity of the Centruflow architecture. We believe that we have found a solution that continues to maintain the cleanliness of the Centruflow architecture, whilst also providing end-users with a powerful application that meets their requirements.

Our approach is to simply separate the 'remote' client/server functionality out into a new plugin, and to then create a new 'standalone' plugin to handle the above system. The major difference is that the 'remote' plugin will send the SPARQL query to a Centruflow Server instance, whereas the 'standalone' plugin will send the same SPARQL query to a local, in-memory graph that has been built by parsing the RDF file.

Whilst this project is not within the scope of this thesis, it is notable that it is only because of the research performed this year in semantic web technologies that we are able

to relatively quickly and easily develop such a major product variation. Had we been using Centruflow 2.0, our approach would have been fraught with proprietary technologies belonging to Thinkmap, and had licensing issues surrounding a free version of our software.

4.6 Centruflow Boot Loader

One major weakness of Centruflow in recent versions (versions 2.0 and 3.0) has been the fact that it is a standalone application, in a time where most people are becoming more familiar with the web 2.0 style of application. This has led to some 'force-back' from the customers of Centruflow, wishing that they may get a web-based version. The primary reason for this is simply due to user convenience, and to be frank, buzz-word bingo.

Whilst we have discussed a long-term solution of developing a web-based Centruflow release based on one of the popular rich Internet application frameworks, we have also developed a short-term solution that considerably eases the deployment and initialisation of Centruflow to employees within a company. This short-term solution is interesting in how it improves the ease of deployment and installation for users. Whilst this solution was not in the scope per se, we feel it was a valuable development that we made this year to improve Centruflow, as it makes Centruflow far more readily usable. The remainder of this section outlines the implementation details of our short-term solution to addressing this important customer concern. We do not bother to discuss the long-term plan, due to the uncertainties associated with it.

4.6.1 Short-Term Solution

Java Web Start [81] is a technology developed by Sun Microsystems that is distributed with the Java Runtime Environment (JRE). It allows for standalone Java applications to be loaded over the web by users clicking on hyperlinks within web pages. We saw this as a means of performing the following:

- Controlling the initial boot process of Centruflow, including:
  - Validating user licenses.
  - Installing Centruflow.
  - Starting an already installed Centruflow.
- Keeping Centruflow plugin files up to date with those stored on the companies Centruflow Server.
- Communicating with an already running instance of Centruflow.

– Passing in parameters into Centruflow.

To enable these requirements, we developed a small application with a Swing-based frontend. It essentially works as a boot loader for Centruflow, following a rather complex set of actions in a well-prescribed process. The following section outlines each of the key actions mentioned above.

4.6.1.1 Installing Centruflow

Up until the development of this boot loader, installation of Centruflow was a manual task consisting of unzipping a zip file. There were numerous flaws with this, as we could not easily control which plugins the customer was using (and whether they had paid for them), and we could not easily and automatically maintain an employees installation such that they are always using the most recent releases of the plugin files.

For an employee who has never run Centruflow before, we developed the boot loader software to automatically scan in common installation positions, and if Centruflow was not found, present the dialog shown in figure 4.12. This dialog allows the user to either install Centruflow, or specify the location as to where Centruflow is installed. If the user specifies a valid location, this location is remembered for future boots so that they are not repeatedly pestered by this dialog. Regardless of which action the user selects, the next action is for the boot loader to communicate with the Centruflow Server to retrieve details on the versions of all available software (including plugins) on the server. These versions are then compared to the local versions. Any necessary changes are made to the local installation (up to a full installation of Centruflow on the local computer).

4.6.1.2 Starting an Installed Centruflow

Prior to showing the installation dialog, the boot loader attempts to find an already installed instance of Centruflow. If it is found, the user is never presented with the dialog shown in figure 4.12, and instead Centruflow is directly started. Prior to this starting however, we can perform the same plugin version check, to once again enable users to remain up to date.

4.6.1.3 Communicating with a Running Centruflow Instance

The other important use case is when there is already an instance of Centruflow running when a user clicks on a hyperlink to start up Centruflow. In this circumstance it would be preferable for the boot loader to check if there is already a Centruflow instance running,

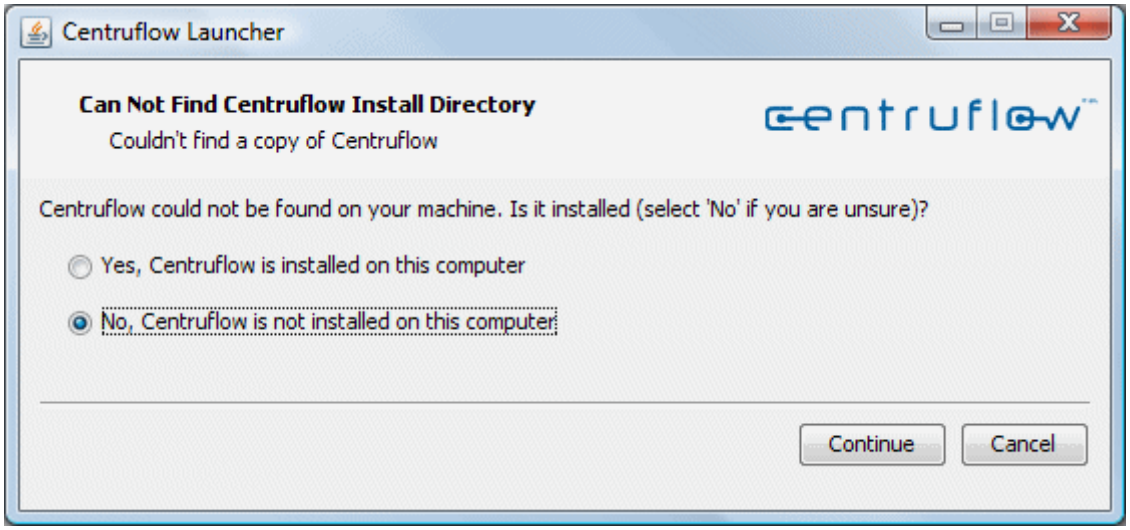


Figure 4.12: Centruflow Boot Loader Dialog shown to users who are not presently running Centruflow, and who do not already have an already installed Centruflow client.

and if so, pass any special parameters along to the Centruflow instance. This operation is similar in nature to how web browsers operate, where a new browser tab is opened if a browser is already running.

Users wishing to start Centruflow via Java Web Start are already browsing the web, and in many circumstances, they have identified some information external to Centruflow that they wish to browse within Centruflow. To allow for this, we don't use a static Java Web Start file (a JNLP file), but instead we dynamically generate this using a Java Server Page (JSP) that acts as if it were a JNLP file. We can then embed the query string given to the JSP into the JNLP, which is downloaded by Java and given to the boot loader. The boot loader can then send this to the Centruflow client so that the client may act appropriately.

We achieved this by embedding a small socket-based server into the Centruflow client that can be communicated with by the boot loader. This allows the boot loader to ask on a specific port if Centruflow is running, and if so, it can pass along the parameters to the Centruflow client. The parameters passed along are the query string provided as part of the Java Web Start URL.

Chapter 5

Validation

5.1 Validation Process

Having developed and implemented algorithms to enable user trust and resource distance calculations, it was important to develop a simple scenario that would validate how well our algorithms work. Our approach was to create a number of user profiles (as outlined in section 3.2.1), and proceed to use them within Centruflow to create tags we believed these users would create. We could then manually run our algorithms using this data and ascertain whether the data is appropriately understood based on the results.

For this test we disabled the time-delay functionality of our algorithm, which filtered out all taggings that did not occur within a certain time window. For this reason, all taggings and untaggings were used to calculate the user trust and inferred distance values. This would not impact the algorithm, and we decided to do this for two reasons:

1. We as yet do not have a definite time window that we believe is appropriate.
2. Disabling the time window functionality allowed us to use Centruflow to input tags, as there was no need to manually edit dates to enable them to be within the relevant time window. This allowed us to test and improve the tagging functionality developed as part of this thesis.

To create a worst-case validation scenario we disabled some functionality within our algorithms, including:

- We chose not to test our relationship disagreement functionality (detailed in section 3.4). What this means is that we did not allow any of our users to disagree with inferred relationships by right-clicking on an inferred relationship and selecting the

User Profile	Number of users
Guru	1
Hypercritical	1
Consensus Seeker	2
Bookmarker	1
Inactive	1
Common	2

Table 5.1: User groups and the number of users in each

disagree option. The reasoning for this is that disagreeing with inferred relationships is a means of correcting incorrect relationships, whilst we were interested in the results where no such correction was provided.

- We disabled the tagging pipeline, so we did not have any transformations such as an OWL ontology to offer additional inferred tags or user trust filtering. This meant that all raw tags were kept and treated as refined tags, meaning no tags were added or removed.

Creating Users

We created a small number of user profiles based on the prototypical users outlined in section 3.2.1. We determined the number of users to create for each user group based on our judgement of the likelihood of each of the user groups. The numbers of each user profile are shown in table 5.1.

What this table shows is that we believe that most users will be consensus seekers (i.e. they will tend to agree with pre-existing tags) or they will be common users (i.e. they will vary, having attributes of all the other prototypical user profiles). Having decided on this distribution of users, we proceeded to create these accounts within Centruflow, ready for the next step.

Adding Taggings And Untaggings

For each user above (there were eight in total), we would log into Centruflow and proceed to act as if we were them. This included adding taggings and a varying number of untaggings. The data we used for the graph was based on the International Semantic Web Conference graph that we used as a test database. This graph provided us with the ability to have gurus using semantic web knowledge that would seem unintelligible to other users. In addition to this, it allowed for all other user profiles to be easily used.

