Semantic-aware news feeds management framework

Fekade Getahun Taddesse

UNIVERSITE DE BOURGOGNE

UFR Sciences et Techniques

THÈSE

Pour obtenir le grade de

Docteur de l'Université de Bourgogne

Spécialité: Informatique

---

par

Fekade Getahun TADDESSE

Soutenue le 30 novembre 2010

Framework de gestion sémantique de flux d'actualités (Semantic-aware News Feeds Management Framework)

Devant le jury composé de

Lionel BRUNIE	Professeur (INSA de Lyon, Lyon)	Rapporteur
Ahmed LBATH	Professeur (Université Joseph Fourier, Grenoble)	Rapporteur
Bernd AMANN	Professeur (LIP6, Paris)	Examinateur
Richard CHBEIR	Maître de Conférences - HDR (Université de Bourgogne, Dijon)	Co-directeur de thèse
Kokou YETONGNON	Professeur (Université de Bourgogne, Dijon)	Directeur de thèse

Laboratoire Électronique, Informatique et Image – LE2I
Equipe Ingénierie Informatique et Bases de Données

A ma mère,

A mon épouse et mes enfants

To my mother,

my wife and my children

REMERCIEMENTS

Pendant ma thèse j'ai eu la chance de recevoir une aide précieuse, une assistance et des conseils d'un certain nombre de personnes sans lesquelles, cette recherche aurait été plus difficile.

La première personne que je tiens à remercier est mon directeur de thèse, M. Kokou YÉTONGNON pour avoir accepté de m'accueillir au sein de son équipe de recherche. Je le remercie pour ses encouragements, ses conseils, sa confiance, et pour le temps passé à l'établissement de ce travail.

J'adresse tout particulièrement ma reconnaissance à mon co-directeur de thèse, M. Richard CHBEIR, pour ses précieux conseils, commentaires, aides, pour son engagement dans l'amélioration du travail et pour le temps passé à l'examen du rapport. En plus d'être un excellent conseiller, Richard est le plus proche comme un ami et une famille, c'est un plaisir de le connaître. Je voudrais également remercier son épouse, Mme Nathalie CHBEIR et leurs enfants pour leur chaleureuse hospitalité.

Mes sincères remerciements à la coopération franco-éthiopienne pour avoir participée au financement de ma thèse et m'avoir ainsi permis de poursuivre mon doctorat en France. Je remercie M. Solomon ATNAFU, M. Dejene EJIGU, M. Abye TASSE et M. Dawit BEKELE pour leur aide illimitée en vue de faciliter cette coopération. Cette thèse ne serait pas concevable que si ce n'était pas pour les bases solides et de la coopération initiée par Dawit et suivi dans le même rythme par Solomon et Dejene.

Je remercie M. Lionel BRUNIE (INSA de Lyon) pour son soutien illimité à la coopération franco-éthiopienne, sa gentillesse, ce privilège qu'il me fait en acceptant de rapporter mon travail et toutes ses remarques constructives. Je remercie M. Ahmed LBATH (Université Joseph Fourier, Grenoble) d'avoir accepté de rapporter mon travail de thèse ainsi que M. Bernd AMANN (LIP6, Paris) pour faire partie de mon jury de thèse.

Pendant mon séjour à l'Université de Bourgogne, j'ai eu l'occasion de parler, d'échanger des idées et de recevoir des critiques sur mon travail de la part de nombreuses personnes. Je tiens particulièrement à remercier Joe TEKLI, Elie RAAD, Mônica RIEIRO PORTO FERREIRA, Marco VIVANI, Gilbert TEKLI, Bechara AL BOUNA, Sylvain Valérie RAKOTOMALALA, Elie ABI-LAHOUD et Yi LUO. Je voudrais également remercier mes collègues du labo Guillermo Valente CARPIO GOMEZ, Damien LEPROVOST, Aries MUSULMAN et Raji GHAWI. Je remercie M. Christophe NICOLLE et M. David GROSS-AMBLARD pour leurs bons conseils, leurs aides et leurs motivations. Je remercie également Mlle Mylène KOSCIELINSKI et sa famille pour tous leurs encouragements et pour leur hospitalité.

Je suis également reconnaissant à tous mes amis et les membres de leurs familles qui m'ont toujours été une source constante d'encouragement à la fois pour moi et pour ma famille. Je remercie ma mère Amsale Kidane, mon frère Dereje, mes sœurs Fanos and Mistre pour tout.

Enfin, et pas des moindres je tiens à remercier mon épouse bien-aimée, Ziyen Gedlu, pour sa patience, son amour et son soutien qu'elle m'a apporté. Je suis redevable à nos petits et très beaux enfants, Belen et Robel, qui n'ont pas pu obtenir l'amour, l'orientation, les soins de leur père au moment où ils en avaient le plus besoin. Je remercie «Aket» (Atklete Zegeye) d'avoir bien pris soin des enfants pendant toutes ces années.

ACKNOWLEDGEMENT

During my PhD thesis, I received help, support and advice from a number of people without them this research could have been much harder.

First of all, I like to thank my supervisor, Prof. Kokou YÉTONGNON, for his confidence to accept me as his student. I thank him for his encouragement, advice, trust, support and for his time spent in reviewing this work.

My humble gratitude goes to my co-supervisor, Dr. Richard CHBEIR, for his invaluable guidance, comments, help, advice, commitment in improving the work and the time spent in reviewing the report. Aside from being an excellent advisor, Richard is as close as a friend and a family, it is a pleasure to know him. I would like to thank also his wife, Natalie Richard and their children for their warm hospitality.

My sincere thanks to the Franco-Ethiopian cooperation to have offered me the chance and the finance for perusing my PhD study in France. I thank Dr. Solomon ATNAFU, Dr. Dejene EJIGU, Dr. Abye TASSE and Dr. Dawit BEKELE for their unlimited help in facilitating this cooperation. This thesis wouldn't be conceived if it wasn't for the strong foundation and the cooperation initiated by Dawit and followed in the same pace by Solomon and Dejene.

I thank Prof. Lionel BRUNIE (INSA de Lyon, Lyon) for his unlimited support in Franco-Ethiopia cooperation, kindness, and the hard-work in keeping track of the progress of this work with constructive critics and also for the privilege to be a rapporteur in my thesis. I thank Prof. Ahmed LBATH (University of Joseph Fourier, Grenoble) to have agreed to be rapporteur and Prof. Bernd AMANN (LIP6, Paris) for being part of my PhD examination committee of my thesis.

During my stay in the University of Bourgogne, I had the opportunity to talk, share ideas and receive comments on my work from many people. Especially, I thank Joe TEKLI, Elie RAAD, Mônica Ribeiro Porto FERREIRA, Marco VIVIANI, Gilbert TEKLI, Bechara AL BOUNA, Sylvain Valérie RAKOTOMALALA, Elie ABI-LAHOUD and Yi LUO. I would like also to thank colleagues in the laboratory Guillermo Valente GOMEZ CARPIO, Damien LEPROVOST, Aries MUSLIM and Raji GHAWI. I thank Prof. Christophe NICOLLE and Dr. David GROSS-AMBLARD for their kind advice, help and motivation. I thank Ms. Mylène KOSCIELINSKI and her families for all the encouragements and hospitality.

I am also grateful to all my friends and family members who have always been a constant source of encouragement both for me and my family. I thank my mother Amsale Kidane, my brother Dereje, my sisters Fanos and Mistre for everything.

Last but not least, I would like to thank my beloved wife, Ziyen Gedlu, for her patience, love and support. I am indebted to our little and beautiful kids, Belen and Robel, who were unable to get the love, guidance, care of their father at the time they need the most. I thank "Aket" (Atklete Zegeye) for taking good care of the kids during all these years.

RÉSUMÉ

Dans le monde du Web, on retrouve les formats RSS et Atom (feeds) qui sont, sans doute, les formats XML les plus populaires et les plus utilisés. Ces formats permettent aux, entre autres, communautés Web, industriels, et services web de publier et d'échanger des documents XML. En outre, ils permettent à un utilisateur de consulter librement des données/informations sans avoir à basculer d'un site à un autre, et cela à l'aide d'applications logicielles. Dans ce cas, l'utilisateur enregistre ses fournisseurs de flux favoris, chaque fournisseur diffuse la liste des nouveaux éléments qui ont été modifiés depuis le dernier téléchargement. Cependant, l'enregistrement d'un certain nombre de sources de flux dans un agrégateur de flux engendre à la fois des problèmes d'hétérogénéité (à cause des différences structurelles et de contenu) et des problèmes de surcharges d'information. Par ailleurs, aucun des agrégateurs de flux existants n'offre une approche qui intègre (ou fusionne) les flux en tenant compte de leurs similarités, du contexte de l'utilisateur et de ses préférences.

Dans cette thèse, nous proposons un framework formel qui permet de traiter l'hétérogénéité, l'intégration et l'interrogation des flux d'actualités. Ce framework est fondé sur une représentation arborescente d'un flux et possède trois éléments principaux qui sont les suivants: *comparateur de flux*, *intégrateur de flux*, et *processeur de requêtes*.

Le *comparateur de flux* permet de mesurer le degré de similarité entre deux éléments/flux en utilisant une base de connaissance intégrant une approche ascendante et progressive. Nous proposons une mesure de similarité à base de concept capable de calculer la similarité entre les flux selon le nombre de leurs concepts communs (et différents) et leurs proximités sémantiques. Nous montrons également comment définir et identifier la relation exclusive entre deux textes ou éléments.

L'*intégrateur de flux* permet de fusionner plusieurs flux provenant de différentes sources tout en tenant compte du contexte de l'utilisateur. Nous montrons dans notre étude comment représenter le contexte d'utilisateur ainsi que ses préférences. Nous fournissons un ensemble prédéfini de règles de fusion qui peuvent être enrichies et adaptées par chaque utilisateur.

Quant au *processeur de requêtes*, il se base sur une étude formelle et plus précisément sur une algèbre dédiée à la fusion des flux continus d'actualités que nous proposons ici. Les opérateurs proposés dans cette algèbre sont aidés par des fonctions à base de similarité. Nous catégorisons les opérateurs de flux selon trois catégories: opérateurs d'extraction, opérateurs ensemblistes et opérateur de fusion. Nous montrons que l'opérateur de fusion généralise l'opération de jointure et les opérateurs ensemblistes. Nous fournissons également un ensemble de règles de réécriture et d'équivalence de requêtes pour la simplification et l'optimisation des requêtes.

Enfin, nous présentons un prototype nommé «Easy RSS Manager» (EasyRSSManager). Ce prototype est un lecteur sémantique de flux et un composant sémantique pour l'interrogation des fenêtres de flux. EasyRSSManager a été utilisé pour valider, démontrer et tester la faisabilité des différentes propositions de notre étude. En particulier, nous avons testé la complexité en temps et la pertinence de nos approches en utilisant à la fois des données réelles et syntaxique.

MOTS-CLÉS:

Similarité des flux, proximité sémantique de flux, voisinage sémantique, règle de fusion, intégration de flux, opérateurs de similarité, algèbre RSS, requête de flux, réécriture de requête

ABSTRACT

In the Web, RSS and Atom (feeds) are probably the most popular and highly utilized XML formats which allow web communities, publishing industries, web services, etc. to publish and exchange XML documents. In addition, they allow a user to consume data/information easily without roaming from site to site using software applications. Here, the user registers her favorite feed providers; and each provider sends the list of news items changed since the last download. However, registering a number of feed sources in feed aggregators cause both heterogeneity and information overloading problems. Besides, none of the existing RSS/feed aggregators provide an approach that integrates (merges) feeds from different sources considering similarity, user contexts and preferences.

In this research, we provide a formal framework that handles the heterogeneity, integration and querying feeds. The framework is based a tree representation of a feed and has three main components: *feed comparator*, *merger* and *query processor*.

The *feed comparator* addresses the issue of measuring the relatedness between news items using a Knowledge Base, a bottom-up and incremental approaches. We proposed a concept-based similarity measure based on the function of the number of shared and different concepts in their global semantic neighborhoods. Here, we use the concept similarity value and relationship as a building block for texts, simple elements and items relatedness algorithms. We show also how to define and identify the exclusive relationship between any two texts and elements.

The *feed merger* addresses the issue of integrating news items from different sources considering a user context. We show here how to represent a user context and her preferences. Also, we provide a set of pre-defined set of merging rules that can be extended and adapted by a user.

The *query processor* is based on a formal study on RSS query algebra that uses the notion of semantic similarity over dynamic content. The operators are supported by a set of similarity-based helper functions. We categorize the RSS operators into extraction, set membership and merge operators. The merge operator generalizes the join and the set membership operators. We also provide a set of query rewriting and equivalence rules that would be used during query simplification and optimization.

Finally, we present a desktop prototype called Easy RSS Manager (*EasyRSSManager*) having a semantic-aware RSS Reader, and semantic-aware and window-based RSS query components. It is designed to validate, demonstrate and test the practicability of the different proposals of this research. In particular, we test the timing complexity and the relevance of our approaches using both a real and syntactic dataset.

KEYWORDS:

Feed similarity, feed relatedness, semantic relatedness, semantic neighborhood, relationship-aware clustering, merging rule, rule-based feed merging, feed query, query rewriting, RSS algebra, feed query

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1
INTRODUCTION

1.1 Introduction

Since 1998, Extensible Markup Language (XML) has been recognized as an international standard for both data formatting, representation and exchange of web data.

Thanks to XML, nowadays, the Web is more than being a read only interconnected collection of web pages. It is rather a collection of distributed and heterogeneous, read/write documents. In particular, the Web 2.0 technologies revolutionize the way people work by providing facilities to create, share, collaborate and communicate without acquiring solid background in web design. Consequently, user's participation in the Web is no longer limited to only browsing but goes beyond. The Web 2.0 empowers users to collaborate using wikis, to share idea and commentary information with blogs, to create and work in social community using social networks, and to notify updates using RSS. In addition, Web 2.0 allows content hosting, tagging, bookmarking and data mashing. According to blog search engine BlogPulse[1], daily around 43,000 blogs are created; currently, there are a total of 126, 861,574 blogs, out of which 1,090,504 blogs per day are active.

RSS and Atom (RSS ADVISORY BOARD, 2009; HAMMERSLEY, B., 2003) are the two popular content syndication web feed formats and technologies that make blogs very

[1] http://www.blogpulse.com/

popular. A web feed exists in various versions and formats (RSS 0.91[2] and 0.92, RSS 1.0[3], RSS 2.0 and Atom[4] 1.0). As a data format, a web feed also called news feed or feed is a machine-readable XML file that allows web sites, content owners, media outlets and bloggers to share their content with other applications in a standardized way. As a technology, the web feed provides a method for getting relevant and up-to-date information to users. Due to these facts, the number of applications using web feeds are increasing everyday: *AmphetaDesk*[5], *PullRss*[6], *Radio UserLand*[7], *SlashCode*/*Slashdot*[8], *Weblog 2.0* (HAMMERSLEY, B., 2003). Noticing the advantages and the new trends existing legacy web pages/articles are transformed into web feed (WANG, J. et al., 2006; NANNO, T. and Okumura, M., 2006) using time pattern discovery and tag pattern mining.

Recently, web users are shifting to web feed for three main reasons:

- **Behavior of feed**: in essence, feed is proposed to facilitate the aggregation of distributed and dynamic information. As the content is in XML format, software tools also known as RSS/feed readers/aggregators (which can be either web-based application e.g., Google Reader, client–oriented e.g., Microsoft Office outlook, or plug-in to Web Browser) allow a user/client to subscribe, read, and access feed content originating from different providers in a place rather than roaming site to site.