Table 5.2 shows the taggings that were input into the taggings database table by the users. Similarly, table 5.3 shows the untaggings input into the untaggings database table. The resource ID field is simply a unique identifier given to each resource.

Resource ID	Tag	User ID	Is Public
conferences/23541	Conference	common-user-1	Yes
conferences/23541	ISWC	common-user-1	Yes
papers/4	SquishQL	hypercritical	Yes
persons/10	SemanticWeb	common-user-1	Yes
persons/11	F.Guerra	bookmarker	No
persons/2	V.Ratnakar	bookmarker	No
persons/5	B.Omelayenko	bookmarker	No
persons/6	HP-labs	common-user-1	Yes
persons/6	Jena	common-user-1	Yes
persons/6	HP-labs	guru	Yes
persons/6	SemanticWeb	guru	Yes
topics/10	DB	common-user-1	Yes
topics/10	DB	common-user-2	Yes
topics/10	RDBMS	common-user-2	Yes
topics/10	DB	consensus-seeker-1	Yes
topics/10	DB	hypercritical	Yes
topics/11	SemanticWeb	common-user-2	Yes
topics/11	SemanticWeb	consensus-seeker-1	Yes
topics/11	Internet	consensus-seeker-2	Yes
topics/11	OWL	consensus-seeker-2	Yes
topics/11	RDF	consensus-seeker-2	Yes
topics/11	SPARQL	consensus-seeker-2	Yes
topics/11	SWRL	consensus-seeker-2	Yes
topics/11	OWL	guru	Yes
topics/11	RDF	guru	Yes
topics/11	SemanticWeb	guru	Yes
topics/11	SPARQL	guru	Yes
topics/11	SWRL	guru	Yes
topics/11	Internet	hypercritical	Yes
topics/15	KM	common-user-1	Yes
topics/4	SemanticWeb	common-user-2	Yes
topics/4	SemanticWeb	consensus-seeker-1	Yes
topics/4	SemanticWeb	guru	Yes
topics/5	SemanticWeb	common-user-2	Yes
topics/5	SemanticWeb	consensus-seeker-1	Yes
topics/5	SemanticWeb	guru	Yes
topics/6	SemanticWeb	common-user-2	Yes
topics/6	SemanticWeb	consensus-seeker-1	Yes
topics/6	SemanticWeb	guru	Yes

Table 5.2: Tags input by our prototypical users.

Resource ID	Tag	User ID
papers/4	common-user-2	SquishQL
persons/6	hypercritical	SemanticWeb
topics/11	hypercritical	OWL
topics/11	hypercritical	RDF
topics/11	hypercritical	SWRL

Table 5.3: Untags input by our prototypical users.

Username	Trust Value
guru	0.679
hypercritical	0.875
consensus-seeker-1	0.667
consensus-seeker-2	0.367
bookmarker	0.500 (By default)
inactive	0.500 (By default)
common-user-1	0.733
common-user-2	0.800

Table 5.4: The calculated trust values for our prototypical users.

5.2 Validation Results

User Trust Calculations

We were pleased with the results of our user trust validation calculations, as they reflected trust values that we believed were representative of our prototypical user profiles. Table 5.4 shows the final trust values calculated for our prototypical user accounts. What this table shows is that based on the taggings and untaggings input in this validation run, we see that the most trustworthy user is actually the hypercritical. It is likely that this is the case because the amount of tag agreement for the terms that the hypercritical user untagged was low, leading the algorithm to place little value in these taggings. Following this is the common-user-2 user, who we should note was the only other user to input an untagging, which explains his increased trust value. Common-user-1 was next, with the likely explanation being that the taggings that were used were not untagged, which placed him above the guru. The guru was next, but as mentioned, he suffered from the untaggings placed against him. Both the consensus seekers fall to the bottom (ignoring the bookmarker and inactive users for now), and particularly consensus-seeker-2. We believe that this is once again due to the influence of the hypercritical user, as the untaggings act to cancel out the taggings input by the consensus-seeker-2 user.

We have two points that need to be raised:

1. We believe that over time the influence of the hypercritical will drop off considerably, as more people add additional tags and agree with the tags already in the system. This will lend itself to all user types apart from the hypercritical.

Resource ID 1	Resource ID 2	Distance
topics/11	topics/6	0.0648
topics/11	topics/5	0.0648
topics/11	topics/4	0.0648
persons/10	topics/4	0.0693
persons/10	topics/6	0.0693
persons/10	topics/5	0.0693
persons/6	topics/4	0.523
persons/6	topics/6	0.523
persons/6	topics/5	0.523
persons/6	persons/10	0.559
topics/11	persons/10	0.707
persons/6	topics/11	0.777

Table 5.5: Results from resource distance algorithm (distances closer to zero represent more highly related resources).

2. Both the inactive user and the bookmarker are invisible to the algorithms, as the inactive has input no tags, and the bookmarker has only input private tags. Should the bookmarker input some public tags now, only these tags will be used to calculate that users trust value.

Inferred Distance Calculations

With the taggings and untaggings input through Centruflow, 12 inferred relationships were calculated where the distance value was less than one (i.e. there was some relationship). These results are shown in table 5.5. What we found was once again encouraging - resources were, we believe, appropriately related. In testing the results of this algorithm within Centruflow (by enabling the inferred relationships data layer) we found that the relationships presented could be perceived as helpful to the user.

5.3 Thoughts

We found that overall our approach works very well, with trust values being calculated appropriately considering the user profile, and the subsequent benefits available to the user are considerable. In particular, we found the inferred relationships, as well as the visualisation of popular tags, seemed to add value to the visualisation. This was backed up by other users who were aware of the research and Centruflow.

Two important points to remember about these results are:

1. This was a worst-case test scenario - we removed our tagging pipeline and disabled access to the relationship disagreement function.

2. We validated using our default parameters that were defined in our algorithm. These calibration parameters are designed to fine tune the system, but require considerably more real-world usage before we can confidently adjust them. In particular, we do not know to what extent users will use the untagging functionality.

Based on the results of this validation, we feel confident enough in our technology to make it publicly available as part of the next release of Centruflow.

5.4 Stress and Performance Test

We performed a number of stress and performance tests as part of the testing of our algorithm. We found that our brute-force algorithm varies quite dramatically in speed of calculation based on not just changes to the the number of taggings, but also to the number of URI's, users and tags used.

To test our brute-force algorithm, we created a simple Java application to create a certain number of records in our raw tags database. The configuration parameter consisted of the total number of taggings required, the number of URI's, the number of users, and the number of tags. We varied these numbers, and ranged the number of taggings required between 10 and 10,000. The results from this test are shown in table 5.7.

As background, this test was performed on a Dell Inspiron 6000 laptop with two gigabytes of RAM and a 1.5GHz Intel Centrino CPU (with 2 megabytes of cache). The test was run whilst the computer was being used for general purpose work such as writing this thesis and programming. It should be noted that this laptop is woefully underpowered when compared to the target hardware that would normally run the Centruflow Server (and hence the brute force algorithms being tested here).

What the tests show is that the total time is actually not largely dependent on the number of taggings, but rather the number of possible combinations of URI's, users and tags. This can be seen in the the last two rows of table 5.7, where the first of the last two rows takes about 20% of the time of the second test. This is due to the increase in the number of potential combinations between the URI's, users and tags. There is however some randomness in these results, for example take the four rows where the number of taggings is 2000. The data that was created in the second run where there were 1000 URI's, 1000 users and 1000 tags actually resulted in a longer running time than when there were 2000 URI's, users and tags. This can be seen in the graph shown in figure 5.1.

From what is shown in these results, we can quickly see that the brute force algorithm will not scale to a situation where there are tens of thousands of taggings, but works adequately well for systems where the number of taggings is less than 10,000. It is likely that in almost

Number of Taggings	Number of Unique URI's, Users, and Tags	Time to calculate Distances	Time to Write Distances to Database	Time to write Refined Tags to Database	Total Time (Milliseconds)
10	10	197	138	393	728
10	1	0	30	563	593
10	100	48	64	389	501
50	10	343	1,283	1,444	3,050
50	25	507	884	1,411	2,802
50	50	495	1,164	1,343	3,002
100	10	456	1,211	2,944	4,611
100	25	706	3,273	2,363	6,342
100	50	1,124	3,611	3,519	8,254
100	100	1,569	1,246	3,664	6,479
100	500	2,319	361	3,731	6,411
250	10	2,156	1,621	9,801	13,578
250	50	3,460	11,547	7,306	22,313
250	100	5,736	9,939	8,593	24,268
250	250	10,953	3,321	8,916	23,190
500	50	7,717	28,934	17,205	53,856
500	250	28,689	15,254	18,020	61,963
1000	50	20,568	42,409	28,377	91,354
1000	250	66,583	83,744	30,167	180,494
1000	500	116,473	33,049	39,310	188,832
1000	1000	175,970	21,673	41,823	239,466
2000	500	325,603	168,160	64,670	558,433
2000	1000	601,310	148,172	106,311	855,793
2000	2000	695,453	28,653	62,158	786,264
2000	5000	1,174,101	16,976	82,393	1,273,470
5000	1000	2,305,599	391,931	153,041	2,850,571
10000	100	750,942	161,363	314,738	1,227,043
10000	1000	1,425,790	1,425,790	308,310	5,154,518

Table 5.7: Results from a performance and stress test.

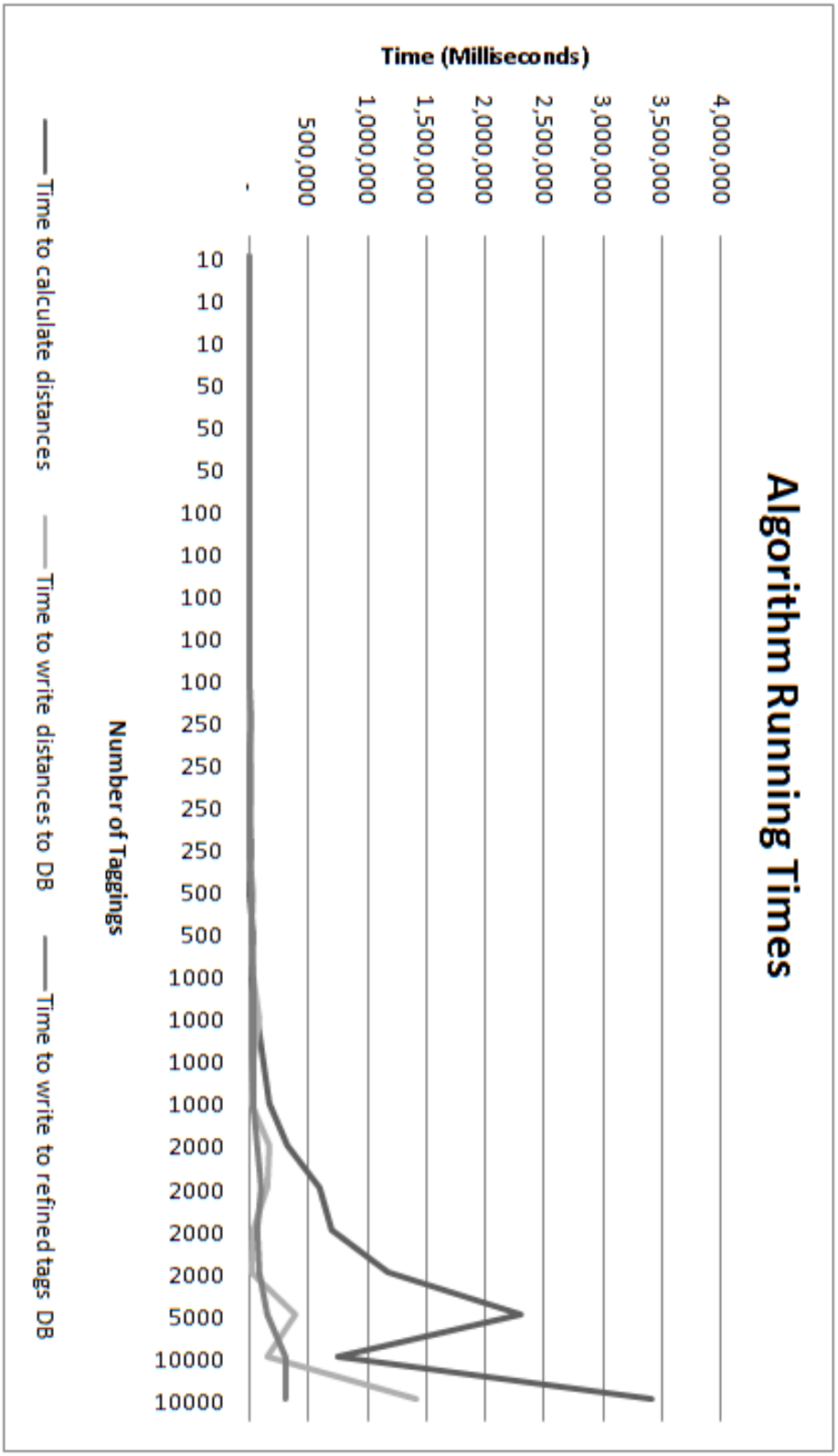


Figure 5.1: Results of the performance/stress tests.

every use case for Centruflow, this will be the case given that most customers only provide Centruflow access to a small subset of their entire organisation. Therefore, we are quite happy with these results.

It should be noted that the algorithm used is a precise copy of the algorithm developed earlier in this thesis, and that performance could potentially be improved quite considerably by paying better regard to the requirements of a computer, in particular memory and database connection requirements. In testing our algorithm, both of these areas proved to have considerable cost to the system, and so more work will be needed in the future to ensure that the algorithm is efficiently using the systems resources.

Chapter 6

Conclusion

6.1 Introduction

Having implemented our algorithms and software, and validated our approach, this chapter looks back on the project goals outlined in chapter 1.4, to assess the effectiveness of this masters thesis. Following this we conclude with recommendations for future work.

6.2 Conclusion

In this thesis we set forth to infuse Centruflow with the ability to reason about tagging information using mathematical algorithms. Prior to attempting this, we had to invest a significant portion of our time into three important areas:

1. Researching and understanding the technologies surrounding the semantic web.
2. Undertaking considerable development work to upgrade Centruflow to version 3.0, which contained the necessary frameworks required for this thesis.
3. Developing the Centruflow Server product that enables Centruflow to query data sources using semantic web technology.

Having undertaken the above work, and with our new algorithms in place and functioning on our tagging data, we were very pleased with the quality of the inferred relationships between nodes within Centruflow. To see Centruflow offering advanced insight into data using this inferred knowledge is impressive, and potentially very valuable. In our demonstrations to key people involved with the Centruflow product, they were similarly impressed. Up until Centruflow 3.0, Centruflow was simply a 'viewer' into a custom database. With the

developments that occurred during this thesis, we now have a framework to enable considerable intelligence within Centruflow, both through semantic web inferencing rules, as well as our tagging technology. In addition, we can far more easily and rapidly visualise a considerably larger amount of a companies information.

We must now release the tagging functionality publicly to allow for real-world testing and gathering of feedback. We can then use this information to improve future versions of Centruflow and the tagging plugin. This is not the end for Centruflow - there is still much more work that can be done to further increase the value to users, as noted in the future work section below. By improving Centruflow through semantic web technologies, we have moved it into a considerably larger playing field - the entire semantic web. As noted in chapter one, we stand on the shoulders of giants, and in return, Centruflow offers clarity and understanding to the future of interconnected information on the semantic web.

6.3 Future Work

There is a considerable amount of work that could be continued on from this thesis that was simply not feasible due to time constraints. We conclude this thesis by exploring the areas of work that should be pursued in the future.

6.3.1 Further Research

Bootstrapping The Taggings System There needs to be greater investigation into the means through which users can be encouraged to participate in the bootstrapping of the tagging system. This is important as without this we can envisage a situation where users of Centruflow derive no value from the tagging system, as there is minimal acceptance of the tagging functionality.

6.3.2 Algorithm Improvements

On-Demand Algorithm Implementation As discussed in section 4.1 we implemented a brute-force implementation for calculating user trust and resource distance. We stated that this approach would only be feasible for small tagging databases, and that an on-demand approach would need to be adopted before long.

Improving The User Trust Algorithm At present the user trust algorithm only uses the taggings that users create to calculate trust. We believe that the user trust algorithm should grow as new social functionality becomes available. In particular, future versions of Centruflow will allow for users to modify information that they deem incorrect. The

user trust algorithm should track these changes to find any user traits that may infer a particularly knowledgeable or malicious user.

6.3.3 Software Improvements

Improved 'Just In Time' Querying To ensure that Centruflow is as efficient as can be, we should continue to improve the algorithms used to query the Centruflow Server such that data is retrieved at the last possible moment.

Improved Memory Management When locally cached data is unlikely to be visited by the Centruflow client, it should be removed from memory so that it does not waste memory. This is very important when dealing with a million or more nodes.

Support For Reasoning (Centruflow Client) Now that Centruflow 3.0 is built on top of the Jena semantic web framework, it is possible for far more advanced reasoning to be introduced easily, for example using SWRL rules.

Multiple Datasource Merging (Centruflow Server) An important function that needs to be developed is the ability to effectively merge multiple data sources such that they can be visualised in one Centruflow visualisation. This requires a great deal of consideration to ensure that all major requirements are effectively met, and exposed in such a way that it can be easily configured by an administrator. Due to the size of the scope for this requirement and the complexity of implementing a solution, this unfortunately could not be a part of this masters thesis. Some of the key requirements and considerations for this include:

- The ability to merge nodes such that one node is composed of information from multiple data sources.
- The ability to easily link information in one graph to information in another graph.
- The ability to support the editing and deleting of information in these scenarios from within the Centruflow client.

General Server Improvements There is a large list of improvements, including:

- Improving the REST architecture with a more powerful implementation, perhaps based on a framework such as Jersey [56].

- Making it far more feasible for the server to retrieve useful information and reason on it than it is now.
- Support for non-database data sources, such as RDF files, and 'spidered' files on a local intranet.