---

[2] RSS 0.92 is upward compatible with RSS 0.91 Userland specification http://backend.userland.com/rss09x (where x = 1 or x = 2)

[3] RSS 1.0 is also called RDF Site summary. It is a lightweight multipurpose extensible metadata description and syndication format conforms to the W3C's RDF Specification and is extensible via XML-namespace and/or RDF based modularization. More detail can be found at: http://web.resource.org/rss/1.0/spec

[4] Atom is an XML-based document format that describes lists of related information known as "feeds". Feeds are composed of a number of items, known as "entries", each with an extensible set of attached metadata. More detail can be found at: http://tools.ietf.org/id/draft-ietf-atompub-format-11.txt

[5] AmphetaDesk is a free, cross platform, open-sourced, syndicated news aggregator available at http://www.disobey.com/amphetadesk/

[6] PullRSS is a template-based RSS to HTML converter, with optional redirects.

[7] http://radio.userland.com/userGuide/reference/aggregator/newsAggregator

[8] http://slashdot.org/

- **Feed is everywhere**: feed is integrated as part of the new web applications such as web blogs and content sharing applications (e.g., YouTube[9], wiki, twitter[10], etc.) to notify changes and update operations. Hence, it is an opportunity for a user to fuse/mashup existing feeds and generates new feeds.
- **Streaming nature**: compared to web documents, web feeds are dynamic in nature. Web feed is a web document in which the content providers are set up to send out notification whenever new materials are available. Hence, the content is available immediately to the feed reader and also to feed search engines. In contrast, web documents/articles are only accessible to public once after it is found by a crawler and indexed by search engines. For instance, according to Golding (GOLDING, A., 2008), the Google News crawler is configured to visit each article's URL only once per day. Hence, a new development or news update wouldn't be visible to users.

However, when clients/users add more and different sources to their feed readers, the amount of news feeds becomes more difficult to manage. This causes the heterogeneity and data/information overload problems[11]. As a result, clients have to read related (and even identical) news more than once as the existing feed engines do not provide facilities for identifying similar feeds. Because of the specific characteristics of web feed, the major challenges for the research community revolve around providing a dedicated similarity measures, a personalization and human computer interaction option, and dedicated operators.

The next section presents these challenges through a set of motivating examples.

[9] http://gdata.youtube.com/feeds/base/videos?q=querystring&client=ytapi-youtube-search&v=2 returns the list of YouTube videos containing the full text "querystring" as RSS feeds.
[10] http://twitter.com/
[11] It refers to the difficulty in making decision caused by lot of information about the same issue.

1.2 Motivation

To motivate our work, let us consider Figure 1.1 and Figure 1.3 showing a list of news extracted from CNN and BBC's RSS feeds. Registering these feeds in existing news readers (such as *Newsgator*, *Google Reader*, *Attensa*) provides the user with access to all news without considering *relatedness* among them. However, identifying and merging related news would enable the user to easily and efficiently acquire information. The user would obviously prefer to access one piece of news about a certain topic, encompassing all relevant and related information (after merging), instead of searching and reading all news articles covering the same topic, which could be extremely time consuming and often disorienting. When the number of registered feeds increases, the need to have a specialized, adaptive, semantic-based RSS querying language is unquestionable. The following scenarios show the reasons and failures of the existing solutions to address user requirements and demonstrate the need for a dedicated RSS framework.

Scenario 1: Semantic relatedness

On one hand feed exists in different version and formats. Table 1.1 shows some of the corresponding elements defined in RSS 2.0 and Atom 1.0. On the other hand, the content heterogeneity is due to the difference in author's culture, writing skill, wordings, etc. This leads to having different contents referring to the same fact.

Hence, a feed based similarity measure has to handle these two problems.

Identifying the similarity/relatedness between news items is a pre-condition in the design of different applications such as merger, and revision control. Herewith, we present the specific cases that should be considered while measuring similarity:

Table 1.1: Comparison between RSS 2.0 and Atom 1.0, extracted from **(BRAY, T., 2005)**

RSS 2.0	Atom 1.0	Comment
rss		Root element in RSS
channel	feed	
title	title	
description	subtitle	
language		xml:lang attribute in atom
item	entry	
description	summary and/or content	Depending on whether full version is provided
guid	id	
link	link	
pubDate	published (in entry)	Atom has no feed level equivalent to pubDate
lastBuildDate(in channel)	Updated	RSS has no feed level update dateTime equivalence

```
<CNN_RSS>
  <item>
    <title>Ministers among Somalia blast dead</title>
    <guid>http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition</guid>          CNN1
    <link>http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition</link>
    <description>An explosion at a graduation ceremony in the Somali capital Thursday killed at least 15 people,
      including three government ministers and nine students, local journalists told CNN.</description>
    <pubDate>Thu, 03 Dec 2009 07:27:47 EST</pubDate>
  </item>
  <item>
    <title>Bin Laden not in Pakistan, PM says</title>
    <guid>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</guid>     CNN2
    <link>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</link>
    <description>Pakistan's prime minister Thursday rejected claims Al Qaeda leader Osama bin Laden is hiding
      within his country as global pressure mounted on Islamabad to tackle terrorists linked to escalating conflict
      in neighboring Afghanistan.</description>
    <pubDate>Thu, 03 Dec 2009 06:23:16 EST</pubDate>
  </item>
  <item>
    <title>Bus blast kills, hurts dozens in Syria</title>
    <guid>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</guid>          CNN3
    <link>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</link>
    <description>An explosion killed dozens of passengers in a bus in the Syrian capital Thursday morning,
      officials said.</description>
    <pubDate>Thu, 03 Dec 2009 06:03:40 EST</pubDate>
  </item>
  <item>
    <title>U.N. chief launches $613M Gaza aid appeal</title>
    <guid>http://edition.cnn.com/2008/WORLD/asiapcf/05/02/oly.hk.torch/index.html?eref=edition</guid>          CNN4
    <link>http://edition.cnn.com/2008/WORLD/asiapcf/05/02/oly.hk.torch/index.html?eref=edition</link>
    <description> United Nations Secretary-General Ban Ki-moon on Thursday launched a humanitarian appeal
    to provide emergency aid to the people of Gaza in the aftermath of Israel's military offensive in the
    region.</description>
    <pubDate>Fri, 02 January 2009 02:56:47 EDT</pubDate>
  </item>
  <item>
    <title>Al-Jazeera: Cameraman home from Gitmo</title>
    <guid>http://edition.cnn.com/2008/WORLD/americas/05/01/gitmo.journalist/index.html? eref=edition
    </guide>
    <link>http://edition.cnn.com/2008/WORLD/americas/05/01/gitmo.journalist/index.html? eref=edition</link>    CNN5
    <description>Al-Jazeera cameraman Sami al-Hajj has been released after nearly six years in the U.S. Navy
    prison at Guantanamo Bay, Cuba, a senior Pentagon official aware of the details of the release told CNN on
    Thursday.</description>
    <pubDate>Thu, 01 May 2008 21:51:15 EDT</pubDate>
  </item>
</CNN_RSS>
```

Figure 1.1: Sample news items from CNN

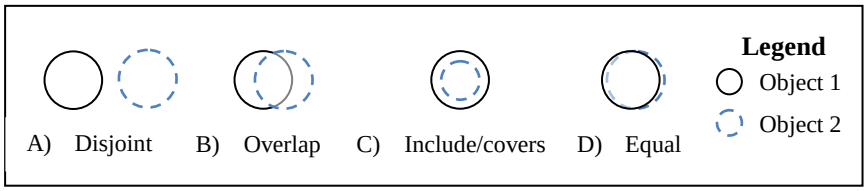


Figure 1.2: Minimum set of relationships between objects- texts or elements

```
<BBC_RSS>
  <item>
    <title>Somali ministers killed by bomb</title>
    <description>A suicide bomber disguised as a woman kills at least 19 people, including government ministers, at a hotel in the Somali capital.</description>                BBC1
    <link>http://news.bbc.co.uk/go/rss/-/2/hi/africa/8392468.stm</link>
    <guid isPermaLink="false">http://news.bbc.co.uk/2/hi/africa/8392468.stm</guid>
    <pubDate>Thu, 03 Dec 2009 13:24:49 GMT</pubDate>
  </item>
  <item>
    <title>Bin Laden not in Pakistan, says PM</title>
    <description>Pakistan's prime minister tells UK counterpart Gordon Brown he does not think Osama Bin Laden is in his country.</description>                BBC2
    <link>http://news.bbc.co.uk/go/rss/-/2/hi/uk_news/politics/8392211.stm</link>
    <guid isPermaLink="false">http://news.bbc.co.uk/2/hi/uk_news/politics/8392211.stm</guid>
    <pubDate>Thu, 03 Dec 2009 11:05:52 GMT</pubDate></item>
  </item>
  <item>
    <title> UN launches $613m appeal for Gaza </title>
    <description> The UN will launch an appeal for $613m to help people affected by Israel's military offensive in Gaza, the body's top official says </description>                BBC3
    <guid isPermaLink="false">  http://news.bbc.co.uk/go/rss/-/2/hi/me/723378828.stm </guid>
    <link> http://news.bbc.co.uk/go/rss/-/2/hi/americas/7378828.stm </link>
    <pubDate>Fri, 02 January 2009 02:56:47 GMT</pubDate>
    <category>Middle-east</category></item>
  </item>
  <item>
    <title>Freed Guantanamo prisoner is home</title>
    <description>A cameraman from the al-Jazeera TV station freed from Guantanamo Bay has arrived home in Sudan.</description>                BBC4
    <link>http://news.bbc.co.uk/go/rss/-/2/hi/americas/7378828.stm</link>
    <guid isPermaLink="false">http://news.bbc.co.uk/2/hi/americas/7378828.stm</guid>
    <pubDate>Fri, 02 May 2008 04:08:38 GMT</pubDate>
    <category>Americas</category><item>
  </item>
</BBC_RSS>
```

Figure 1.3: Sample RSS news items extracted from BBC

1. The content of an element might be identical or similar to another element (equality in Figure 1.2.D)

   Example 1.1: **Equal news**: The title element of CNN2, <title>Bin Laden not in Pakistan, PM says</title>, and title of BBC2, <title>Bin Laden not in Pakistan, says PM</title>, are identical as both share the same concepts i.e., concepts in the content of CNN2 are also concepts of BBC2 and vice versa.

2. The content of an element might be similar and totally included in another element (inclusion in Figure 1.2.C)

   Example 1.2: **Including news**: The title content of CNN4, "U.N. chief launches $613M Gaza aid appeal", includes the title content of BBC3, "UN launches $613m appeal for Gaza"[12].

3. Two news items may refer to similar and related concepts (overlapping in Figure 1.2.B)

   Example 1.3: **Overlapping news**: The title element of CNN1, <title>Ministers among Somalia blast dead</title>, and title of BBC1, <title>Somali ministers killed by bomb</title>, share common concepts. Their content shares identical concept '*Minister'* and related concepts '*Somalia*' and '*Somali'*, '*kill*' and *'dead'*.

4. News might have different or slightly different titles but refer to almost the same issues

   Example 1.4: Similarity between different elements: The title content of CNN5, "Al-Jazeera: Cameraman home from Gitmo", and the title content of BBC4, "Freed Guantanamo prisoner is home", share little (i.e., common concepts are "*home*" and "*Guantanamo*"[13]). However, the contents of corresponding news items are similar.

5. A news item may not share anything with another news item (case of disjoint relationship in Figure 1.2.A)

[12] After a pre-process of stop word removal, stemming, ignoring non textual values and semantic analysis.
[13] "Gitmo" indicates the Guantanamo prison.

These four examples demonstrate the need to consider two issues when comparing RSS items:

1) the need to consider the content of elements having different labels as computing relatedness between contents of elements having identical labels is not enough to identify the overall items relatedness (c.f. Example 1.4).
2) the need to identify the relationships (i.e., *disjointness, overlap/intersection, inclusion*, and *equality* c.f. Example 1.1 to Example 1.4), which have never been considered in any of the existing XML-related (*xSim* (KADE, A. M. and Heuser, C. A., 2008)), flat texts similarity approach such as *tf-idf* (MCGILL, M. J., 1983), or RSS oriented correlation-based phrase matching approaches (PERA, M. S. and Ng, Y, 2007)

It is to be noted that identifying the items relatedness is complex as the quality of textual information is dependent on the author's style of writing and use of words, nouns, verbs, etc. (identical topics might be described differently, while different topics might be described using similar concepts).

Scenario 2: Context-aware merging of news items

Alice, a medical doctor, registers all her favorite medical news feeds, blogs and result of searching[14] medical journal (e.g., PubMed) and medical RSS search engines (e.g., RSS4Medics) in her RSS reader. She uses her RSS reader from her personal computer at home or a portable computer (PDA, Smartphone, etc.) during coffee break.

When using her personal computer, she likes to read the different perspectives of each article. However, during the coffee time, she prefers to read only the latest of similar news items, a news item that includes/generalizes other news; otherwise, keep the different perspectives of each article.

[14] http://www.rss4medics.com, http://www.medworm.com

This scenario shows the need to:

1) identify the context of Alice which includes location (where she is e.g., at home, in her office, etc.), the type of device she is using (e.g., PC, Smartphone, etc.)
2) measure the relatedness between news items
3) identify the set of actions that fits with the preferences of Alice (e.g., keep the latest of similar news items, keep a news item that includes/generalizes other news, keep both news, etc.) and
4) have an easy and adaptive system.

Scenario 3: **Semantic-based RSS operators**

Registering a number of news feeds in a RSS aggregator often causes data overloading problem. One of a known solution to alleviate this problem is the use of query operators. The content of web feed flows periodically as per the updating rule of the content owner. Liu et al, in their RSS survey (LIU, H. et al., 2005) reported that on average 55% (out of 100000 registered feeds in 45 days) update their content within 1 hour. Unlike the traditional database query processing, data is relatively static and the query is unknown, for stream query processing, data is relatively dynamic and the query is known. Thus, the query processing in stream is continuous over each arriving feed.

The following five examples demonstrate the need to have specialized RSS based querying operators.

> **Example 1.5 Joining feeds**: Bob, a journalist, wants to get all news items of CNN and BBC having similar titles and published between 5 and 7 o'clock on December 3, 2009 (for instance). This query involves joining set of news items of both sources within the given timestamp while considering the semantic information embedded in the title element.

One of the current approaches to handle this query is to use Nested Loop Join (NLJ) in XQuery[15] of news items using comparison expression defined on the content[16] of title elements such as in Figure 1.4. However, this wouldn't provide expected result as the comparison expression (in the `where` clause) is restricted to exact text matching and yet without semantic. As a result, news refereeing to the same fact (e.g., the pair of news CNN2 and BBC2) but written differently wouldn't be included in the final result.

```
for $I in docs(... /cnn.rss), $j in docs(../bbc.rss)
where fn:compare($I/title.content, $j/title. content) EQ 0 return
<result>{$I, $j} </result>
```

Figure 1.4: CNN Join BBC using NLJ

Another way to handle this problem is to use data mashup tools (such as Yahoo! Pipes which put all news items in the two sources). However, none of the existing mashup tools neither consider the timely nature of the feeds nor handle the semantic heterogeneity problem embedded in the content of news items.

> **Example 1.6 Merging feeds**: Bob wants also to retrieve all hourly news items published by CNN and BBC while keeping the redundant[17] news items.