6.4 Summary

This chapter has summarised the results of this thesis, and presented work that may be continued in the future to extend this thesis. The remainder of this thesis contains a

References

[1] T. Berners-Lee, "The World Wide Web: A very short personal history," Last visited 10/10/07. [Online]. Available: http://www.w3.org/People/Berners-Lee/ShortHistory

[2] "Centruflow," Last Visited 04/10/07. [Online]. Available: http://www.centruflow.com

[3] "World Wide Web Consortium (W3C)." [Online]. Available: http://www.w3.org/

[4] K. Cheruvettolil, "Does anybody know of a practical implementation of Semantic Web Technologies?" [Online]. Available: http://tinyurl.com/youxu5

[5] D. Takahashi, "A 100 Trillion Words On The Internet; An Afternoon At Google Developer Day." [Online]. Available: http://tinyurl.com/ywaldl

[6] "The Web Standards Project," Last visited 23/08/07. [Online]. Available: http://www.webstandards.org/about/mission/

[7] "Internet World Stats," Last visited 17/06/07. [Online]. Available: http://www.internetworldstats.com/stats.htm

[8] "Remote Method Invocation (RMI)," Last visited 27/10/07. [Online]. Available: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[9] M. Henning, "The Rise and Fall of CORBA," Last visited 23/08/07. [Online]. Available: http://acmqueue.com/modules.php?name=Content&pa=showpage&pid=396&page=1

[10] "SOAP Version 1.2 Part 0: Primer (Second Edition)," Last visited 17/06/07. [Online]. Available: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

[11] S. Loughran and E. Smith, "Rethinking the Java SOAP Stack," Hewlett-Packard Bristol Laboratories, Tech. Rep., May 2005.

[12] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation.

[13] T. O'Reilly, "What Is Web 2.0," Last visited 17/06/07. [Online]. Available: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[14] "MySpace." [Online]. Available: http://www.myspace.com

[15] "Facebook." [Online]. Available: http://www.facebook.com

[16] "Bebo." [Online]. Available: http://www.bebo.com

[17] "LinkedIn." [Online]. Available: http://www.linkedin.com

[18] "TradeMe." [Online]. Available: http://www.trademe.co.nz

[19] "eBay." [Online]. Available: http://www.ebay.com

[20] T. Berners-Lee, *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its inventor*. HarperCollins, 1999.

[21] W3C, "Gleaning Resource Descriptions from Dialects of Languages (GRDDL)," Last visited 05/10/07. [Online]. Available: http://www.w3.org/TR/2007/REC-grddl-20070911/

[22] J. Daly, M. Forgue, and Y. Hirakawa, "W3C Completes Bridge Between HTML/Microformats and Semantic Web," Last visited 05/10/07. [Online]. Available: http://www.w3.org/2007/07/grddl-pressrelease

[23] B. Adida and M. Birbeck, "RDFa Primer 1.0," Last visited 05/10/07. [Online]. Available: http://www.w3.org/TR/xhtml-rdfa-primer/

[24] T. Berners-Lee, "Semantic Web Road map," September 1998. [Online]. Available: http://www.w3.org/DesignIssues/Semantic.html

[25] "prefuse visualisation toolkit." [Online]. Available: http://prefuse.org

[26] "Abstract Engineering Limited," Last visited 04/10/07. [Online]. Available: http://www.aeiou.co.nz

[27] J. Giles and J. Dietrich, "Refactoring Centruflow Using Plugins," *Projects*, vol. 15, 2007.

[28] BusinessWeek, "Q&A with Tim Berners-Lee," Last visited 28/07/07. [Online]. Available: http://www.businessweek.com/technology/content/apr2007/tc20070409_961951.htm

[29] S. Bratt, "Semantic Web, and other technologies to watch," Last visited 16/11/07 2007. [Online]. Available: http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/

[30] W3C, "Extensible Markup Language (XML) 1.0 (Fourth Edition)." [Online]. Available: http://www.w3.org/TR/2006/REC-xml-20060816/

[31] K. Sall, "Well-Formed vs. Valid Documents," Last visited 18/10/07. [Online]. Available: http://wdvl.internet.com/Authoring/Languages/XML/XMLFamily/XMLSyntax/sall3_1.html

[32] "Resource Description Framework (RDF)," Last visited 07/10/07. [Online]. Available: http://www.w3.org/RDF/

[33] F. Manola and E. Miller, "RDF Primer," Last visited 18/10/07. [Online]. Available: http://www.w3.org/TR/REC-rdf-syntax/

[34] "Notation 3 (N3): A readable RDF syntax," Last visited 07/10/07. [Online]. Available: http://www.w3.org/DesignIssues/Notation3.HTML

[35] D. Beckett, "Turtle - Terse RDF Triple Language," Last visited 14/11/07. [Online]. Available: http://www.dajobe.org/2004/01/turtle/

[36] S. Powers, *Practical RDF*. Beijing, Cambridge: O'Reilly, 2003.

[37] D. Brickley and R. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema," Last visited 18/10/07. [Online]. Available: http://www.w3.org/TR/rdf-schema/

[38] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," Last visited 1/10/07. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[39] E. Prudhommeaux and A. Seaborne, "SPARQL Query Language for RDF (W3C Working Draft 4 October 2006)," Last visited. [Online]. Available: http://www.w3.org/TR/2006/WD-rdf-sparql-query-20061004/

[40] A. Seaborne and G. Manjunath, "SPARQL/Update: A language for updating RDF graphs," Last visited 1/10/07. [Online]. Available: http://jena.hpl.hp.com/ afs/SPARQL-Update.html

[41] W3C, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," Last visited 14/11/07. [Online]. Available: http://www.w3.org/Submission/SWRL/

[42] "RuleML." [Online]. Available: http://www.ruleml.org/

[43] C. Bizer, "Semantic Web Trust and Security Resource Guide," Last visited 14/10/07. [Online]. Available: http://sites.wiwiss.fu-berlin.de/suhl/bizer/SWTSGuide/

[44] J. M. R. Jr., "Finding Bacon's Key: Does Google Show How the Semantic Web Could Replace Public Key Infrastructure?" Last visited 14/10/07 2002. [Online]. Available: http://www.w3.org/2002/03/key-free-trust.html

[45] R. Matthew, R. Agrawal, and P. Domingos, "Trust Management for the Semantic Web," 2003. [Online]. Available: citeseer.ifi.uzh.ch/matthew03trust.html

[46] J. Golbeck, B. Parsia, and J. Hendler, "Trust Networks on the Semantic Web," in *Proceedings of Cooperative Intelligent Agents*, 2003.

[47] J. Golbeck and J. Hendler, "Reputation Network Analysis for Email Filtering," 2004. [Online]. Available: citeseer.ist.psu.edu/golbeck04reputation.html

[48] A. Abdul-Rahman, "The PGP Trust Model," Department of Computer Science, University College London, Tech. Rep.

[49] W. Stallings, "The PGP Web of Trust," Last visited 14/10/07 1995. [Online]. Available: http://www.byte.com/art/9502/sec13/art4.htm

[50] "WordNet::Similarity," Last visited 14/10/07. [Online]. Available: http://www.d.umn.edu/ tpederse/similarity.html

[51] R. P. I. Cognitive Science, "Measures of Semantic Relatedness (MSR)." [Online]. Available: http://cwl-projects.cogsci.rpi.edu/msr/

[52] "WordNet." [Online]. Available: http://wordnet.princeton.edu/

[53] "GeneOntology." [Online]. Available: http://www.geneontology.org/

[54] P. Resnik, "Using Information Content to Evaluate Semantic Similarity in a Taxonomy," in *IJCAI*, 1995, pp. 448–453. [Online]. Available: citeseer.ist.psu.edu/resnik95using.html

[55] G. Varelas, E. Voutsakis, P. Raftopoulou, E. G. M. Petrakis, and E. E. Milios, "Semantic similarity methods in wordNet and their application to information retrieval on the web," in *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*. New York, NY, USA: ACM Press, 2005, pp. 10–16. [Online]. Available: http://portal.acm.org/citation.cfm?id=1097051

[56] "Jersey: RESTful Web Services made easy in Java." [Online]. Available: http://jersey.dev.java.net/

[57] T. V. Wal, "Folksonomy Coinage and Definition," Last visited 16/10/07. [Online]. Available: http://vanderwal.net/folksonomy.html

[58] "Del.icio.us." [Online]. Available: http://del.icio.us

[59] "Flickr." [Online]. Available: http://flickr.com

[60] U. A. Mejias, "Tag Literacy," Last visited 16/10/07. [Online]. Available: http://ideant.typepad.com/ideant/2005/04/tag_literacy.html

[61] M. Guy and E. Tonkin, "Folksonomies: Tidying up Tags?" *D-Lib Magazine*, vol. 12, no. 1, January 2006. [Online]. Available: http://www.dlib.org/dlib/january06/guy/01guy.html

[62] A. Mathes, "Folksonomies - Cooperative Classification and Communication Through Shared Metadata," Last visited 16/10/07. [Online]. Available: http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html

[63] P. Keller, "Tags: Database schemas," Last visited 1/10/07 2005. [Online]. Available: http://www.pui.ch/phred/archives/2005/04/tags-database-schemas.html

[64] N. Borwankar, "TagSchema," Last visited 21/10/07. [Online]. Available: http://tagschema.com/blogs/tagschema/

[65] A. Nolley, "MySQLicious - del.icio.us to MySQL Mirroring," Last visited 21/10/07. [Online]. Available: http://nanovivid.com/projects/mysqlicious/

[66] "TagCommons." [Online]. Available: http://tagcommons.org/

[67] H. S. Al-Khalifa and H. C. Davis, "Towards better understanding of folksonomic patterns," in *HT '07: Proceedings of the 18th conference on Hypertext and hypermedia,*.

[68] S. Golder and B. A. Huberman, "The Structure of Collaborative Tagging Systems," Aug 2005. [Online]. Available: http://arxiv.org/abs/cs.DL/0508082

[69] P. Mika, "Ontologies Are Us: A Unified Model of Social Networks and Semantics." in *The Semantic Web - ISWC 2005, Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10,*.

[70] D. Levandowsky, Michael; Winter, "Distance between Sets," *Nature*, no. 234, pp. 34–35, November 1971.