Handling this query could be currently done using the Outer Nested Loop Join concept (ONLJ) of XQuery 1.1 with the joining comparison condition in the `where` clause of XML query as shown in Figure 1.5.

```
outer for $I in docs(... /cnn.rss), $j in docs(../bbc.rss)
where fn:contains($I/title.content , $j/title. content) = True
return <result>{$I, $j} </result>
```

Figure 1.5: CNN Outer join BBC

[15] XQuery (ROBIE, J. et al., 2009) is a query language based on tree for finding and extracting elements and attributes from XML documents.
[16] Given an element e, its content is accessed via *e.content*.
[17] A news item is the redundant of another news item if there is equality or inclusion relationship in-between.

However, doing that would cause the following drawbacks:

1. semantically identical news would be considered different (e.g., CNN2 and BBC2),
2. related news items (in particular those overlapping[18] or included such as CNN1 overlap with BBC1, and CNN4 include BBC3) won't be in the result set even if XPath[19] function `fn:contains` (CLARK, J. and DeRose, S., 1999) is used to consider the case of inclusion. Hence, users wouldn't apprehend the relationship existing between the news items and would be forced to read the related news independently as if they are different.

Example 1.7. Evolution of news item: Bob wants to do analysis on evolving[20] news items published by BBC and issues the query: get all news items of BBC that evolved in the last 24 hours.

To handle this query, one has to identify the *inclusion* relationship of two related news items over a period of time and merge them together. However, this hasn't been considered in any of the existing solutions, including Google News. The later provides only a timeline graph that shows the number of sources that cover a story (defined with a set of keywords) together with a change over time of articles. The news in the timeline shares only some keywords.

Example 1.8. **Query By Example and Query optimization**: Bob wants to retrieve all news items published within the last two hours by CNN and BBC and are similar to a given news item extracted from Reuters.

Handling this kind of queries requires performing:

[18] Two news are related with overlap relationship if both share some common data/information
[19] XPath (CLARK, J. and DeRose, S., 1999) is a query language based on tree used to navigate through nodes, elements and attributes in an XML document. It defines set of functions to manipulate simple values.
[20] A news items evolves if its updated version is published later on.

- similarity join over the result of similarity selection (that identify all news items similar to the given example) over each source, or
- selecting the result of joining news items from the two sources.

Even though these two query plans provide the same final result, the order of doing the operations generate different overall cost. Hence, there is a need to choose the plan with lesser cost.

In addition, Bob's query commonly called Query By Example – QBE is one of the basic operations in feed context but not handled with any of the current approaches. This type of query demands the need to have an easy to use user interface.

The last four examples (Example 1.5. to Example 1.8.) demonstrate the need to have specialized RSS operators that take into consideration the timely nature of the news feed (Example 1.5. to Example 1.8), relatedness/similarity (Example 1.5 and Example 1.8), relationship existing between texts and elements (such as *Equality*, *Inclusion*, *Overlapping*, and *Disjointness*) (Example 1.6 and Example 1.7) while considering semantic information to analyze their meaning. In addition, the QBE in Example 1.8, shows the need to have adaptive and easy to use user interface.

Hence, the main objectives of this thesis are:

1) Integrating semantic information in news feed management
2) Measuring the semantic relatedness between entities to be compared
3) Querying dynamic news items using semantic-aware and context-aware operators, and
4) Facilitating the news feed management using easy to use interface

The rest of this chapter is organized as follows. Section 1.3 provides an overview of our approach. Section 1.4 elicits the main contribution of this thesis work. Section 1.5 provides the roadmap of the report.

1.3 Overview of our approach

In this thesis, we propose the Semantic-Aware News Feeds Management Framework shown diagrammatically in Figure 1.6. Our Framework is composed of three main and interacting components: RSS relatedness, Merger and RSS query processor.

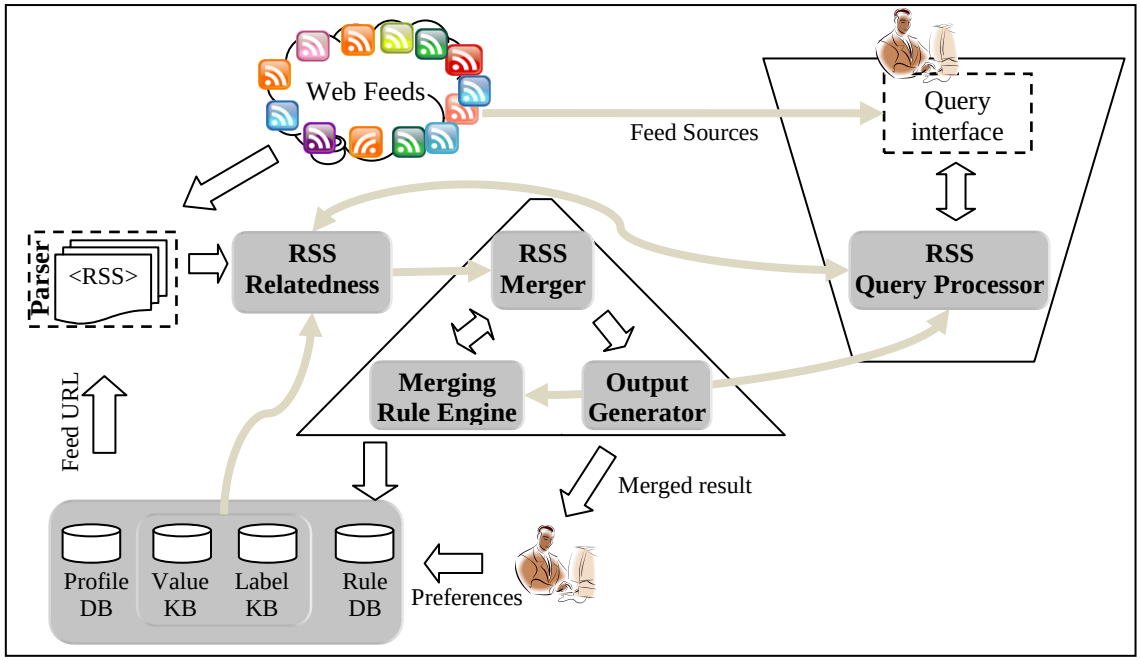


Figure 1.6: Semantic-aware feeds management framework

The RSS relatedness (c.f. Chapter 3 for detail) measures the extent to which two feed contents are related using two types of Knowledge Bases (value and label), to handle both structural and content heterogeneity problems, and return a pair containing similarity and relationship values. The relatedness between feed contents is computed by combining the relatedness between its components, texts and elements, using both mathematical and heuristic based aggregation approaches. For instance, we compute the similarity between textual values using the cosine of the angle separating the vectors representing the components of each text. Each vector contains the weight of word/concept computed using our enclosure similarity reflecting concept occurrence and maximum similarity.

The enclosure similarity between two words (one from each text) is computed as the ratio of the number of concepts/words shared in the neighborhood (collection of semantically related concepts) of each word over the neighborhood of the second word. The relationship between textual values is identified using notion of interval defined on similarity and two threshold values – disjointness and equality. Then, the relatedness between elements is computed by combining the relatedness between labels and contents.

The RSS merger (c.f. Chapter 4 for detail) provides an adaptive and easily customizable rule-based feed integration approach. The rules are both pre-defined and can be personalized later by the user. The rule engine extracts rules personalized by the user (stored in rule database) and informs the merger what to do when collection of feed contents satisfying known conditions are found. The RSS merger sent its result to output generator to produce a result in the format suggested by the user.

The RSS Query processor (c.f. Chapter 5 for detail) processes continuous query using a set of semantic- and threshold- based operators that accept window(s) as input. The processor interprets a user query string as RSS content (i.e., text or element) and computes the corresponding similarity between the query string and each member of the window(s) in collaboration with the RSS relatedness component. The proposed operators solve the issue of querying dynamic and author dependent textual information.

1.4 Contributions

The major contributions of this thesis are the following:

1. we propose dedicated RSS relatedness measures able to compute similarity and identify relationship at different levels of granularity – texts, or elements
2. we propose dedicated RSS algebra composed of set of similarity functions and extraction operators. The algebra contains a novel operator called *Merge* that

generalizes the binary join, and the set membership operators; we show that only *select* and *Merge* operator are needed in feed context

3. We propose a context-aware and rule-based framework that allows the user to define rules, personalize sources and system parameters.
4. We develop a prototype –*EasyRSSManager*- to validate and demonstrate the practicability of the different proposals made in this thesis.
5. We test experimentally the relevance of our approaches using both real and synthetic news datasets.

1.5 Roadmap

This thesis reports is organized as follows:

Chapter 2 reviews the works related to the realm of our research problems. We review works in the area of concept similarity, XML comparison, merging and XML algebra.

Chapter 3 details our approaches to handle the heterogeneity problems and also to measure the relatedness between a pair of concepts, texts, and elements.

Chapter 4 details our context-aware and rule-based feeds merging approach. It discusses the merging framework with its components and the merging algorithm.

Chapter 5 details our dedicated feed query operators. We define a set of window-based and semantic-aware operators based on the feed data model. We study the property and query rewriting approach.

Chapter 6 presents our prototype –*EasyRSSManger* and the set of experiments conducted to validate our approaches.

Chapter 7 concludes the thesis report by drawing conclusions, contribution and our future research directions.

CHAPTER 2
RELATED WORKS

2.1 Introduction

It is to be recalled that a RSS news feed is text-content rich, semantically heterogenous and dynamic XML document. Hence, efficent retreival of news feed is related to the issue of measuring concept similarity, XML document comparision, aggregation or integration of XML documents and retrieval of XML documents.

The objective of this chapter is to investigate the different approaches in words/concept-based similarity measures, XML document similarity, merging/integration of XML documents, and querying XML database.

The rest of the chapter is organised as follows. In Section 2.2, we discuss the XML data model. In Section 2.3, we provide a review of industrial products related to news management. Section 2.4 assesses works related to concepts similarity measures. Section 2.5 reviews the three main approaches in XML documents comparsion. Section 2.5 reviews basic technique to integrate or merge in distributed database design and semi-structured/XML documents. In Section 2.7, we review works in XML query algebra. Finally, Section 2.8 summerizes the chapter.

2.2 XML data model

XML document represents hierarchically structured data. It can be modeled as either Ordered Labeled Tree (OLT) or Unordered Labeled Tree (UOLT). In both models, each node of the tree is an XML element and is written with an opening and closing tag. An element can have one or more XML attributes representing name-value pairs with element. An edge connecting nodes represents parent-child relationship. In OLT, the children of each node are ordered from left to right following their order of appearance in the document. In the work of (ZHANG, Z. et al., 2003; NIERMAN, A. and Jagadish, H. V., 2002), OLTs have been implemented using special and distinct ordering attributes names. The attributes nodes appear as first child of their encompassing element node, ordered by the attribute name (NIERMAN, A. and Jagadish, H. V., 2002). In the work of (SCHLIEDER, T. and Meuss, H., 2002), attributes of an element are transformed into two nodes related with parent-child relationship attached to the element. The parent element is named after the attribute name and the child is text node with sequence of words describing the value of the attribute.

XML documents may also have elements defining hyper-links or reference to other documents or elements (using XLINK[21], elements associated with ID, IDREF and/or IDREFS tokenized-attribute[22]). Including such links in the model gives rise to a graph rather than a tree and these links can be important in actual use of the XML data.

In the context of news feed document (HAMMERSLEY, B., 2003), *link, id* and *guide* elements contain unidirectional reference to external and actual document which doesn't change the definition of a tree. Consequently, we disregard reference/linkage between

[21] XLINK (DEROSE, S. et al., 2001) is a W3C specification that defines the XML Linking Language which allows elements to be inserted into XML documents in order to create and describe links between resources.

[22] A tokenized type attribute is specified using value of type ID, IDREF or IDREFS. ID attribute name is unique in an XML document and acts as unique identifier for the elements. IDREF or IDREFs have a value matching to the value of an ID attribute of some element in the XML document (BRAY, T. et al., 2006)

elements. Figure 2.1 shows a sample RSS feed and the equivalent tree. An element that contains only simple values is called simple element, otherwise it is complex element.

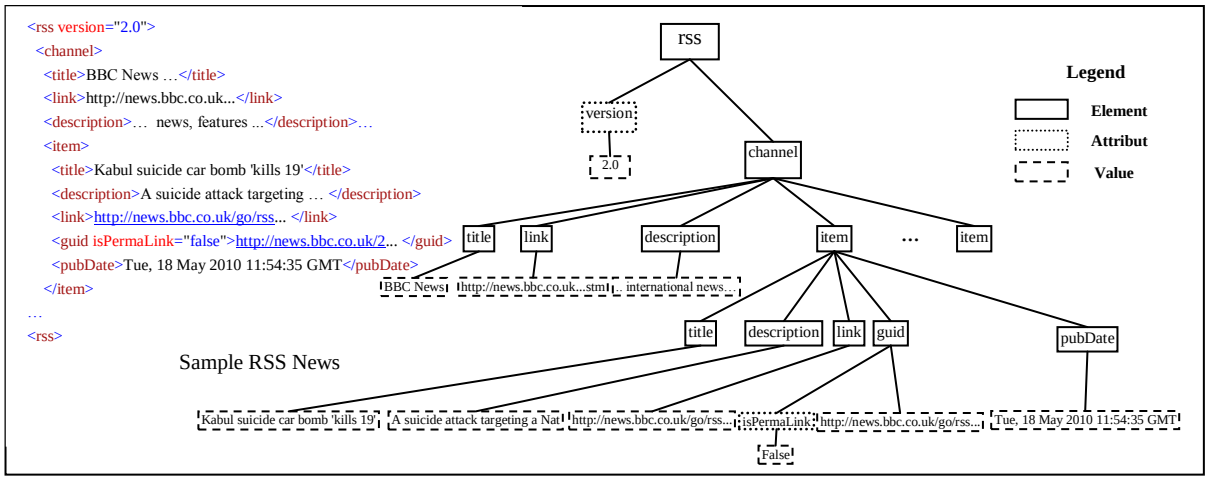


Figure 2.1: Tree representation of Sample news feed

2.3 Industrial products

The known commercial search engines like Google, Yahoo and Microsoft provide keyword-base news searching, aggregation of news from different sources, clustering and personalization services. In the next sub-section we present Google News, Yahoo! News and Microsoft's Bing News, followed by feed aggregators and data mashups.

2.3.1 Commercial news search engines

Google News[23]

Google News aggregates news articles from more than 4500 worldwide news sources, groups automatically similar ones together (using pre-defined clusters as Top Stories, U.S. Business, Sci/Tech, Entertainments, Sports, and Health), and displays them according to each user's personalized interest and/or news popularity. Google News

[23] http://news.google.com/

applies duplicate detection approach to show only the original stories from the source together with links to several news articles related to it. Even if Google News clusters the news articles, clicking on the option "all *n* news articles" shows all news items in which some of them are similar (even identical), related (i.e., share common information - overlap, and include) but readers have to read all to decide on what to do such as disregard them or not. Recently, Google News implements keywords-base trending of the popular news sorted in chronological order of recentness. The keywords-based searching of news articles is supported with a dedicated keyword-based inverted list. The inverted list index file is consulted to look for the candidate documents that contain the keywords.

Yahoo! News[24]

Yahoo! News provides similar service as Google News and aggregates more than 5000 news sources using semi-automatic method (i.e., combination of algorithmic and human editors). In addition, Yahoo provides trending on the popular news (identified with keywords) ordered on recentness. However, Yahoo doesn't allow personalization neither on the source nor preference of content.

Microsoft's News[25]

Microsoft's Bing News search engine provides the same service as Google News, and displays localized news depending on the user's location in the United States.

Table 2.1 shows the comparison between the three news search engines presented above. In general, the news search engines categorize the set of news into a set of pre-defined clusters, and navigation within the cluster is possible. In addition, the retrieval is keyword-based without similarity, and location-based personalization option. However, none of them provide a personalization option that assists a user on how to present those news articles in the same cluster.

[24] http://news.yahoo.com/
[25] http://www.bing.com/news

Table 2.1: Commercial news search engines with the supported operations

	Personalization		Features	Support for		
	type supported	Source base?		relationship	structured query	QBE
Google News	- keyword-base filtering	×	- Timeline of event - duplicate detection - keyword searching - automatic clustering to predefined clusters	×	×	×
Yahoo! News	- keyword-base filtering	×	- trending of event - keyword searching - semi-automatic clustering to pre-define clusters	×	×	×
Bing News	- location-base personalization	×	- keyword searching - clustering of news into predefined clusters	×	×	×

2.3.2 Feed aggregators

The existing RSS/feed aggregators focus mainly on the reformatting and displaying of news items without prioritizing, rearranging, merging, clustering, etc. Feedsifter[26] and FeedRinse[27] provide keyword-based filtering (either to allow or prohibit) of news items within a given feed but this approach is very tedious and not scalable to large scale.

Recently, in (BERGAMASCHI, S. et al., 2007) the authors presented a semantic news feed aggregator that group related news having same topic values. They applied clustering of the titles of the news feeds selected by the user. Each cluster contains news related under the following dimensions:

1) Spatial perspective: the news with the similar titles published in different newspapers;
2) Temporal perspective: the news with the similar titles published in different times.

[26] http:// www.Feedsifter.com
[27] http://www. FeedRinse.com

Then, the similarity between items (using only the title) is computed using Jaccard (BAEZA-YATES, R. and Ribeiro-Neto, B., 1999) similarity methods.

2.3.3 Data mashup

In web feed context, news exists in different formats and versions. In addition, some provides only summary, full news, with associate multimedia information (e.g. video clip, sound, etc) and integrating them is an issue that needs to be investigated.

Currently, the advent of Web 2.0 allows users to mashups data or services so as to create a service that serves a new purpose. Most of the mashup tools are used to remix news articles published by differnt providers (Yahoo-pipes[28], Damia(ALTINEL, M. et al., 2007), Mashmaker (ENNALS, R. J. and Garofalakis, M. N., 2007), Piggy Bank (HUYNH, D. et al., 2007), WebScripter (YAN, B. et al., 2003), Drapper (SHIR, E. and Aizen, J., 2005) and Potluck (HUYNH, D. F. et al., 2008)).

Damia, Yahoo! pipes and Mashmaker, use XML based data model as integration mechanism. Hence, schemas of the feeds are converted into the internal schema manually (case of damia, mashmaker) or using semi-automatic method (Yahoo! pipes and drapper).

Damia

IBM provides a mashup tools Damia (ALTINEL, M. et al., 2007) to assemble data feeds from the Web, enterprise data sources, and result of quering data stored in relational database such as Mirocsoft Access[29] and DB2[30]. Damia supports three types of operators: ingestion, augmentation and publication operators. The ingestion operators transform non XML data (Excel, CVS, HTML) into internal model using wrappers. The augmentation operators perform the data management operations using set of operators to: extract information from sequences (Extract), filter tuples (Filter), iterate over items in a

[28] http://pipes.yahoo.com/pipes/
[29] http://office.microsoft.com/access
[30] http://www.ibm.com/db2

sequence (Iterate), construct a new sequence from other sequences (Construct), join (Fuse), sort (Sort), aggregate (Group). The publication operator convert the result of the mashup into common output formats such as JSON, HTML, XML (e.g. RSS).

Yahoo!pipes

Yahoo!pipes provides a graphical user interface for building a new mashup that aggregate web feeds, web page, and other services, create a Web-application from other various sources and publish those applications. A pipe is composed of one or more modules; each module perfoms a task such as retrieving a feed from Web, filtering, and combining. The data manipulation operators are shown in Table 2.2. In addition, it allows users to pipe information from atmost 5 sources and setup rules on how content should be formulated using filter, union, extract, sort, unique, trunct and other operators. In general, the pipe allows aggregating web data using the RSS 2.0 as internal or gloabl schema.

Apatar [31]

Apatar is an open source *Extract-Transform-Load* and mashup data integration application. Datamap in Apatar allows a user to link data between the sources and the targets. It is composed of data sources, and operators that allow defining the flow of data from the source(s) into the target(s). Apatar allows connectivity to various data sources and uses object-based internal data model, and hence specific objects are created for each data source. In the process, users have to define the structure of the output document, specify the correspondence between the input and the output fields using transform operator. Table 2.2 shows the operators supported by Apatar.

MashMaker

MashMaker (ENNALS, R. J. and Garofalakis, M. N., 2007) is a web-based tool for editing, querying and manipulating web data. MashMaker is integrated as part of a web

[31] http://www.apatar.com/

page (explorer) and allows a user to create a mashup by browsing and combining different web pages. To build the mashup, set of web pages are combined into one. The combination is done using widget, a small application that can be added to a web page.

Dapper

Dapper (SHIR, E. and Aizen, J., 2005) is web-based service that enable users to create an interactive feed from websites. Here, users have to choose the data sources, and elements to be seen in the output. It allows only extract, copy and paste operators.

In general the data mashups detailed above and summarized in Table 2.2 (a detail comparison between mashups can be found in (DI LORENZO, G. et al., 2009)), support union, join, filter and sort operation and none of these applications provide an approach that consider semantic based matching.

Table 2.2: Data manipulation operators offered by mashup tools

Mashup tool	Internal model	Data manipulation	Description of the operation
Damian	- XML	- Merge	- combine source feeds based on expression that is applied to the feeds. The expression compares an item value from the first feed with an item value from the second feed. All items satisfying the expression are merged or joined in the resulting new feed
		- Union	- Combine two or more feeds into one feed. The entries from the first feed are added first then the entries from second feed.
		- Filter	- extract those feeds that satisfy a given condition
Yahoo! pipes	- XML	- Union	- combine a data from different sources
		- Sort	- sort on key
		- Filter	- used to extract specific items from a feed that meet the filter condition.
Apatar	- Object	- Aggregate	- combine two different data sources. The user must define the structure of the output and specify the correspondence between the input and the target in the aggregate operator
		- Filter	- used to extract the data that specify the condition
		- Join	- combine those data items that satisfy the join condition
MashMaker, Dapper	- XML	- Copy - Paste - Extract	- Elementary operators to extract and copy and put it another place

In the next sub-section, we review the main approaches in concept-based similarity measures.

2.4 Concepts similarity

In the fields of Natural Language Processing (NLP) and Information Retrieval (IR), semantic knowledge also called Knowledge Base (thesauri, taxonomies and/or ontologies) provides a framework for organizing entities such as words/expressions (SMEATON, R. and Richardson, A. F., 1995; LIN, D., 1998), generic concepts (RODRÍGUEZ, M. A. and Egenhofer, M. J., 2003; EHRIG, M. and Sure, Y., 2004), web pages (MAGUITMAN, A. G. et al., 2005) into a semantic space. Subsequently, the Knowledge Base is utilized to compare/match the entities with respect to their

corresponding similarity/relevance degrees with one another. In this section, we detail the notions related to semantic knowledge and concept similarity measures.

2.4.1 Semantic Knowledge

In the last two decades, semantic knowledge has been applied in the area of machine translation and learning (TOVE MANUAL, 1995), word sense disambiguation (DAHLGREN, K, 1995), query expansion and rewriting (HOEBER, O. et al., 2005), document classification (PENG, X. and Choi, B., 2005), document similarity (SONG, I. et al., 2007), design of question-answer system (GONZÁLEZ, J. L. V. and Rodríguez, A. F., 2000), etc. Semantic knowledge can be represented as frames (MINSKY, M., 1975), rules, semantic networks (NASROLAHI, S. et al., 2009) and KL-ONE (BRACHMAN, R. J. and Schmoke, J. G., 1985), and expressed using recent variants of description logics and RDF schema (RDFS) (MCBRIDE, B, 2004), and Web Ontology Language (OWL) (MCGUINNESS, D. L. and Harmelen, F., 2004).

A semantic knowledge generally comes down to a semantic network which is composed of a collection of nodes representing concepts and arc/edge representing a semantic relationship between the concepts.

A sample semantic knowledge extracted from WordNet[32] is shown in Figure 2.2.

[32] WordNet (WORDNET 2.1, 2005) is a domain independent lexical database for the English language provided by the University of Princeton. It groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets

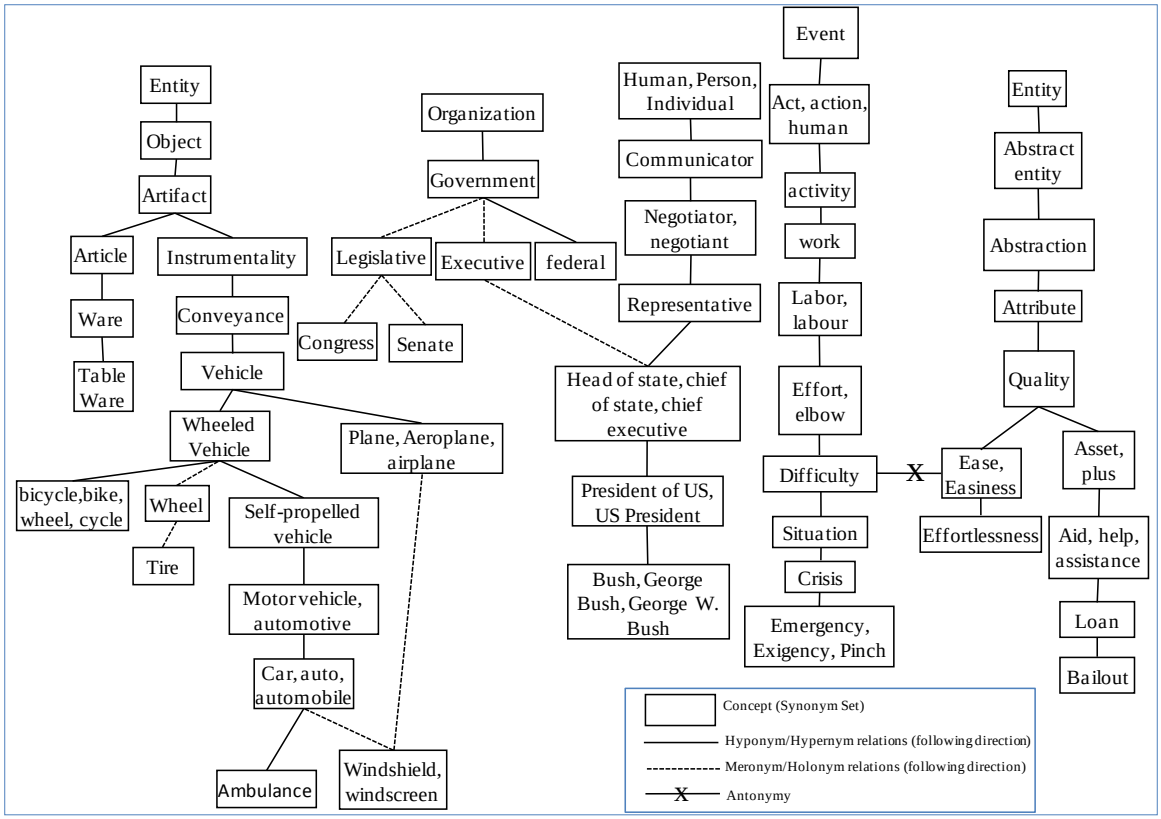


Figure 2.2: Fragment of WordNet taxonomy

2.4.2 Semantic Relations

Hereunder, we detail the most popular semantic relations employed in the literature, (WORDNET 2.1, 2005; LIN, D., 1998; MILLER, G. A. et al., 1990):

- *Synonym* (≡): Two words/expressions are synonymous if they are semantically identical, that is if the substitution of one for the other does not change the initial semantic meaning (e.g., Car ≡ Auto).
- *Hyponym* (≺): It can be identified as the *subordination* relation, and is generally known as the *Is Kind of* relation or simply *IsA* (e.g., Car ≺ automotive).

- *Hypernym* (≻): It can be identified as the *super-ordination* relation, and is generally known as the *Has Kind of* relation or simply *HasA* (e.g., Automotive ≻ Car).
- *Meronym* (≪): It can be identified as the *part-whole* relation, and is generally known as *PartOf* (also *MemberOf*, *SubstanceOf*, *ComponentOf*, etc.) (e.g., Windshield ≪ Car).
- *Holonym* (≫): It is basically the inverse of meronym, and is generally identified as *HasPart* (also *HasMember*, *HasSubstance*, *HasComponent*, etc.) (e.g., Car >> Windshield).

Table 2.3: Property of relations

Property / Relation	Reflexive	Symmetric	Transitive
Synonymy (≡)	✓	✓	✓
Hyponym (≺)	✓	×	✓
Hypernym (≻)	✓	×	✓
Meronym (≪)	✓	×	✓
Holonym (≫)	✓	×	✓

Other semantic relations such as Possession, RelatedTo, Cause/Effect (WORDNET 2.1, 2005) may exist between concepts. However, the Hyponym/Hypernym and Meronym/Holonym relations constitute the major part of the semantic knowledge.

Table 2.3 reviews the most frequently used semantic relations along with their properties (WORDNET 2.1, 2005; LIN, D., 1998; MILLER, G. A. et al., 1990). Note that, the transitivity property is not limited only to semantic relations of the same type and could also exist between different semantic relations as shown in Example 2.1.

Example 2.1: Referring to the knowledge base shown in Figure 2.2:

- 'US President' ≺ 'Head of state' and 'Head of state' ≪ 'Executive', transitively we infer that, 'US President' ≪ 'Executive'.
- 'Tire' ≪ 'Wheel' and 'Wheel' ≪ 'Car', transitively we infer that 'Tire' ≪ 'Car'.

Formally, given three concepts C_i, C_j and C_k related with semantic relation R_{ij} (between C_i and C_j) and R_{jk} (between C_j and C_k) in a given Knowledge Base, Table 2.4 details the transitivity relationship that might connect concept C_i and C_k using the semantic relations shown in Table 2.3. The relevance of identifying these relationships would be shown in Chapter 3 while identifying semantic neighborhood of a concept.

Table 2.4: Intra transitivity semantic relationships

R_{ij} \ R_{jk}	$\equiv$	$\prec$	$\succ$	$\ll$	$\ll$
$\equiv$	$\equiv$	$\prec$	$\succ$	$\ll$	$\gg$
$\prec$	$\prec$	$\prec$	$\emptyset$	$\ll$	$\emptyset$
$\succ$	$\succ$	$\emptyset$	$\succ$	$\emptyset$	$\gg$
$\ll$	$\ll$	$\ll$	$\emptyset$	$\ll$	$\emptyset$
$\gg$	$\gg$	$\emptyset$	$\gg$	$\emptyset$	$\gg$

Notice that, a value of $\emptyset$ in the table denotes the absence of relationship between the C_i and C_k.

In the next two sub-sections, we assess the concept similarity approaches that are categorized into two: distance-based and information content-based approaches.