[71] G. Gilbert, "Distance between Sets," *Nature*, no. 239, p. 174, September 1972.

[72] Richard Newman, "Tag ontology design." [Online]. Available: http://www.holygoat.co.uk/projects/tags/

[73] "OpenLink Software," Last visited 2/10/07. [Online]. Available: http://www.openlinksw.com/virtuoso/

[74] "Aduna Software," Last visited 2/10/07. [Online]. Available: http://www.aduna-software.com

[75] "Aperture Framework," Last visited 2/10/07. [Online]. Available: http://aperture.sourceforge.net/

[76] "Thetus Corporation," Last visited 02/10/07. [Online]. Available: http://www.thetus.com/

[77] G. Goth, "Data-Driven Enterprise: Slouching toward the Semantic Web," *IEEE Distributed Systems Online*, vol. 7, 2006.

[78] "Jena." [Online]. Available: http://jena.sourceforge.net

[79] "Jetty." [Online]. Available: http://www.mortbay.org/

[80] C. Bizer and R. Cyganiak, "D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs," Last visited 03/10/07. [Online]. Available: http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2rq/index.htm

[81] “Java Web Start.” [Online]. Available: http://java.sun.com/products/javawebstart/

Glossary

API Stands for 'Application Programming Interface', it is a collection of source code interfaces that allow an application to interact with a code library.

Centruflow Centruflow is a Java-based application that is used to visualise structured data, particularly data found in corporate databases. It visualises the information as a connected, bidirectional graph.

Data source A data source is any collection of information, including database, XML file, RDF file, SPARQL endpoint, Microsoft Word or Excel file, etc.

FOAF Friend of a friend, is a OWL ontology used to describe information about a person, and the relationships between people.

GPL The General Public License is often referred to as a 'viral' license, in that any public software project that makes use of GPL'ed code must also release its source code under the GPL to the public. This is not very business-friendly, and so it is often avoided by businesses.

HTML HyperText Markup Language. HTML is the language used to define the structure of webpages. Recently a new version has been released that is well-defined XML, known as XHTML.

HTTP HyperText Transport Protocol is a communications protocol used on the Internet to transfer information between computers.

Metadata Metadata is simply data about data. In the case of taggings, metadata includes the date the tagging was applied, which template it belongs to, etc.

OWL Web Ontology Language. OWL is used to define new classes of resources, and specify how they relate.

RDF Schema RDF Schema is the simplest ontology language developed by the W3C. It does not have the complexity/expressiveness of OWL, and so is useful in circumstances where the needs of the ontology are basic.

RPC Remote Procedure Call. RPC is a technology used to allow for applications to communicate with each other through a network.

SPARQL Stands for 'Sparql Protocol and RDF Query Language'. Is a query language somewhat resembling SQL queries, but is inherently graph-based and is therefore used to query RDF graphs.

SWRL Semantic Web Rule Language. This is a language that allows for rules to be inferred based on data in an RDF graph.

Thinkmap Thinkmap is a New York based software company specialising in the development of visualisation technology. Centruflow 1.x and 2.x made use of Thinkmap technology.

WWW World Wide Web. This is perhaps the most popular aspect of the Internet. The world wide web is the series of interconnected pages, generally developed in HTML and retrieved using HTTP.

XPath XPath is a language for querying portions of an XML document. It can also compute values. XPath is based on a tree representation of the XML document. XPath is used extensively in XSLT.

XSLT Extensible Stylesheet Language Transformation is an XML language used to transform XML documents into other documents (not necessarily XML). XSLT does not change the original document but rather creates a new document.

RDF The Resource Description Format is a simple ontology language used to express knowledge on the internet.

XML Extensible Markup Language is essentially a text format for representing data using tags.

Appendix A

Use Cases

Whenever we develop for Centruflow, it is important to relate back to use cases, as use cases directly reflect the needs of end users. This appendix outlines all use cases that should be made available to users as part of Centruflow 3.0 over and above those in Centruflow 2.0. Despite this document being an attempt to outline use cases, it does also offer thoughts on how implementation and/or interface design *may* be done, and what constraints/requirements must be considered. We consider this to be a valuable exercise as it provides a single point of reference for all thoughts relating to the features that are planned for the next major Centruflow release.

A.1 Use Cases

A.1.1 Tagging Entities (Bookmarking)

A.1.1.1 Summary

A user tags an entity to effectively impart implicit knowledge against an explicit data model. Tagging is effectively equivalent to bookmarking in the usual sense, insofar that a user may later return to their tagged entities by searching based on a tag (see use case A.1.2). Where tagging extends bookmarking includes the following:

- The same tags can be applied to multiple entities by the same user (and by any other user).
- An entity can have multiple tags applied to it.

There is also the underlying benefit to tagging - it allows for the identification of related entities through tagging similarities. This is outlined in use case A.1.4. A user should be

made aware that through proper use of tagging that this 'second-order benefit' will make itself apparent. In effect, the act of tagging therefore offers two benefits to the user: a means to bookmark entities, and a means to find implicit relationships between entities.

A.1.1.2 Basic Course of Events

To tag an entity, the user must work through the following steps:

1. The user must select the entity that they wish to tag.
2. The user must select the option to tag the entity.
3. The user should enter the tags that they wish to give to the entity.
    (a) As the user inputs tags, they should be prompted by the system with suggested tags. This includes both the users own tags as well as popular tags that have been applied to the entity.
4. After inputting all tags, the user should select the option to submit all tags.
5. The system should store these tags in a persistent data store.

A.1.1.3 Implementation Thoughts

The major limitation to getting the second-order benefits of tagging is one of 'bootstrapping the system' - users must be actively encouraged to tag prolifically, and also encouraged to use common and sensible tags. 'Common and sensible tags' refers to many things, including:

1. Using a particular tagging syntax (i.e. do we allow spaces in tags, or does that mean two separate tags have been applied? Do we use camel type or underscores to conjoin words?)
2. A system of proposing tags that have already been applied to the current entity.
3. A system of 'auto-completing' the tag that the user is typing by looking up the other tags in the system.

By doing this, the user is not restricted in the terms that they may use for a tag (allowing complete freedom - in other words a 'folksonomy'). This strengthens the ability for Centruflow to offer better insight into implicit relationships within Centruflow (i.e see use case A.1.4).

A.1.2 Tag Searching

A.1.2.1 Summary

A user, having input tags into Centruflow, will want to later return to the entities they tagged. To enable this the user should be able to input their desired tag and see all entities that have been given this tag. The search could be using just the users tags, or alternatively, the entire tag database (i.e. all tags contributed by all users). The user will be given all entities that have this particular tag applied to them. To further refine their search the user may proceed to enter additional tags. The user may sort the results based on criteria such as the number of times the particular tag has been applied, or the recency with which the tag has been applied.

A.1.2.2 Basic Course of Events

To search for entities based on tags, the following steps should be followed:

1. The user should bring up the 'tag search' dialog.
2. The user should type into the dialog the tags that they are wanting to search for.
   (a) The system, as the user types, should refine the results dialog to show only tags that have the tag.
   (b) The system should suggest related tags to the user if the number of results is small.
3. The user may optionally:
   (a) Select a node from the dialog to make it the current 'selected node' onscreen.
   (b) Filter the results dialog based on the available criteria.

A.1.2.3 Implementation Thoughts

An important question is how to handle multiple tags being used as search criteria, in other words, are they joined using conjunction or disjunction? The use of conjunctions between terms would return results that only have all tags, whereas disjunction will return any entity with one or more of the tags input by the user. Of course, one approach could be to ask the user whether they want to use conjunction or disjunction.

To display the search results to the user could be implemented in one of two ways: visually (as an extension to the graph) or as a list to the side of the graph. Which implementation is used will need to be researched in terms of end-user understanding.

A.1.3 Tag Untagging

A.1.3.1 Summary

When the user is using Centruflow, they will frequently be shown tags associated with entities. In some circumstances, the user may see a tag that they believe to be so ludicrous or wrong that they may wish to do something about it - they may want to say 'I do not believe that this tag fairly represents the entity for which it is associated with in this particular case'. Allowing the user to do this allows them to impact the 'related entities' output in use case A.1.4. It effectively provides two pieces of important information:

1. The tag in this instance does not seem overly valid to a particular user.
2. The user who applied the tag is likely to be less trustworthy.

The concept of tag untagging is a novel one - it seemingly has not yet been pursued in any of the popular online tagging services (in particular del.icio.us). This will therefore be one of the more novel bits of research that is being undertaken as part of this masters research.

A.1.3.2 Basic Course of Events

Whenever the user is presented with a tag, they should be able to easily interact with it. Given this assumption, the basic course of events is:

1. The user selects the tag that they disagree with.
2. The user selects the 'untag' option related to the tag.
3. The system takes this and submits it to a persistent data store.

A.1.3.3 Implementation Thoughts

One important consideration is the difference between public and private tags. Private tags should be discouraged as they can not be shown to other users or used in similarity calculations. The one place where private tags should be used is when they will clearly not make sense to other users, and in making them public could result in other users untagging the tag.

A.1.4 Browsing Related Entities

A.1.4.1 Summary

This use case allows a user to be referred to other entities that may not be explicitly shown in any graph, but instead be the result of inferring relationships based on the similarity of entities using a distance function. The benefit to a user is that such implicit relationships (that were invisible in previous versions of Centruflow unless made explicit through the physical addition of an edge to the graph) can be clearly viewed and interacted with.