2.4.3 Distance-based approaches

The distance-based approaches use the distance/path-length between concepts in semantic knowledge as basic parameter.

Simple edge counting/path length approach is the easiest method to measure the similarity between words/concepts. In this approach, the similarity is commonly computed as the minimum number of edges separating the two words/concepts (RADA, R. and Bicknell, E., 1989; RESNIK, P., 1995). Rada & Bicknell (RADA, R. and Bicknell, E., 1989) use the Medical Subject Heading Knowledge Base and count the

number of edges between terms in the MeSH hierarchy as a measure of the conceptual distance between terms. It is denoted as:

$$Sim_{\text{Rada \&}Bicknell}(C_1, C_2) = \text{len}(C_1, C_2) \quad (2.1)$$

Leacock & Chodorow (LEACOCK, C. and Chodorow, M., 1998) propose scaled concept-based measure by including the maximum depth of the semantic knowledge as a path length normalization factor. It is denoted as:

$$Sim_{\text{Leacock \& Chodorow}}(C_1, C_2) = \frac{-\log(\text{len}(C_1, C_2))}{2 \times D} \quad (2.2)$$

where:

- D is max depth of a concept in a semantic knowledge.
- $\text{len}(C_1, C_2)$ returns the path length/distance between C_1 and C_2.

Wu & Palmer (WU, Z. and Palmer, M. S., 1994) evaluate a conceptual similarity between pair of concepts in hierarchy-based Knowledge Base using their most common ancestor. The similarity measure takes into consideration the depth of the least common ancestor concept as well as the distance separating each concept from the least common ancestor. It is denoted as:

$$Sim_{\text{Wu \& }Palmer}(C_1, C_2) = \frac{2 \times \text{depth}(C)}{\text{len}(C_1, C) + \text{len}(C, C_2) + 2 \times \text{depth}(C)} \quad (2.3)$$

where:

- C is the least common ancestor that subsumes C_1 and C_2
- depth(C) is the depth of C (i.e. the distance separating C from the root of the semantic network)
- $\text{len}(C_1, C_2)$ returns the path length between C_1 and C_2

2.4.4 Information content-based approaches

The above distance-based measures assume that edges are uniform or have the same type and hence represent uniform distance. In real semantic network, the distance covered by a single link can vary with the network density, node depth, and information content of the corresponding nodes. One attempt towards this issue is the use of the maximum information the concepts share in common.

In hierarchical semantic network, the common information is identified as a function of the information in the least common ancestor that subsumes both concepts.

Definition 2.2 [Information Content]

In information theory, the information content (IC) of a concept C is computed as negative log likelihood using the probability theory. The probability of a concept C is computed as the aggregate frequency of all words/expressions subsumed by the concept C in a given corpus. It is denoted as:

$$\mathrm{IC}(\mathrm{C}) = -\log \mathrm{p}(\mathrm{C}) = -\log\left(\frac{\mathrm{Freq}(\mathrm{C})}{\mathrm{N}}\right) \tag{2.4}$$

where:

- $\mathrm{p}(\mathrm{C})$ is the probability of encountering an instance of *C*
- $\mathrm{Freq}(\mathrm{C}) = \sum_{\mathrm{w}\in\mathrm{C}} \mathrm{Count}(\mathrm{w})$: total number of occurrence of words subsumed by C, in the given corpus
- N: total number of words encountered in the corpus ■

According to Resnik (RESNIK, P., 1995), semantic similarity between two concepts C_1 and C_2 depends on a measure of the extent to which they share common information in ISA taxonomy. It is denoted as:

$$Sim_{\text{Resnik}}(\text{C}_1, \text{C}_2) = \text{IC}(\text{C}) \tag{2.5}$$

where, IC(C) is the information content of their least common subsume of C_1 and C_2.

Notice that, Resnik computes IC using the frequency of 1 million words in Brown Corpus of American English. According to Resnik formula (2.5), the similarity of two pair of concepts having the same least common ancestor is the same. The drawback of this approach is demonstrated in Example 2.2.

Example 2.2: Referring to the Knowledge Base shown in Figure 2.2. The similarity between *Car* and *Plane* is the same as the similarity between *Wheeled Vehicle* and *Plane* as the least common ancestor of each pair is vehicle i.e., Sim_{Resnik}*(Car, Plane)* = Sim_{Resnik}*(Wheeled vehicle, plane)*. However, in reality similarity between Wheeled Vehicle and Plane is more than the similarity between *Car* and *Plane.*

In an attempt to address this problem, Lin's universal similarity measure (LIN, D., 1998) defines the similarity between two concepts as a ratio of the amount of information needed to state their commonality and the information needed to fully state each of them. It is denoted as:

$$Sim_{Lin}(C_1, C_2) = \frac{2 \times IC(C)}{IC(C_1) + IC(C_2)} \tag{2.6}$$

In real semantic networks, the distance covered by a single link can with regard to the network density, node type and the information content of the corresponding nodes. Jiang and Conrath (JIANG, J. J. and Conrath, D. W., 1997) add that link distances could also depend vary according to link type and combine taxonomical distance (i.e., path length) with corpus statistical information (i.e., information content) to compute the semantic distance. Hence, the semantic distance between two concepts is qualified with the computational evidence derived from the distributional analysis of the corpus data. It is denoted as:

$$Dist_{Jiang\ \&\ Conrath}(C_1, C_2) = IC(C_1) + IC(C_2) - 2 \times IC(C) \quad (2.7)$$

Recently hybrid based approaches (QIN, P. et al., 2009; HONG-MINH, T. and Smith, D, 2008) that combine the edge counting and information content in WordNet taxonomy have been proposed. Zhou *et al*. (ZHOU, Z. et al., 2008) combine the path length and IC of each concept as a metric and the weight of each metric is adapted manually. Qin et al. (QIN, P. et al., 2009) combine the semantic distance approach of Jiang and Conrath (JIANG, J. J. and Conrath, D. W., 1997) with the Lin's universal similarity measure (LIN, D., 1998), whereas Hong-Minh & Smith (HONG-MINH, T. and Smith, D, 2008) combine the edge counting with the IC while taking into consideration the link strength and depth of the semantic knowledge.

The concept similarity measures presented in this section are capable to identify similarity taking into consideration mainly is kind of relation. However, none of these measures are capable to identify the relationship existing between concepts.

In the next section we present the review of semi-structured/XML document similarity approaches.

2.5 Semi-structured/XML documents comparison

In the literature, various semi-structured/XML similarity/comparison approaches are proposed. We categorize the proposals to three: structure-based, content-based and hybrid.

2.5.1 Structure-based similarity

The structural similarity is mainly computed using tree edit distance (BILLE, P., 2005). For instance, Chawathe (CHAWATHE, S. S., 1999), Nireman and Jagadish (NIERMAN, A. and Jagadish, H. V., 2002) and Tekli et al (TEKLI, J. et al., 2007)

consider the minimum number of edit operations, insert node (sub-tree), delete node (sub-tree), move node (sub-tree), and update node, to transform one XML tree into another. The work of Chawathe (CHAWATHE, S. S., 1999) has been considered as a base to a number of XML structured comparisons. Chawathe restricts insertion and deletion operations to leaf nodes and allows relabeling of nodes anywhere in the document while disregarding the move operation. In his paper, Chawathe uses tree edit comparison approach of Wagner-Fisher (WAGNER, R. and Fisher, M., 1974) in association with node tag and its depth pair (label, depth). He further extends the approach for external memory based similarity computation and identifies I/O, RAM and CPU costs. The overall complexity is quadratic and depends on the maximum number of nodes in the tree. Recently, Tekli et al. (TEKLI, J. et al., 2007) use semantic tag similarity together with the tree edit distance in computing the similarity between heterogeneous XML documents.

However, evaluating a tree edit distance is computationally expensive and does not easily scale up to large collections. As a result, other techniques that exploit the structural characteristic of XML documents have been proposed such as tag similarity (BUTTLER, D., 2004), edge similarity (LIAN, W. et al., 2004) and path set match (RAFIEI, D. et al., 2006).

Flesca et al. (FLESCA, S. et al., 2005) use Fast Fourier Transform to compute similarity between XML documents. They extract the sequence of start tags and end tags from the documents, and convert the tag sequence to a sequence of numbers to represent the structure of the documents. The number sequence is then viewed as a time series and the Fourier transform is applied to convert the data into a set of frequencies. The similarity between two documents is computed in the frequency domain by taking the difference in magnitudes of the two signals.

However, in feed context the structural similarity approaches alone is not enough, as in most cases news feeds of the same version and type are similar automatically.

2.5.2 Content-based similarity

In content-based similarity of XML documents, the similarity is computed using the contents/values of documents without assigning any special significance to the tags or the structural information. For example, Information Retrieval (IR) search engines typically ignore markup in HTML documents when matching phrases. The similarity can be done with/without considering semantics. In IR (MCGILL, M. J., 1983), the content of a document is commonly modeled with sets/bags of words where each word (and subsumed word(s)) is commonly given a weight computed with Term Frequency (TF), Document Frequency (DT), Inverse Document Frequency (IDF), and the combination TF-IDF(BAEZA-YATES, R. and Ribeiro-Neto, B., 1999).

The known approach to measure the similarity between two texts is aggregating the similarity of their corresponding lexical components, using vector space model (MCGILL, M. J., 1983) or fuzzy information retrieval (OGAWA, Y. et al., 1991). This has been improved by considering stemming, stop-word removal, part-of-speech tagging etc. However, lexical-based text similarity wouldn't identify semantic similarity. For example "*A cemetery is a place where dead people's bodies are buried*", and "*A graveyard is an area of land, sometimes near a church, where dead people are buried*" are similar but the similarity is dependent on the semantic similarity existing between *cemetery* and *graveyard*, *place* and *land*, in addition to the commonality of the texts.

The semantic similarity between two texts has been measured using different techniques. Mihalcea et al (MIHALCEA, R. et al., 2006) extend the lexical texts similarity approach by aggregating the maximum similarity between the corresponding words of the two texts combined with word specificity. It is denoted as:

$$Sim(T_1, T_2) = \frac{1}{2}\left(\frac{\sum_{w\in\{T_1\}} maxSim(w, T_2)\times idf(w)}{\sum_{w\in\{T_1\}} idf(w)} + \frac{\sum_{w\in\{T_2\}} maxSim(w, T_1)\times idf(w)}{\sum_{w\in\{T_2\}} idf(w)}\right) \quad (2.8)$$

The maximum similarity is computed using two corpus based similarity metrics PMI-IR (Pointwise Mutual Information and Information Retrieval) (TURNEY, P. D., 2001) and LSA (Latent Semantic Analysis) (LANDAUER, T. K. and Dumais, S. T., 1997)) and six Knowledge Base metrics: Jiang and Conrath (JIANG, J. J. and Conrath, D. W., 1997), Leacock and Chodorow (LEACOCK, C. and Chodorow, M., 1998), Lesk (LESK, M. E., 1986), Lin (LIN, D., 1998), Resnik (RESNIK, P., 1995), Wu and Palmer (WU, Z. and Palmer, M. S., 1994). PMI-IR measures the extent to which two words coexist together in very large corpus such as the Web. LSA represents the term co-occurrence in the corpus using a dimension reduction technique operated by a Singular Value Decomposition (SVD) and the similarity is computed using vector-based similarity method (e.g., cosine, dot product). However, these measures are not capable to identify the relationship that exists between two texts.

In fuzzy information retrieval (OGAWA, Y. et al., 1991), the similarity between two texts is computed by aggregating texts fuzzy association which depends on correlation between keywords. The keywords correlation factor that measures the similarity between two words is computed with the frequency of keywords, co-occurrences and relative distance in very large corpus such as Wikipedia[33]. The normalized correlation coefficient nC_{ij} between two words w_i and w_j in a given corpus is computed as:

$$nC_{ij} = \frac{\sum_{w_{i \in V(w_i)}} \sum_{w_j \in V(w_j)} \frac{1}{d(w_i, w_i)}}{|V(w_i)| \times |V(w_i)|} \tag{2.9}$$

where:

- $d(w_i, w_i) = \left|Position(w_i) - Position\left(w_j\right)\right|$ is the distance between the words
- $V(w_i)$ and $V\left(w_j\right)$ represents the list of keywords in a Wikipedia document
- $|V(w_i)|$ represents the number of words in the document

[33] http://www.wikipedia.org/

Given k different Wikipedia documents containing both keywords w_i and w_j, the unigram correlation factor cf_{ij} is computed as the average of the normalized correlation coefficients of the keywords in each document.

$$Cf_{i,j} = \frac{\sum_{m=1}^{k} nc_{i,j}^{m}}{k} \tag{2.10}$$

where, $nc_{i,j}^{m}$ is the normalized correlation coefficients of w_i and w_j computed on the m^{th} document.

Then, a phrase correlation factor is defined using the n-gram correlation factors. A fuzzy association (OGAWA, Y. et al., 1991) between a phrase p in the first text and all the phrases in the second text is computed as the complement of a negative algebraic product of all correlations of p and each distinct phrase p_k in the other text. It is denoted as:

$$\mu_{pi,2} = 1 - \prod_{p_k \in T_2} (1 - nCf_{i,k}) \tag{2.11}$$

The degree of similarity between two texts is computed as the average of the fuzzy association for each phrase p_i in the first text and phrases in the second text. However, computing the correlation coefficient is both time and space consuming.

Recently, (GUSTAFSON, N. and Pera, M. S., Ng, Y., 2008; PERA, M. S. and Ng, Y., 2008; PERA, M. S. and Ng, Y, 2007) used the fuzzy model approach to measure the similarity between two RSS news articles using the text content extracted from title and description elements. The authors used pre-computed keyword correlation factors between pair of keywords and define fuzzy association in order to get asymmetric similarity value. In (PERA, M. S. and Ng, Y., 2008), the authors use phrase matching approach (such as n-gram) in finding similar RSS articles collected from the same or different sources. However, the approach disregards structural heterogeneity (caused by differences in versions and formats associated to tag names) and the similarity approach

is restricted to an RSS content descriptor (composed of the content of title and description elements).

In the next sub-section, we provide a detail review on the hybrid similarity measures which are related to our study.

2.5.3 Hybrid similarity

Recently, the combination of structure and content based similarity values has been proposed in detecting document similarity (VIYANON, W. and Madria, S. K., 2009), document clustering (TRAN, Tien et al., 2008), data integration (VIYANON, W. et al., 2008), etc. The structural similarity value is computed for instance with Path Similarity (RAFIEI, D. et al., 2006), Edge similarity (LIAN, W. et al., 2004), or Tag similarity (BUTTLER, D., 2004); and the content similarity value is computed with classical Vector Space Model (SALTON, G. et al., 1975) or extended Vector Space Model (FOX, E. A., 1983), fuzzy logic, etc. These two similarity values are combined using entropy, weighted sum, or average methods.

Ma & Chbeir (MA, Y. and Chbeir, R., 2005) proposed a bottom-up approach to combine the instance similarity values to get corresponding simple elements similarity and aggregate simple elements similarity value to get document or complex elements similarity value. In computing text similarity (instance of type text), a semantic similarity restricted to atomic values, with the help of dedicated semantic knowledge (a weighted edge tree), is demonstrated. The weight of an edge represents the asymmetric similarity between the two concepts. The semantic similarity between two concepts is computed as the product of the weight associated to the edge connecting concepts in the semantic knowledge. A parent node in the semantic knowledge is semantically identical to all its descendents and similarity between a child and its parent is equal to 1/ n (n is number of children of the parent). In addition, the approach used to compute the structural similarity

value and the method used to combine the structural and content similarity values is not detailed.

In (KIM, T. et al., 2007; GHOSH, S. and Mitra, P., 2008), a combination of path similarity and content similarity computed using cosine similarity is proposed. In (KIM, T. et al., 2007) the authors argue that the weight of a content term should reflect its frequency, the importance associated to the tag and inverse document frequency. However, the authors didn't state the approach used to combine the two similarity values. Ghosh & Mitra in (GHOSH, S. and Mitra, P., 2008) use the weighted sum of the two similarity values to get the final similarity values. The weight is computed automatically using an entropy approach.

In (KIM, W., 2008), Kim proposed an approach that combines string-based structured similarity value with weight-based content similarity value. In weight-based content similarity approach, the root node has a weight of 1, and the weight of the parent node is equally shared among children nodes. However, only child has half of the weight of the parent. The author assumes that if corresponding nodes don't have identical weight then the documents are different. This work lacks clarity in each of the following points: (1) structural similarity is restricted to lexical units and not semantic-aware (for instance *star* and *actor* are not identical but are related), (2) the approach used to compute the similarity between the content of two simple elements or leaf nodes is not clearly stated, and (3) the definition of content similarity is not clear as an element could be complex and its content is dependent on all the sub-elements. Based on this approach, any two RSS news items are identical.

Recently, in (XIA, X. et al., 2009), Xia et al. propose an Extended Vector Space Model (FOX, E. A., 1983) having three sub-vectors to measure the similarity between two documents. In this approach, any XML document is partitioned into three independent parts: metadata, body and link, taking into consideration the level of the element and the number of keywords/terms in the text node. The similarity between metadata sub-trees is

computed using classical vector space with a weight reflecting the existence of structural term (i.e., path from the root to key term) and the similarity is computed as dot product of the vectors. The similarity between body sub-trees involves two vectors containing path and content terms and the weighted sum is used to combine the two similarity values. The similarity between links is computed using dice similarity[34] method. Finally, the three similarity values are combined using a weighted sum. However, neither the structural nor the content similarity is semantic-aware and in news feed context this approach comes down to the use of classical vector space as feeds are not deep nested XML documents.

In (KADE, A. M. and Heuser, C. A., 2008), *XSim*, a structural and content aware XML comparison framework is presented. Here, the similarity between the elements of two XML trees is assessed in two steps. In the first step, every XML tree is decomposed into sub-trees in the top-down manner. For each sub-tree, path-content pair is identified. The content of a node is the concatenation of the content of its leaf sub-nodes. The sub-trees contents of two XML trees are then compared against each other using a string similarity function. *XSim* computes the matching between XML documents as an average of matched list similarity values. The similarity value is computed as an average of content, tag name and path similarity values without considering semantics. This approach suffers of two problems: 1) the authors didn't specify how the correspondence between sub-trees is identified, and 2) the approach is very much similar to the content-based approach that ignores the structure of the tag as content similarity between the root nodes determines the similarity between the documents.

Relational SQL-based approach in XML document similarity is detailed in XDoI (VIYANON, W. et al., 2008), XML-SIM (VIYANON, W. and Madria, S. K., 2009) and XDI-CSSK (VIYANON, W. and Sanjay, M., 2009). The authors underline the need to fragment XML documents in a data centric manner into sub-trees representing

[34] Dice similarity or Dice coefficient is related to the Jaccard similarity index. The similarity between objects is twice the number of commonality over the total number of in both objects.

independent objects. The process starts by mapping XML documents to relational database using XREL (YOSHIKAWA, M. et al., 2001). The database stores the documents, attributes, sub-trees, paths, and an XML key is associated to each sub-tree. The similarity between two sub-trees is determined in two steps (VIYANON, W. and Sanjay, M., 2009):

1) matching sub-trees with key values (key matching reduces unnecessary matching)
2) matching sub-trees using similarity measures based on XML content and structure.

The similarity between two documents (base t_i and target t_j) is computed using Sub-tree Similarity Degree on the base document (*SSD1*), Sub-tree Similarity Degree based on both documents (*SSD2*) and Path Sub-tree Similarity Degree (*PSSD*). *SDD1* is related to the percentage of the number of leaf nodes *n* having the same textual values out of the total number of leaf nodes in base documents. *SSD2* is the ratio of common matched leaf-node values between the base and target sub-trees. SDD1 and SDD2 are denoted as follows:

$$SSD1(t_i, t_j) = \frac{n}{|t_i|} \times 100\%$$
$$SSD2(t_i, t_j) = \frac{2n}{|t_i| + |t_j|} \times 100\% \quad (2.12)$$

where:

- t_i and t_j are the sub-trees in the target and destination documents
- *n* is the number of leaf nodes having the same textual values
- $|t_i|$ and $|t_j|$ are the numbers of leaf nodes in the base and target documents respectively

In computing the structural or Path Similarity Degree (PSD), two complementary approaches are documented.

1) In (VIYANON, W. and Sanjay, M., 2009), the PSD is computed in two steps. Firstly, relabeling elements tag with the least common ancestor of their corresponding tag name's using the Wu & Palmer (WU, Z. and Palmer, M. S., 1994) similarity metric on the WordNet taxonomy. Secondly, PSD is computed as a ratio of the number of common labels on the paths from the base and target sub-trees having the same textual value to the number of path elements in the base sub-tree.
2) In (VIYANON, W. and Madria, S. K., 2009), the PSD is computed as the average of aggregated similarity between tag names (using Rensnik's (RESNIK, P., 1995)).

Finally, the similarity between the sub-trees is computed as the product of the average of PSD and SSD. Two sub-trees are similar if their similarity value is greater than a given threshold.

This approach is not usable in identifying the similarity between news feeds as the content is text rich and author dependent and defining unique key to RSS is close to impossible. In Table 2.5, we summarize the hybrid XML similarity approaches.

Table 2.5: Summary of combined XML document similarity approaches

	Structural similarity	Content similarity	Combining method
(MA, Y. and Chbeir, R., 2005)		Knowledge based	Weighted sum
(KIM, T. et al., 2007)	Path similarity	Vector space	
(KIM, W., 2008)	Extended depth first search string similarity	Normalized weight of node	
(GHOSH, S. and Mitra, P., 2008)	Path similarity	Vector space	Entropy based weight sum
XSIM (KADE, A. M. and Heuser, C. A., 2008)	Tag name, path similarity	String similarity	average
XDOI(VIYANON, W. et al., 2008)	Path similarity degree	Content similarity degree	Average
XML-SIM (VIYANON, W. and Madria, S. K., 2009)	Tag similarity		
XDI-CSSK (VIYANON, W. and Madria, S. K., 2009)	Path similarity degree		
(XIA, X. et al., 2009)	Path similarity	Extended Vector space	Weighted sum

In the next section, we present the review of the three approach used to merge semi-structured and XML documents.

2.6 Merging

Merging refers to combining inputs together in order to get a unified output. In the literature, merging has been studied extensively in different application domains such as distributed database design (KROGSTIE, J. et al., 2007; POULOVASSILIS, A. and McBrien, P., 1998; BERGAMASCHI, S. et al., 2001; HAMMER, J. et al., 1997; COHEN, W., 1998; LENZERINI, M., 2002), belief management (KONIECZNY, S. et al., 2004), version and revision control (BERLINER, B., 1990; TICHY, W., 1985; COLLINS-SUSSMAN, B. et al., 2004), information systems (BERNSTEIN, P. A. and Haas, L. M., 2008), and model management (BRUNET, G. et al., 2006; NEJATI, S. et al., 2007; POTTINGER, R. A. and Bernstein, P. A, 2003).

In reality, there are two main factors that make the merging process complicated:

1) objects may overlap, in that they share some concepts but the overlapped concepts might be presented differently in each object
2) an object may evolve through a number of different versions and the merge should be recomputed if the original object is updated

The first factor is related to the need of identifying a relationship that may exist between the objects to be merged; and this necessitates a semantic based approach that identifies the degree of overlap. The second factor is related mostly to version and revision system; and it is also an issue in news evolution management as a news item could evolve over time as new developments might be added to already published news.

Independent of the application domains, a merger provides a way to combine objects (i.e. schemas, models, documents, etc) and provides unified view so as to perform various type of analysis.

Herewith, we present the review of literature focusing on merging in distributed database and semi-structured/XML data.

2.6.1 Distributed database

Merging of information/data is one of the key issues in the design of federated, heterogeneous and distributed databases. A number of studies have been made with approaches based on schema integration/merging (e.g., (KROGSTIE, J. et al., 2007)), particularly the use of a global conceptual schema (e.g., (POULOVASSILIS, A. and McBrien, P., 1998; BERGAMASCHI, S. et al., 2001)). In federated and heterogeneous database integration (HAMMER, J. et al., 1997; COHEN, W., 1998), transparency and merging is achieved with the use of wrappers, mediators and views (Local-as-view (ULLMAN, J. D., 1997) or global-as-view (HALEVY, A. Y., 2001)) that convert the user's query to be processed against the native database schema.

In the web based heterogeneous and distributed database integration, XML-based common data model such as XML DTD, MIX (LUDÄSCHER, B. et al., 1999) or XML Schema (LEE, K. et al., 2002) is used. XML schema is generic and supports both built in, user defined and inheritance types. Hence, it is complete for a data model in the integration process. However, the use of XML schema causes both structural and semantic heterogeneity problem. The classification based conflict identification method of Lee *et al*. (LEE, K. et al., 2002) later adopted by Tseng (TSENG, F. S.C., 2005) categorizes conflicts into two: *Conflicts of similar schema structures* and *Conflicts of different schema structures*. In these systems (TSENG, F. S.C., 2005; RAJESWARI, V. and Varughese, K. Dharmishtan K., 2009), a user issues a global query and the global site decomposes the query and sends the sub-queries to each of the relevant sites. Each local site executes the query and responds the result in XML format. The DBA of each site prepares XSLT that transforms a local data into global conceptual schema. However, merging in database design focus only on integrating the structurally different database without considering their content which is not enough in web-feed context.

2.6.2 Semi-structured/XML documents

Merging hierarchically semi-structured data-centric files (e.g., drawings, structured texts, XML documents, web-pages) has been studied by different researchers: Fontaine (FONTAINE, R.L., 2002), Lindholm (LINDHOLM, T., 2003; LINDHOLM, T., 2004) and Hunter & Liu. (HUNTER, A. and Liu, W., 2006; HUNTER, A. and Liu, W., 2006). Given two semi-structured/XML documents (T_1 and T_2), merger provides a new document as a result. We categorize the approaches into four: template-based (TUFTE, K. and Maier, D., 2001; TUFTE, K. and Maier, D., 2002; WEI, W. et al., 2004; LAU, H. and Ng, W., 2005), 2-ways (FONTAINE, R.L., 2002), 3-ways merging (LINDHOLM, T., 2003; LINDHOLM, T., 2004) and propositional fusion rules (HUNTER, A. and Liu, W., 2006; HUNTER, A. and Liu, W., 2006).

The 2-ways, 3-ways and template-based approaches promote the use of hard-coded merging rules. The merging rules decide on what to do when a particular condition is satisfied (such as a node is inserted, deleted, moved, or updated).

In both 2-way (FONTAINE, R.L., 2002; CURBERA, F., 1998; RAJPAL, N., 2002) and 3-ways (LINDHOLM, T., 2003; LINDHOLM, T., 2004) merging – a delta file containing the corresponding nodes of T_1 and T_2 (identified using tree edit (LINDHOLM, T., 2003; LINDHOLM, T., 2004) or Wu et al. (WU, S. et al., 1990) Longest Common Subsequence (LCS) string algorithm (FONTAINE, R.L., 2002)), the perceived operation and conflicts is generated. Here, the hardcoded merging rules make sure that operations made in T_2 are reflected in the merged document (i.e., insert, delete, update, and moved nodes in T_2) and hence the result is similar to the right-outer join operation using label equality as join condition.

The template based approach defines a merge template as a rule. Merge template is an expression/predicate that specifies the structure of the merged result. In (TUFTE, K. and Maier, D., 2002), it specifies what action should be triggered when the values of two structurally identical sub-documents are identified. Two elements match if their corresponding values referenced by paths are equal and the merger join them using inner, or outer join types. In (TUFTE, K. and Maier, D., 2002), the authors showed that the merge operation is logically performing a lattice-join of two XML documents in a subsumption lattice.

In (WEI, W. et al., 2004), the authors extend the merge template with two Boolean path expression match templates provided by the user so as to merge heterogeneous XML documents with their associated DTDs. The merging operator unionizes all matching elements of both documents if either the first template match expression (which act as default joining condition) or the later alternate template match expression (defined as second Boolean expression) is True. In (BUNEMAN, P. et al., 1999), the authors

proposed Deep Union operator, which is similar in nature to template merge operator, to combine edge-labeled trees having identical key values.

Hunter et al. have published several papers (HUNTER, A. and Summerton, R., 2006; HUNTER, A. and Summerton, R., 2004; HUNTER, A. and Summerton, R., 2003; HUNTER, A. and Liu, W., 2006; HUNTER, A. and Liu, W., 2006) concerning the use of Knowledge Bases and fusion rules in merging information. The authors are particularly interested in merging semi-structured information such as structured reports: XML documents having the same structure and the text entries are restricted to individual words or simple phrases, dates, numbers and units. Here, the tags represent semantic information and are associated to predefined functions. The merging process is controlled by propositional fusion rules (HUNTER, A. and Liu, W., 2006; HUNTER, A. and Liu, W., 2006) (kind of scripting language) applied to tags having the same name. The antecedent of the fusion rule is a call to investigate the information in the structured news reports and the background knowledge. The consequence of the fusion rule is a formula specifying actions to be taken to form the merged report.

The merging approaches detailed in this section are not applicable to text-rich and structurally different XML documents such as RSS due to two reasons: (1) the rules are not flexible as the merging rules are hardcoded; (2) the approaches are restricted to structurally identical XML document and text entry restricted to words and small phrases without natural language processing. Hence, Hunter’s fusion rule couldn’t be applied to text rich and author dependent XML document.

In the next section, we provide the state of art in querying XML documents using the known both traditional database query algebra and native XML algebras.

2.7 XML algebra

In the database community, it is common to translate query language into algebraic expression mainly for two reasons: 1) to validate the correctness of the query, and 2) to optimize query expression using query rewriting and query optimization options.

Algebra serves as intermediate representation of user query and it must be powerful enough to express all possible queries in certain query language. The 1970s Codd (CODD, E. F., 1970) relational model is the most popular and complete to manipulate alpha-numeric data. In this model, a data is represented as set of *n-ary* relations; each relation has an unordered set of tuples (rows) and attributes (that takes value from the corresponding domain). Codd defined six basic operators: selection, projection, cross product and union, difference, and rename as first class citizens in managing alpha numeric data.

Querying XML database has been done using the extension of relational approach (SCHMIDT, A. et al., 2000; KAPPEL, G. et al., 2000; MANOLESCU, I. et al., 2000; SHANMUGASUNDARAM, J. et al., 1999; ZHANG, X. et al., 2001), Object Oriented approach (CATANIA, B. et al., 2000), Object Relational approach (SHIMURA, T. et al., 1999) and native XML approach (NAUGHTON, J. F. et al., 2001; KANNE, C. and Moerkotte, G., 2000). In the following sub-sections, we present algebra related to XML and XML stream.