A.1.4.2 Basic Course of Events

This assumes that there is always a 'selected node'. In the case of Centruflow, this can be considered to be the 'centre node', i.e. the node that was last clicked on. Assuming that there is always a clear centre node, the basic course of events is:

1. The user selects a node on screen, which proceeds to become the centre node.
2. The system submits this node and calculates related nodes..
3. The system returns a short list of related entities.
4. Optionally, the user may:
   (a) Shorten or lengthen the list to see more or less results.
   (b) Choose to see 'next results', i.e. the next page of results like on a search engine.
   (c) Select a node in this list to make it the centre node.

A.1.4.3 Implementation Thoughts

Related entities are entities related to a 'central node' based on the similarity of tags. Therefore, providing this information to the user is a second-order effect of the tagging entities use case (use case A.1.1). To calculate the similarity of two entities will involve a novel approach based on the similarity of their tag sets, taking into account user trust and 'cleansing of the tags database'. This should allow for high quality suggestions to be provided to the user, and these suggestions should be frequently updated to take into account the current state of the tags database as well as the user trust calculations.

It is important to take into account user trust, as it provides a means of placing more value to tags that have been input by trustworthy users. Once again, to calculate trust we will use a novel approach that recursively defines trust based on the trust of other users who

have tagged and untagged resources that the user has tagged. These trust calculations are one of the criteria that can be used to cleanse the tags database prior to calculating related nodes; there are many more that could be used in addition to this based on user testing (i.e. finding what criteria provides users with the best suggestions).

Appendix B

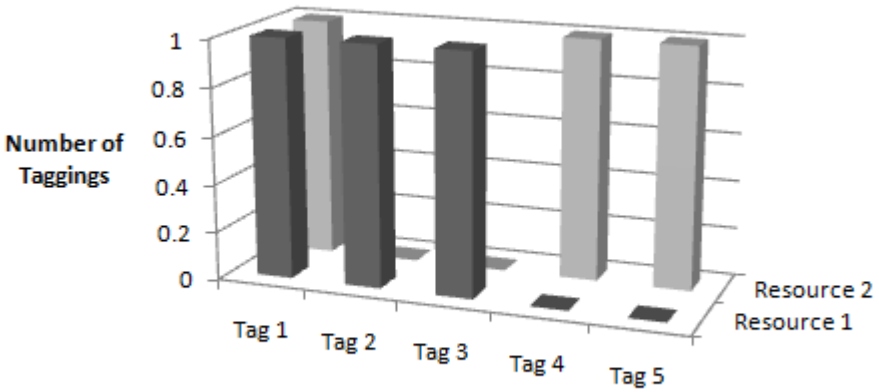
Resource Distance Examples

B.1 Tagging Distance Examples

B.1.1 Example Set One

Scenario One Consider two resources r1 and r2, which have the following tags:

- r1 has three tags (t1,t2,t3). Each tag has only been applied once.
- Similarly, r2 has three tags (t1,t4,t5). Each tag has only been applied once here also.



Notice that both resources have a single instance of the t1 tag. This means that they are somewhat related. Firstly, we calculate the *tagSum* for these two resources, and then we

may use this in our similarity and distance calculations:

$$tagSum(t_1, r_1, r_2) = ((1+1) \times \frac{1}{1}) = 2$$
$$overlapTagSum(t_1, r_1, r_2) = 2 \times \frac{1}{1} = 2$$
$$tagSum(t_2, r_1, r_2) = ((1+0) \times \frac{1}{1}) = 1$$
$$tagSum(t_3, r_1, r_2) = ((1+0) \times \frac{1}{1}) = 1$$
$$tagSum(t_4, r_1, r_2) = ((1+0) \times \frac{1}{1}) = 1$$
$$tagSum(t_5, r_1, r_2) = ((1+0) \times \frac{1}{1}) = 1$$

As should be obvious from the above, where there is zero instances of a tag in one resource, regardless of the number of tags belonging to the other resource, the *tagSum* will always be one. Therefore, from here on out, we will only show *tagSum* calculations where there is something non-trivial.

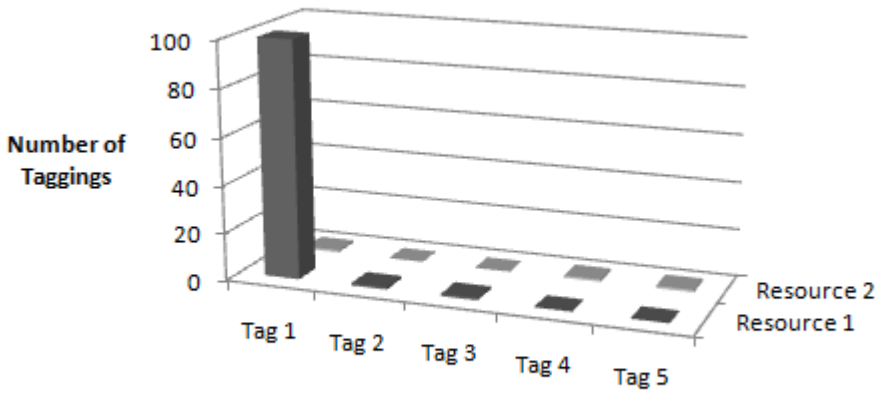
$$tagSimilarity(r_1, r_2) = \frac{\Sigma_{t \in taggings(r_1) \cap taggings(r_2)} tagSum(t, r_1, r_2) \times overlapRatio(t, r_1, r_2)}{\Sigma_{t \in taggings(r_1) \cup taggings(r_2)} tagSum(t, r_1, r_2)} = \frac{2 \times 1}{6}$$

$$tagDistance(r_1, r_2) = 1 - \frac{2}{6} = \frac{4}{6} = 0.667$$

The end result is that resources r1 and r2 are related, but only weakly.

Scenario Two Consider two resources r1 and r2, which have the following tags:

- r1 has three tags (t1,t2,t3). t1 has been applied 100 times (by different users), whilst t2 and t3 have been applied only once.
- Similarly, r2 has three tags (t1,t4,t5). Each tag has only been applied once here also.



Notice that both resources still have overlap due to the t1 tag, however now the number of instances is different.

$$tagSum(t_1, r_1, r_2) = ((100 + 1) \times \frac{1}{100}) = 1.01$$
$$overlapTagSum(t_1, r_1, r_2) = 1.01 \times \frac{1}{100} = 0.0101$$

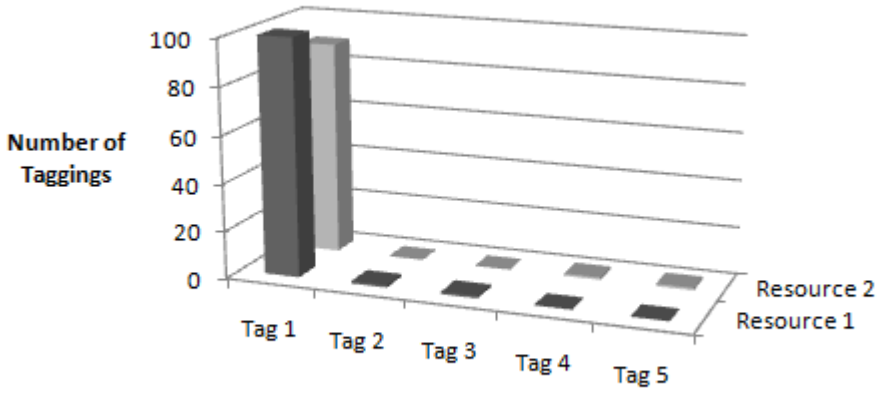
All other *tagSum* calculations (i.e. for t2, t3, t4, and t5) are equal to two.

$$tagSimilarity(r_1, r_2) = \frac{0.0101}{9.01}$$
$$tagDistance(r_1, r_2) = 1 - \frac{0.0101}{9.01} = 0.999$$

What this shows is that the distance between two resources actually increases the greater the difference between the *number of instances* of an overlapped tag. The resources in scenario two are actually further apart than in scenario one.

Scenario Three Consider two resources r1 and r2, which have the following tags:

- r1 has three tags (t1,t2,t3). t1 has been applied 100 times (by different users), whilst t2 and t3 have been applied only once.
- Similarly, r2 has three tags (t1,t4,t5). t1 has been applied 90 times (by different users), whilst t4 and t5 have been applied only once.



Notice that both resources still have overlap due to the t1 tag, however now the number of instances is closer than in previous scenarios.

$$tagSum(t_1, r_1, r_2) = ((100 + 90) \times \frac{90}{100}) = 171$$
$$overlapTagSum(t_1, r_1, r_2) = 171 \times \frac{90}{100} = 153.9$$

All other *tagSum* calculations (i.e. for t2, t3, t4, and t5) are equal to one.

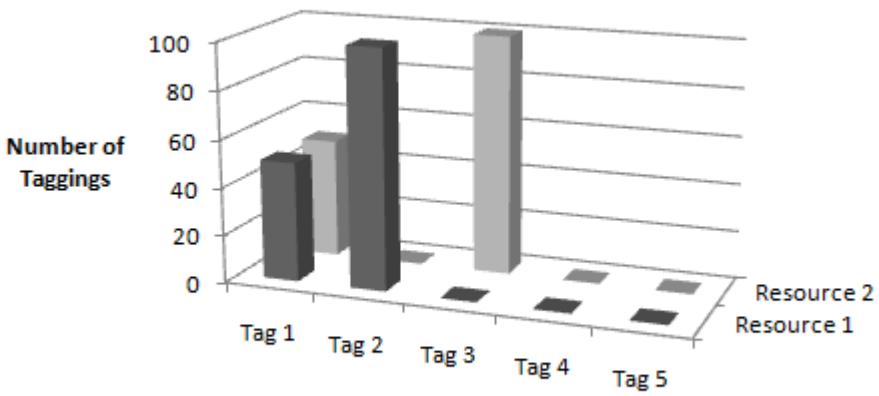
$$tagSimilarity(r_1, r_2) = \frac{153.9}{175}$$
$$tagDistance(r_1, r_2) = 1 - \frac{153.9}{175} = 0.1206$$

Notice now that the distance between r1 and r2 is the closest of all three scenarios. This is because the number of t1 tags that have been applied to r1 and r2 is rather large, so we take this into account, and value it more than the scenarios where one or both resources was only tagged with a tag once.