2.7.1 Database oriented algebra

Several extensions of Relational or Object-Oriented database management systems (DBMSs) have been provided to represent XML documents as a collection of relations or objects respectively. User queries are represented in the extended form of SQL or Object Query Language, executed in the database and finally the result of the query is reconstructed as XML document using a set of XML construction operators. For instance, relation like data model has been used in semi-structured and XML retrieval such as YAL

(SARTIANI, C. and Albano, A., 2002), SAL (CATANIA, B. et al., 2000) and XAT (RUNDENSTEINER, X. and Zhang, E., 2002).

YAL (SARTIANI, C. and Albano, A., 2002) uses *Env* relation like hierarchical data structure as data model and supports operation existing in both relational and object-oriented DBMSs. *Env* is an unordered collection of tuples, in which each tuple describes a set of variable bindings. *Env* is very much similar to YAT tab structures (CLUET, S. et al., 1998). It allows manipulating a set of tuples rather than trees and hence optimization and execution techniques are based on tuples. It provides two *boarder* operations: *path* (extracts information from persistence root that satisfies the filter condition and to build variable binding) and *return* (uses the variable binding and the output filter to produce new XML documents). The YAL algebra supports both set and list based operators such as selection, projection, TupJoin, Join, DJoin, Map, Sort, TupSort, and GroupBy. The predicate language in YAT is rich and supports universal and existential constraints in addition to comparison on simple values. The tuple oriented operators such as TupJoin accept two *Env*, a predicate and returns concatenation of tuples of *Env* satisfying the predicate. The join version accepts a combining function *f* that combines the tuples that satisfy the predicate. The DJoin, dependency join, joins two *Env* e_1 and e_2, where the evaluation of e_2 depends on e_1.

In XAT (RUNDENSTEINER, X. and Zhang, E., 2002), the rainbow system uses *XAT Table*, which is similar to *Env* of YAL, and supports XQuery. XAT implements three groups of operators (shown in Table 2.6) to handle both relational and XML sources: (1) *XML operators* to represent and retrieve XML documents, (2) *SQL operators* to formulate relation-like query and construct XAT table as output of the query and (3) *special operators* to assist query.

However, the SQL extensions are not suitable to XML streams (BABCOCK, B. et al., 2002) in general and news feed in particular as SQL can neither read XML data as it is, nor can generate XML document directly as output.

2.7.2 XML Native algebra

The native XML DBMSs use set of languages (such as: Quilt (CHAMBERLIN, D. et al., 2000), XQuery (ROBIE, J. et al., 2009), XPath (CLARK, J. and DeRose, S., 1999), YaTL (SARTIANI, C. and Albano, A., 2002)) to formulate a query.

The XML algebra of Fernández et al. (FERNÁNDEZ, M. F. et al., 2000) is probably the first that uses regular-expression types similar to DTDs or XML schemas. It is documented that its revised version has been submitted as a working draft of W3C XML Query Working Group. The authors proposed projection (similar to path navigation in XPath), iteration (similar to FOR statement in XQuery) and order dependent join operators. We believe that this algebra is very much similar to XML query language and its impact is clearly shown in the design of Quilt (CHAMBERLIN, D. et al., 2000) and XQuery (ROBIE, J. et al., 2009).

The XML algebras can be classified into two groups: tree-based and node-based. The tree-based algebras (e.g., (JAGADISH, H. V. et al., 2001; SARTIANI, C. and Albano, A., 2002; NOVAK, L. and Zamulin, A. V., 2006)) represent XML documents as rooted labeled tree, whereas the node based algebra (e.g., (FRASINCAR, F. et al., 2002; BEECH, D. et al., 1999; CATANIA, B. et al., 2000)) represent the inputs as a collection of vertices/nodes or graph.

Tree Algebra for XML (TAX) (JAGADISH, H. V. et al., 2001) manipulates XML data modeled as forests for labeled, ordered, rooted trees. Each node of the trees has a virtual attribute called pedigree which carries the history of “where it came from” i.e., document-id + offset-in-document and it acts as a unique value. TAX allows selection, projection, cartesian product, group by, set membership (union, intersection, and difference) operators. These operators accept pattern tree (i.e., a collection of numbered nodes related with parent-child (pc) or ancestor-descendent (ad) relations and formula/s presenting node names and predicates) and collection of nodes as input and return a set of

witness trees as output. Later, the pattern tree has been extended with generalized pattern tree (CHEN, Z. et al., 2003) and tree logical class (PAPARIZOS, S. et al., 2004).

TOSS (HUNG, E., et al., 2004) is an ontology-based semantic query extension of TAX. It is build on top of Xindice database system and consists of three components: Ontology Maker, Similarity Enhancer and Query Executor. The objective of TOSS is to integrate, and handle structural and schema conflicts existing in the XML data sources. In TOSS, for each XML file (source) an ontology describing tag names and corresponding relationship is generated automatically. Then, the generated ontologies are manually aligned and semantically enhanced (with the semantic enhancer component) by regrouping similar concepts. The user query is transformed into a query that uses the enhanced ontology. However, the semantic similarity is restricted to tag name and proper nouns or short textual values.

In XAL (FRASINCAR, F. et al., 2002), an XML document is regarded as rooted and directed graph. The algebra accepts a set of nodes as input and returns a set of nodes as output. The authors classified the operators into three:

a. extraction operators that retrieve information from XML document and returns collection of vertices from the original XML graphs: *projection*, *select*, *sort*, *distinct*, *union*, *unorder*, *join*, *union*, *intersection*, *difference*. Two vertices are equal if they have the same value independent of the tag name difference
b. meta-operators that control the evaluation of expression, and represent repetitions either at the input or operator level using *MAP* and *Kleene Star*
c. construction operators that build new XML documents from the extracted data using *create vertex*, *create edge*, and *copy operators*

In attempt to return set of relevant results to a given semi-structured query, researchers have proposed threshold-based (COHEN, S. et al., 2003; THEOBALD, M. et al., 2005) TopK operators. The threshold base TopK algorithm returns the top *k* data that have similarity value greater than the threshold value provided by the user or automatically

approximated. Some researchers (COHEN, S. et al., 2003; GUO, L. et al., 2003) adapted the traditional keyword-based searching approach to XML data. XSEarch (COHEN, S. et al., 2003), TopX (THEOBALD, M. et al., 2005) and XRank (GUO, L. et al., 2003) allow users to search for a set of XML fragments using keywords and the result is ranked on a score value that reflects keyword frequency and specificity, and proximity to the query. In Table 2.6, we summarize the XML algebras.

Table 2.6: Summary of XML Algebra

Algebra	Project	Data model	Supported operator	Note
(BEECH, D. et al., 1999)	- note to W3C	- directed graph	- navigation : follow - selection, join - construction: create vertex, edge - sort, map, unorder, distinct	- Ordered algebraic operators JOIN
SAL (BEERI, C. and Tzaban, Y., 1999)		- Ordered collection of Edge-labeled directed graph (OEM)	- selection, join, mapping - extended or list mapping –variable binding - group by - regular expression matching	
XML-QL (FERNÁNDEZ, M. F. et al., 2000)			- Projection, Iteration, Selection, Join	- Regular expressions base
TAX (JAGADISH, H. V. et al., 2001)	- TIMBER XML Database system	- ordered labeled rooted tree	- Projection, Selection, Cartesian product, Grouping	- Use pattern tree similar to Xtasy input filter operator
YAL (SARTIANI, C. and Albano, A., 2002)	- Xtasy DBMS (COLAZZO, D. et al., 2001)	- Unordered forest of labeled tree stored in OR database in *Env* model	- Border: path, return - Selection, filter, projection, TupJoin, Join, DJoin, MAP, Sort, TupSort, GroupBy	- preserve order using the TupSort operator
XAL (FRASINCAR, F. et al., 2002)		- Collection of ordered vertices - rooted connected graph	- Extraction: projection, selection, unordered, distinct, sort, join, product, union, intersection, difference - Meta- Map, Kleene star - Construction – create vertex, edge, copy	- heuristic based query optimization
XAT (RUNDENSTEINER, X. and Zhang, E., 2002)	- Rainbow system	- order based XAT table in OR format	- XML operator: Expose, tagger, Navigate, set operators, compose - SQL: project, selection, join, theta join, set operator, distinct, group by, order by - Special operators: source, SQLStat, For, If, Merge, Name	
(PAPAKONSTANTINOU, Y. et al., March 2003)	- Enosys XML Integration Platform	- Relational table Based on XML-QL	- union (without duplicate elimination) - projection - select - join - navigation –getD - Source - groupby - construction: crElt,cat, crList	- Data integration Query expressed in XCQL
TOSS(HUNG, E., et al., 2004)	- Xindice system	- Order directed Tree model	- ontology based extension of TAX (JAGADISH, H. V. et al., 2001) - Projection, Selection, Cartesian product, Grouping	- supports similarity operator on simple data; terms

2.7.3 Stream oriented algebra

Niagara system (NAUGHTON, J. F. et al., 2001) allows a user to query the Internet without specifying the XML sources while considering only the context (a context is similar to path expression – set of tag names related with containment relationship) in which the text exists. The authors show the streaming nature of the Internet and underlined the need to transform users' query (formulated with the help of their graphical interface) into XML-QL. The XML-QL references the set of candidate XML files generated with structure-aware Search Engine. However, the information provided on the Internet is very vast and the existence of syntactically different yet semantically related and identical XML data are unquestionable.

In addition, in (KOSTAS P., Timos K. S., 2006), Kostas P defines an important step towards stream algebra and presented some window-based operators such as selection, join, union, and aggregation with a predicate restricted to exact equality. The standard query language XQuery 1.1 provides the option to generate windows using the window clause that accept two boundary conditions, however to the best of our knowledge there doesn't exist an operator that uses this windows.

2.8 Summary

In this chapter, we have presented the most important and relevant works to the subject of this thesis. The related works are grouped into four. Firstly, we discussed and categorized concept similarity measures and pin-point their drawbacks. Most of the approaches are restricted mainly to the hierarchal *ISA* semantic relation and hence concepts related with other relation such as *PartOf* are considered unrelated. For instance *sim*(*Windshield*, *plane*) is zero. In addition, the concept similarity measures discussed are not capable of identifying the relationship that exists between concepts: two concepts could be synonym

(identical), one concept includes the other concept, both shares some information or they are disjoint.

Secondly, we have also discussed the different XML document similarity approaches that has been proposed in the literature and presented why it is not applicable to the news feed context. The similarity between two XML documents is measured by combining their structural and/or content similarity values. The structural similarity is computed with edit distance, tag similarity, path similarity, and edge similarity methods. The content similarity is computed using vector space, extended vector space, n-gram, etc considering semantic information or not. The existing XML document similarity approaches are capable to measure the extent to which the documents share the same information. However, these approaches ignore the importance of identifying the relationship (equal, include, overlap, or disjoint) between two XML objects at different level of granularity (text, simple elements or complex elements) which is a requirement in the design of different applications such as XML merger, access control and security.

Thirdly, we have assessed merging data in distributed database and semi-structured/XML documents management. Even though there are number of research works that address the issue of integrating data/information from different source, none of the existing work addresses the issue of providing a merging framework that fits to text rich, dynamic and writer dependent data using flexible and user provided merging rules. Even if the approach in (KROGSTIE, J. et al., 2007) considers the topological relations (equality, inclusion and disjointness), it does not consider the domain knowledge information in handling semantic conflicts or relationships between entities and its applicability is restricted to model merging.

Finally, we discussed the known algebraic approaches to query XML documents and identified the drawbacks in handling news feeds query. Most of the algebras assume the existence of unique document/node id or key value. In feed context defining such key value is almost impossible as its content is dynamic and highly dependent on authors'

verbification and style of writing. Besides, none of the existing XML algebras provides operators that take into consideration similarity and relationship existing between the contents of feed document.

CHAPTER 3
SEMANTIC-AWARE NEWS FEED RELATEDNESS

Abstract

One of the aims of our research was to measure the extent to which two news items are similar/related while considering the heterogeneity problem caused due to the various versions and formats of a feed, and the style and verbification of the authors. To achieve this, we choose a Knowledge Base approach that contains the set of related textual values and element labels stored in semantic network. The purpose of this chapter is to present a generic, easily customizable and extensible concept-based similarity measure that uses the set of concepts related with various semantic relations. Our concept similarity measure is based on the function of the number of shared and different concepts considering their global semantic neighborhoods. This similarity measure correlates more to the human concept rating and is capable to identify the similarity and relationship between concepts. To identify the relatedness between news feeds, we apply a bottom-up and incremental approach. Here, we use the concepts similarity values and relationship as a building block for texts, simple elements and items relatedness algorithms. In addition, these three algorithms identify relatedness (having similarity and relationship value) and runs in a polynomial timing depending on both semantic and syntactic information.

3.1 Introduction

The concept of similarity is very important in different domains e.g., mathematics, computer science, biology, management, medicine, meteorology, psychology, etc. In each field, the definition of similarity is personalized. According to the Merriam Webster English dictionary, the similarity is defined as "quality of being similar, resemblance, like, alike," and refers to: (1) having characteristics in common, (2) alike in substance or essentials; or (3) not differing in shape but only in size or position. In psychology, the similarity refers to the degree to which people classify two objects as similar depending on their experience, knowledge and behavior.

With respect to the above definition, a similarity measure has to take into consideration the characteristics or building blocks of the objects (i.e., behavior in form of attribute, structure, shape, etc.) to be compared. However, the degree of having the commonality, likeness in building blocks or not differing in shape is subjective. Hence, it is not easy to compare the quality of two different similarity measures.

In multimedia context, shape is one of the basic features used to represent an object; and objects having similar shape (SYEDA-MAHMOOD, T. et al., 2010) might be considered similar (supporting definition 3). Similarly, in structure-base XML retrieval, documents having the same structure are considered as similar.

One of the earliest approaches to measure the similarity between a pair of objects is a geometric model. In this model, objects are represented as points in some coordinate space (multi-dimensional space) such that the inverse of the distance separating these points represent the similarity value. In this model, a metric distance function *d* assigned to every pair of points a non negative number satisfying the following three axioms:

a) Minimality: $d(A,B) \geq d(A,A) = 0$
b) Symmetry: $d(A,B) = d(B,A)$
c) Transitivity/Triangular inequality: $d(A,B) + d(B,C) \geq d(A,C)$

Based on the universal law of generalization proposed by Shepard in (SHEPARD, R. N., 1987), distance and similarity are related via an exponential function. Hence, the closer the objects, the higher is the similarity. It is denoted as:

$$sim(c_1, c_2) = e^{-d(c1,C2)} \tag{3.1}$$

The similarity value is a number between 0 and 1. The following basic properties are extracted from the distance axioms.

a) Self-similarity: $Sim(A, A) = 1$.
   i.e., $sim(A, A) = e^{-d(A,A)} = e^{-0} = 1$
b) Sim(A,B) = 0, A and B shares nothing in common
c) Maximality: $sim(A, B) \leq sim(A, A)$, the similarity between a pair of objects is less than self-similarity value.
   i.e., $sim(A, B) = e^{-d(A,B)} \leq e^{-d(A,A)} = 1$

The following two properties are arguable by differnt researchers.

d) Symmetry: Sim(A,B) = Sim(B,A).
   i.e., $sim(A, B) = e^{-d(A,B)} = e^{-d(B,A)} = sim(B, A)$
   The similarity between A and B is same as the similarity between B and A.
e) Transitivity: $Sim(A, B) \wedge Sim(B, C) \Rightarrow Sim(A, C)$

   i.e., if A is similar to B and B is similar to C, then A is similar to C.