B.1.2 Example Set Two

Scenario One Consider two resources r1 and r2, which have the following tags:

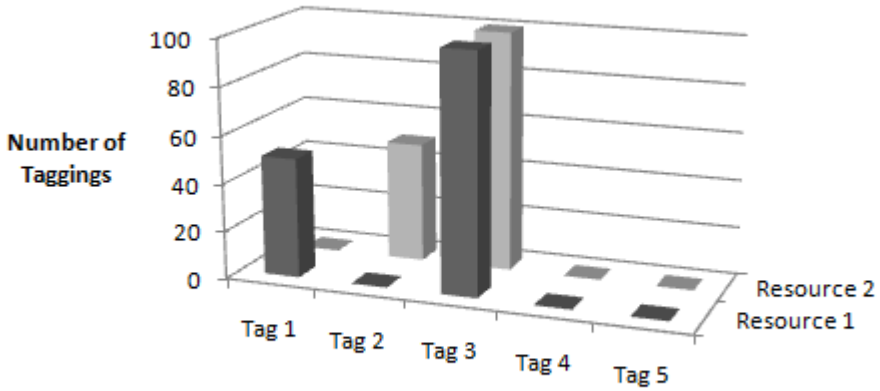
- r1 has two tags. t1 has been applied 50 times, whilst t2 has been applied 100 times.
- Similarly, r2 has two tags. t1 has been applied 50 times, whilst t3 has been applied 100 times.



Notice that both resources have a 50 instances of the t1 tag, but each has a unique tag (t2 and t3). This means that they should only be very, very weakly related, as both t2 and t3 have been applied a lot.

$$tagSum(t_1, r_1, r_2) = ((50 + 50) \times \frac{50}{50}) = 100$$
$$overlapTagSum(t_1, r_1, r_2) = 100 \times \frac{100}{100} = 100$$
$$tagSum(t_2, r_1, r_2) = ((100 + 0) \times \frac{1}{100}) = 1$$
$$tagSum(t_3, r_1, r_2) = ((0 + 1000) \times \frac{1}{100}) = 1$$
$$tagSimilarity(r1, r2) = \frac{100}{102}$$
$$tagDistance(r1, r2) = 1 - \frac{100}{102} = 0.0196$$

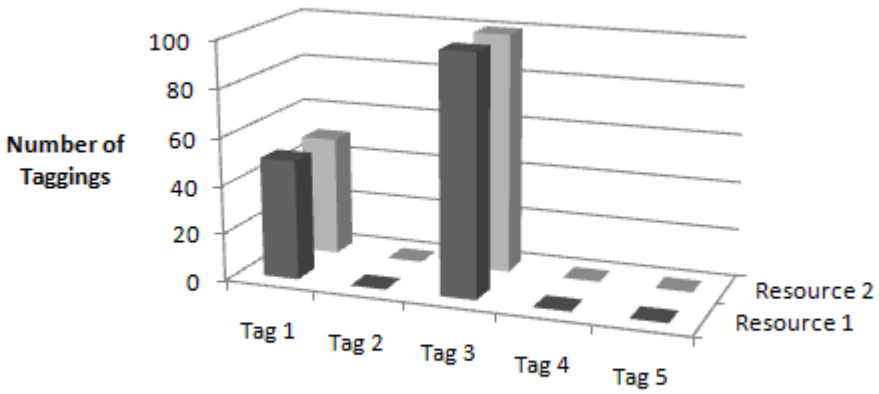
Scenario Two Scenario two is simply a reverse of the previous scenario: r1 and r2 have a unique tag t1 and t2 that have been applied 50 times, and they share t3, which has been applied 100 times for r1 and 100 times for r2.



$$tagSum(t_1, r_1, r_2) = ((50 + 0) \times \frac{1}{50}) = 1$$
$$tagSum(t_2, r_1, r_2) = ((0 + 50) \times \frac{1}{50}) = 1$$
$$tagSum(t_3, r_1, r_2) = ((100 + 100) \times \frac{100}{100}) = 200$$
$$overlapTagSum(t_3, r_1, r_2) = 200 \times \frac{100}{100} = 200$$
$$tagSimilarity(r1, r2) = \frac{200}{202}$$
$$tagDistance(r1, r2) = 1 - \frac{200}{202} = 0.0099$$

The end result is that resources r1 and r2 are very related, as the distance between them is very tiny.

Scenario Three Scenario three is simply a merge of the previous two scenarios, in other words r1 and r2 share all tags.



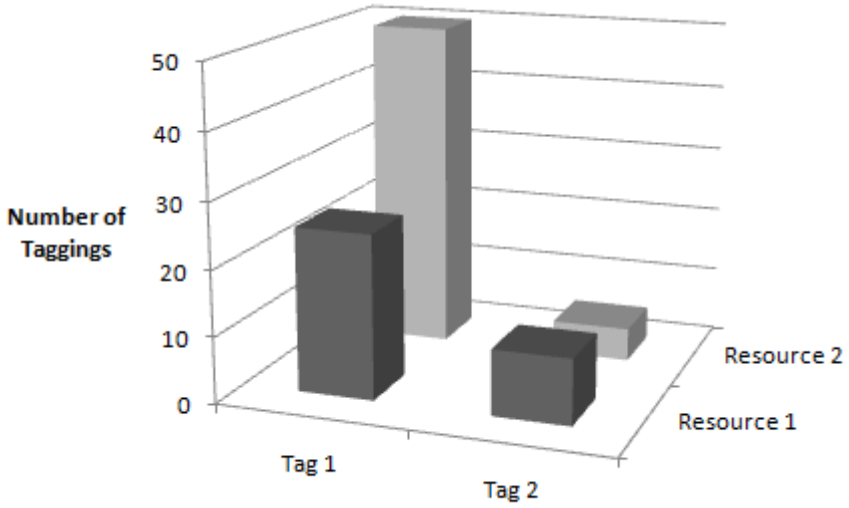
$$tagSum(t_1, r_1, r_2) = ((50 + 50) \times \frac{50}{50}) = 100$$
$$overlapTagSum(t_1, r_1, r_2) = 100 \times \frac{50}{50} = 100$$
$$tagSum(t_2, r_1, r_2) = ((100 + 100) \times \frac{100}{100}) = 200$$
$$overlapTagSum(t_2, r_1, r_2) = 200 \times \frac{100}{100} = 200$$
$$tagSimilarity(r1, r2) = \frac{300}{300} = 1$$
$$tagDistance(r1, r2) = 1 - 1 = 0$$

The end result is that resources r1 and r2 are completely related, as the distance between them is now 0.

B.1.3 Example Set Three

Scenario One Consider two resources r1 and r2, which have the following tags:

- r1 has two tags (t1,t2). t1 has been tagged 25 times and t2 has been tagged 10 times.
- r2 has the same two tags, but with different values: t1 has been tagged 50 times, and t2 has been tagged 5 times.



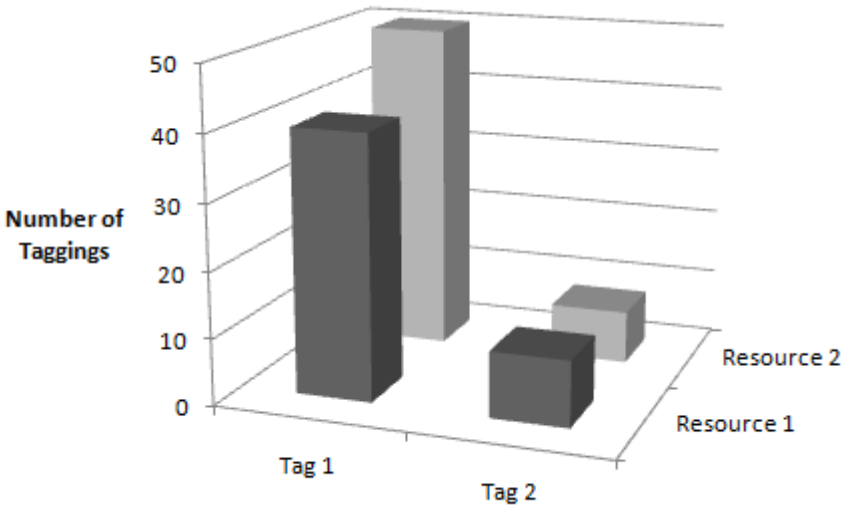
$$tagSum(t_1, r_1, r_2) = ((25 + 50) \times \frac{25}{50}) = 37.5$$
$$overlapTagSum(t_1, r_1, r_2) = 37.5 \times \frac{25}{50} = 18.75$$
$$tagSum(t_2, r_1, r_2) = ((10 + 5) \times \frac{5}{10}) = 7.5$$
$$overlapTagSum(t_2, r_1, r_2) = 7.5 \times \frac{5}{10} = 3.75$$
$$tagSimilarity(r1, r2) = \frac{22.5}{45}$$
$$tagDistance(r1, r2) = 1 - \frac{22.5}{45} = 0.5$$

The end result is that resources r1 and r2 are related, but as expected, only by a distance of 0.5.

Scenario Two This scenario alters the tag values from the previous scenario. They are now:

- For r1, t1 has been tagged 40 times and t2 has been tagged 10 times.
- For r2, t1 has been tagged 50 times, and t2 has been tagged 8 times.

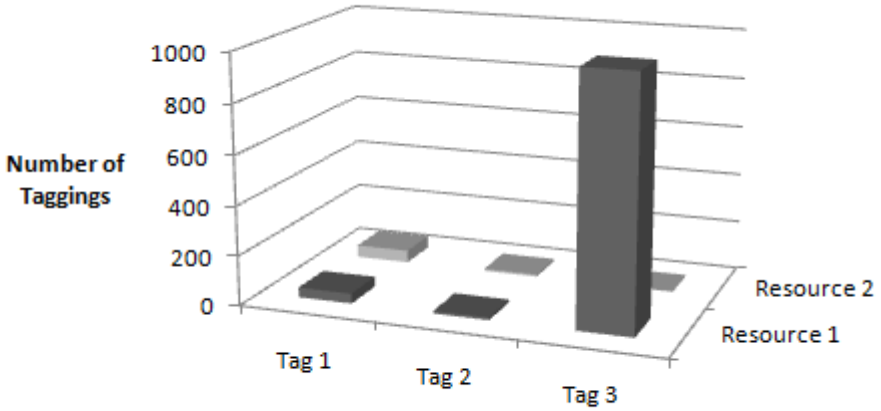
The key point in this scenario is that the tag values have come closer together than in the previous scenario.



$$tagSum(t_1, r_1, r_2) = ((40 + 50) \times \frac{40}{50}) = 72$$
$$overlapTagSum(t_1, r_1, r_2) = 72 \times \frac{40}{50} = 57.75$$
$$tagSum(t_2, r_1, r_2) = ((10 + 8) \times \frac{8}{10}) = 14.4$$
$$overlapTagSum(t_2, r_1, r_2) = 14.4 \times \frac{8}{10} = 11.52$$
$$tagSimilarity(r1, r2) = \frac{69.27}{86.4} = 0.802$$
$$tagDistance(r1, r2) = 1 - \frac{69.27}{86.4} = 0.198$$

By bringing the tag values closer together, the distance between the two resources has decreased, as one would expect.

Scenario Three This scenario alters the previous scenario by simply adding t3 to r1. t3 is a tag that has been applied 1000 times. All other values are the same as scenario two. This therefore does not modify the overlap value whatsoever, but of course the sharedTagSet is increased significantly.



$$tagSum(t_1, r_1, r_2) = ((40 + 50) \times \frac{40}{50}) = 72$$
$$overlapTagSum(t_1, r_1, r_2) = 72 \times \frac{40}{50} = 57.75$$
$$tagSum(t_2, r_1, r_2) = ((10 + 8) \times \frac{8}{10}) = 14.4$$
$$overlapTagSum(t_2, r_1, r_2) = 14.4 \times \frac{8}{10} = 11.52$$
$$tagSum(t_3, r_1, r_2) = ((1000 + 0) \times \frac{1}{1000}) = 1$$
$$tagSimilarity(r1, r2) = \frac{69.27}{87.4} = 0.802$$
$$tagDistance(r1, r2) = 1 - \frac{69.27}{87.4} = 0.207$$

By adding in this extra tag, we have shown that the calculation acts as we expect - r1 and r2 can not be overly related if 1000 people have tagged r1 with t3, but no one has tagged r2 with t3. Still, there is considerable overlap of the other two tags, so overall the distance between the two resources is relatively low.

B.1.4 Example Set Four

To calculate the effect that untaggings have on the effective tag count, we must define what the γvalue is. For now, we will test each of our scenarios below with 4 values of γ, those being 0.25, 0.5, 0.75 and 1.

Scenario One Consider two resources r1 and r2, which have the following tags:

- r1 has three tags (t1,t2,t3), with the following tag counts: t1: 25, t2: 42, t3: 10. In

addition to this, they all have some untags applied to them. The untag counts are: t1: 12, t2: 0, t3: 15.

- r2 has four tags (t1,t2,t4,t5), with the following tag counts: t1: 7, t2: 15, t4: 20, t5: 23. In addition to this, they all have some untags applied to them. The untag counts are: t1: 2, t2: 10, t4: 1, t5: 3.

Effective Tag Counts

$$effectiveTagCount(t, r) = \max(0,\, tagCount(t, r) - \gamma \times untagCount(t, r))$$
$$effectiveTagCount(t_1, r_1) = \max(0,\, 25 - \gamma \times 12) = \{22, 19, 16, 13\}$$
$$effectiveTagCount(t_2, r_1) = \max(0,\, 42 - \gamma \times 0) = \{42, 42, 42, 42\}$$
$$effectiveTagCount(t_3, r_1) = \max(0,\, 10 - \gamma \times 15) = \{6.25, 2.5, 0, 0\}$$
$$effectiveTagCount(t_1, r_2) = \max(0,\, 7 - \gamma \times 2) = \{6.5, 6, 5.5, 5\}$$
$$effectiveTagCount(t_2, r_2) = \max(0,\, 15 - \gamma \times 10) = \{12.5, 10, 7.5, 5\}$$
$$effectiveTagCount(t_4, r_2) = \max(0,\, 20 - \gamma \times 1) = \{19.75, 19.5, 19.25, 19\}$$
$$effectiveTagCount(t_5, r_2) = \max(0,\, 23 - \gamma \times 3) = \{22.25, 21.5, 20.75, 20\}$$

Tag Similarity/Distance What follows is the tag distance calculations for all four gamma parameters. Unless noted below, the tag sums for t3, t4, and t5 are as follows:

$$tagSum(t_3, r_1, r_2) = tagSum(t_4, r_1, r_2) = tagSum(t_5, r_1, r_2) = 1$$

Tag Distance for$\gamma = 0.25$

$$tagSum(t_1, r_1, r_2) = ((22 + 6.5) \times \frac{6.5}{22}) = 8.42$$
$$overlapTagSum(t_1, r_1, r_2) = 8.42 \times \frac{6.5}{22} = 2.49$$
$$tagSum(t_2, r_1, r_2) = ((42 + 12.5) \times \frac{12.5}{42}) = 16.22$$
$$overlapTagSum(t_2, r_1, r_2) = 16.22 \times \frac{12.5}{42} = 4.83$$
$$tagSimilarity(r1, r2) = \frac{2.49 + 4.83}{8.42 + 16.22 + 1 + 1 + 1} = \frac{7.32}{27.64}$$
$$tagDistance(r1, r2) = 1 - \frac{7.32}{27.64} = 0.735166425$$

Tag Distance for$\gamma = 0.50$

$$tagSum(t_1, r_1, r_2) = ((19 + 6) \times \tfrac{6}{19}) = 7.89$$

$$overlapTagSum(t_1, r_1, r_2) = 7.89 \times \frac{6}{19} = 2.49$$

$$tagSum(t_2, r_1, r_2) = ((42 + 10) \times \frac{10}{42}) = 12.38$$

$$overlapTagSum(t_2, r_1, r_2) = 12.38 \times \frac{1}{10} = 2.95$$

$$tagSimilarity(r1, r2) = \frac{2.49+2.95}{7.89+12.38+1+1+1} = \frac{5.43}{23.27}$$

$$tagDistance(r1, r2) = 1 - \frac{5.43}{23.27} = 0.766652342$$

Tag Distance for$\gamma = 0.75$

$$tagSum(t_1, r_1, r_2) = ((16 + 5.5) \times \frac{5.5}{16}) = 7.39$$
$$overlapTagSum(t_1, r_1, r_2) = 7.39 \times \frac{5.5}{16} = 2.54$$
$$tagSum(t_2, r_1, r_2) = ((42 + 7.5) \times \frac{7.5}{42}) = 8.83$$
$$overlapTagSum(t_2, r_1, r_2) = 8.83 \times \frac{7.5}{742} = 1.58$$
$$tagSum(t_3, r_1, r_2) = ((0 + 0) \times \frac{1}{0}) = 0$$
$$tagSimilarity(r1, r2) = \frac{2.54 + 1.58}{7.39 + 8.83 + 1 + 1} = \frac{4.12}{18.12}$$
$$tagDistance(r1, r2) = 1 - \frac{4.12}{18.12} = 0.772626931$$

Tag Distance for$\gamma = 1$

$$tagSum(t_1, r_1, r_2) = ((13 + 5) \times \frac{5}{13}) = 6.92$$
$$overlapTagSum(t_1, r_1, r_2) = 6.92 \times \frac{5}{13} = 2.66$$
$$tagSum(t_2, r_1, r_2) = ((42 + 5) \times \frac{5}{42}) = 5.60$$
$$overlapTagSum(t_2, r_1, r_2) = 5.60 \times \frac{5}{42} = 0.667$$
$$tagSum(t_3, r_1, r_2) = ((0 + 0) \times \frac{1}{0}) = 0$$
$$tagSimilarity(r1, r2) = \frac{2.66 + 0.667}{6.92 + 5.60 + 1 + 1} = \frac{3.327}{14.52}$$
$$tagDistance(r1, r2) = 1 - \frac{3.327}{14.52} = 0.770867768$$

Summary We see that by varying the gamma value we can get a varied tagDistance, with the four results being roughly in the same area of 0.77.