In the research community, the validity of the similarity properties symmetry and transitivity are arguable and are domain dependent. For instance, the similarity of *car* to *vehicle* is greater than the similarity of *vehicle* to *car*; in (TVERSKY, A., 1977), Tversky reported that most people judge the similarity of *son* to *father* to be greater than the similarity of father to son; and the similarity of *North Korea* to *China* to be greater than the similarity of *China* to *North Korea*. The validity of triangular inequality in similarity is challenged with an example reported by James (JAMES, W., 1890): consider the similarity between countries: Jamaica is similar to Cuba (because of geographical

proximity); *Cuba* is similar to *Russia* (because of their political affinity); but *Jamaica* and *Russia* are not similar at all. Tversky (TVERSKY, A., 1977) noticed that the geometric model is not capable to represent all kind of objects.

The other approach in similarity analysis is the use of feature tree (TVERSKY, A., 1977) as a representational model and each object is viewed as a node representing a set of features. A feature, represented as a node of a tree, denotes the characteristics of an object; it is shared by other objects that follow the arc (edge) that connect them. The similarity between a pair of objects is computed as a ratio/function of the commonality and difference existing between the objects. We follow the ratio model similarity approach in computing the similarity between concepts as detailed in Section 3.3.

However, similarity without semantic or contextual information returns a less relevant result. Noticing this fact a number of researches (c.f. review on concepts similarity in Section 2.4) have been accomplished to reduce the gap existing between the objects to be compared. It is to be recalled that the use of semantic information (review on concepts similarity in Section 2.4) improves the relevance of similarity result. But, the concept measures either consider only one relation ISA. In this chapter, we provide a generic, easily configurable and extensible measure. In addition, we provide bottom-up based approach to aggregate the relatedness between basic components to get relatedness at higher level.

The rest of this chapter is organized as follows: In Section 3.2, we define the basic notions used in the chapter such as feed data model, Knowledge Base and related concepts. In Section 3.3, we detail our concept similarity measure. Section 3.4 presents text representation and relationships identification followed by our text relatedness approach. Section 3.6 presents our feed relatedness algorithms. In Section 3.7, we present the computational complexity of our relatedness algorithms. We conclude the chapter by providing the summary in Section 3.8.

3.2 Preliminaries

As described in Section 2.2, a news feed is represented as unordered collection of XML elements/nodes where each node corresponds to an element having a name, content and zero or more attributes. An element with only a textual value is a simple element otherwise, it is a complex element. Notice that, we disregard other types of nodes such as comment, entity, processing instruction, as they do not contain basic information related to the feed news items.

3.2.1 News feed data model

A news feed is an XML document formatted with either RSS (with its different versions) or Atom for the purpose of publishing and distributing a news item. The various versions of RSS consistently follow the same overall structure, adding or removing certain elements depending on the version at hand (for instance element source is part of RSS 0.9x while guid is in RSS 2.0). The two popular currently used feed formats are RSS 2.0 and Atom 1.0 which have different structures caused by the use of different tag names as shown in Table 1.1.

Notice that, in this report, RSS refers to any web feed formatted with either RSS 2.0 or Atom 1.0.

Definition 3.1 [Rooted Labeled Tree]
A *rooted labeled tree T* is a set of $(k + 1)$ nodes $\{r, n_i\}$, with $i = 1, \ldots, k$. The root of T is r and the remaining nodes $n_1, \ldots, n_k$ are partitioned into m sets $T_1, \ldots, T_m$, each of which is a tree. These trees are called sub-trees of the root of T. ■

Figure 3.1 represent tree definition. The RSS tree depicting news item *CNN1* of Figure 1.1 is shown in Figure 3.2.

$$T ::= \{r, \{Elt\}\}$$
$$Elt \quad ::= sTag\ Content\ eTag$$
$$sTag ::= "\langle"Name\ Attribute * "\rangle"$$
$$Attribute ::= attName\ "="AttValue$$
$$Content ::= string\ |Elt +$$
$$eTag \ ::= \langle/Name\rangle$$
$$r \mid Name \mid attName\ |AttValue ::= string$$

Figure 3.1: Definition of Tree

Notice that, in this report the term tree means rooted labeled and unordered tree.

Definition 3.2 [Element]

Each node of the rooted labeled tree *T* is called an *element* of *T*. Each element *e* in Figure 3.1 has a name, content and zero or more attributes. Given an element *e*, *e.name*, *e.content* and *e.attributes* refers to the name, content and attributes respectively. The name of an element is generally an atomic text value (i.e., a single word/expression), whereas the content may assume either an atomic text value, a composite text value (sentence, i.e., a number of words/expressions), or other elements. An attribute has a name and value and both assume atomic text value. ■

Definition 3.3 [Simple/Composite Element]

An element *e* is *simple* if *e.content* assumes either an atomic or composite textual value[35]. In XML trees, simple elements come down to leaf nodes.

> For instance, <title>Ministers among Somalia blast dead</title> of RSS item *CNN1* is a simple XML element having e.name = "*title*" and *e.content* = "Ministers among Somalia blast dead".

An element *e* is *composite* if *e.content* assumes other elements. In XML trees, composite

[35] In this report, we do not consider other types of data contents, e.g., numbers, dates, … since RSS is mainly composed of textual data.

elements correspond to inner nodes. ■

For instance, the element *CNN1* in Figure 1.1, <item><title>Ministers among Somalia blast dead</title><guid> http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition</ guid><link> http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition </link><description>An explosion at a graduation ceremony in the Somali capital Thursday killed at least 15 people, including three government ministers and nine students, local journalists told CNN.</description><pubDate>Thu, 03 Dec 2009 07:27:47 EST</pubDate> </item>, contains *title*, *guid*, *link*, *description* and *pubDate* as children and hence it is a composite element.

Definition 3.4 [RSS Item Tree]

An *RSS item tree* is a tree *T* having one composite element, the root node *r* (usually with *r.name* = '*item*' or *r.name* = '*entry*'), and k simple elements $\{n_1, \ldots, n_k\}$ describing the various RSS item components. ■

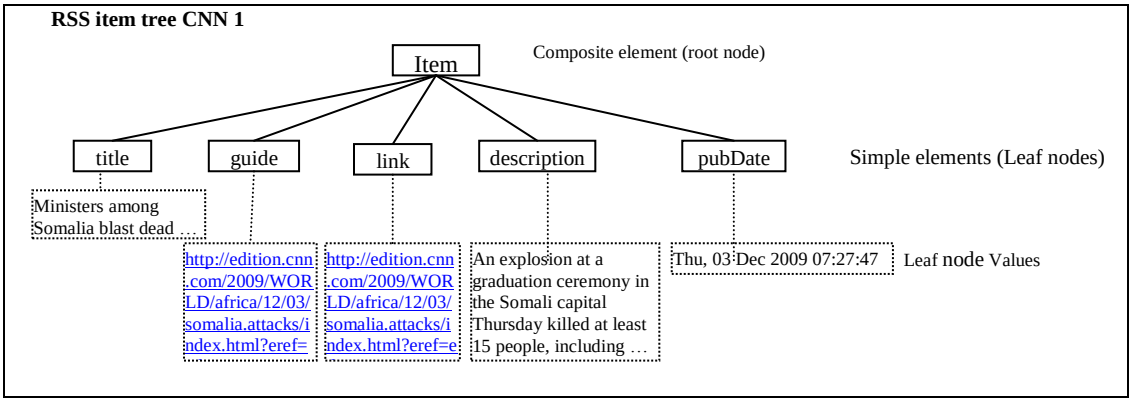


Figure 3.2: Tree representation of RSS item CNN1 in Figure 1.1

3.2.2 Knowledge Base

Knowledge Bases (KB) also called semantic networks (RICHARDSON, R. and Smeaton, A.F., 1995; LIN, D., 1998; JIANG, J. J. and Conrath, D. W., 1997) (thesauri, taxonomies

and/or ontologies) provide a framework for organizing entities (words/expressions, generic concepts, web pages, etc.) into a semantic space.

The use of application dependent Knowledge Base (KB) facilitates and improves the relatedness result. In attempt to provide a generic and extensible solution that eases the structural and content heterogeneity problems in document similarity, we introduce two types of Knowledge Bases:

(i) *value-based* VKB: used to describe the textual content of RSS elements, and

(ii) *label-based* LKB: used to organize RSS labels.

Note that, as the content of an element and its label are textual values, one single Knowledge Base could have been used. But, since RSS labels might belong to different versions, formats and can also be defined by applications or users following a user defined document schema, an independent *label-based* Knowledge Base seems more appropriate than a more generic one such as WordNet (WORDNET 2.1, 2005) (adequate for treating generic textual content). Formally, KB is defined as follows.

Definition 3.5 [Knowledge Base]

A Knowledge Base *KB* is a collection of concepts *C* in semantic network, related with semantic relationship *R*, i.e.,

$$KB = (C, E, R, f) \tag{3.2}$$

where:

- *C* is the set of concepts (a concept is a set of synonymous words/terms/expressions) or synonym sets as in WordNet (WORDNET 2.1, 2005))
- *E* is the set of edges connecting the concepts, where $E \subseteq C \times C$
- *R* is the set of semantic relations, $R = \{\equiv, \prec, \succ, \ll, \gg, \Omega\}$ the synonymous

term/words/expressions being integrated in the concepts. The symbols in R underline respectively the synonym (SYN or $\equiv$), hyponym (IsA or $\prec$), hypernym (HasA or $\succ$), meronym (PartOf or $\ll$), holonym (HasPart or $\gg$) and Antonym (OPP or Ω) relations, as defined in (GETAHUN, F. et al., 2007) and presented in Section 2.4.1)

- f is a function designating the nature of edges in E, $f: E \rightarrow R$. ■

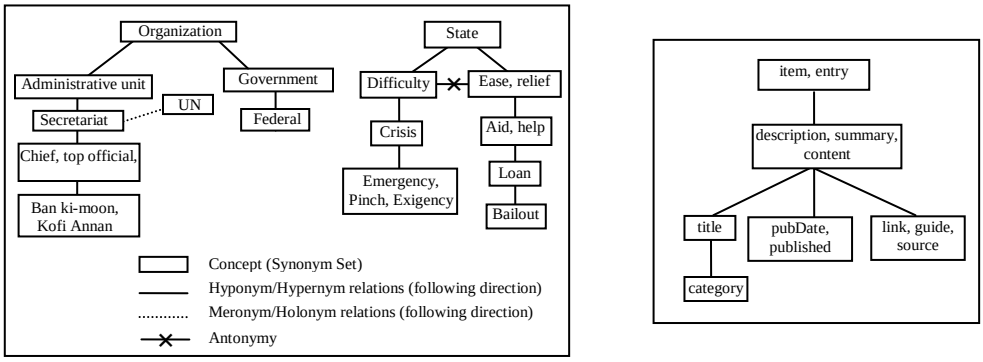
Notice that, in value Knowledge Base, we consider that each value concept designates a certain meaning, and thus is made of the set of synonymous words/expressions corresponding to that meaning (cf. Figure 3.3.A, *Emergency*, *Pinch*, *Exigency* are synonyms and share the same meaning).

Figure 3.3.B shows a sample example of a label Knowledge Base, built using the most popular labels extracted from RSS 2.0 and Atom 1.0. This Knowledge Base assists measuring the relatedness between a pair of heterogeneous news items. Referring to the label Knowledge Base in Figure 3.3.B, *description*, *summary* and *content* have the same meaning.

3.2.2.1 Neighborhood

In our approach, the *neighborhood* of a concept C_i underlines the set of concepts $\{C_j\}$, in the Knowledge Base, that are subsumed by C_i w.r.t. a given semantic relation. It is exploited in identifying the relatedness between texts (i.e., RSS element labels and/or textual contents) and consequently RSS elements/items. In our previous work (GETAHUN, F. et al., 2007), we used the *neighborhood* concept to identify implication between textual values, operators, and consequently semantic predicates (e.g., predicate *Location="Paris"* implies *Location Like "France"*) in uncontrolled space (i.e., the neighborhood threshold is equal to the maximum depth of the Knowledge Base). We noticed that *neighborhood* in unrestrained depth/distance relates unrelated or highly dissimilar concepts through the root of the Knowledge Base. Here, we extend this

approach (GETAHUN, F. et al., 2007) and adopt three types of *neighborhoods*: *semantic neighborhood*, *global semantic neighborhood* and *restricted global semantic neighborhood.*



A) Sample value Knowledge Base - VKB, with multiple root concepts, extracted from WordNet.

B) Sample RSS label Knowledge Base – LKB

Figure 3.3: Sample value and label Knowledge Bases.

Definition 3.6 [Semantic Neighborhood]

Given a Knowledge Base KB, a threshold[36] ε and a semantic relation $r \in \{\equiv, \prec, \ll\}$, the *semantic neighborhood* of a concept C_i within ε is defined as the set of concepts C_j in Knowledge Base KB related with the relation r either directly or transitively[37]. It is formally denoted as:

$$N^{r}_{KB,\varepsilon}(C_i) = \{C_j | C_i \; r \; C_j, dist(C_i, C_j) \leq \varepsilon\} \quad (3.3)$$

where, the function *dist* returns the distance between the concepts and it might refers to hop count or path length.

Notice that, if there are several paths that connect the two concepts we always took the shortest path. ■

[36] A threshold value refers to the number of hops or the path length separating two concepts.

[37] Notice that, the transitivity property between semantic relationships is not necessarily limited to only the semantic relationship type.

Example 3.1: Applying the semantic neighborhood to some of the concepts existing in the value Knowledge Base, VKB, in Figure 2.2, we have:

$N^{\equiv}_{VKB,0}(emergency) = \{\{emergency, pinch, exigency\}\}$

$N^{\prec}_{VKB,1}(emergency) = \{\{crisis\}\}$

$N^{\ll}_{VKB,1}(emergency) = \{\emptyset\}$

$N^{\ll}_{VKB,2}(Windshield)$

$$= \{\{car, auto, automobile\}, \{motor\ vehicle, automotive\}, \{plane, aeroplane, airplane\}, \{vehicle\}\}$$

The Meronym relationship between *Windshield* and *Motor Vehicle*, *Automotive* and *Windshield* and *Vehicle* i.e., $\{WindShield\} \ll \{Motor\ vehicle, automotive\}$ and $\{WindShield\} \ll \{vehicle\}$ is indirect and is caused by transitivity between *IsA* and *PartOf* semantic relationships (c.f. Table 2.4 for detail).

Definition 3.7 [Global Semantic Neighborhood]

Given a Knowledge Base KB and a threshold ε, the *global semantic neighborhood* of a concept C_i within ε is the union of its semantic neighborhood defined with the synonymy ($\equiv$), hyponymy ($\prec$) and meronymy ($\ll$) semantic relations altogether within the same threshold ε. Formally:

$$\overline{N_{KB,\varepsilon}}(C_i) = \bigcup_{r \in \{\equiv, \prec, \ll\}} N^{r}_{KB,\varepsilon}(C_i) \quad (3.4)$$

■

Example 3.2: Referring to the value Knowledge Base VKB in Figure 2.2 and using the semantic neighborhood identified in the Example 3.1.

$\overline{N_{VB,1}}(emergency)$

$$= N^{\equiv}_{VKB,1}(emergency) \cup N^{\prec}_{VKB,1}(emergency) \cup N^{\ll}_{VKB,1}(emergency)$$

$\overline{N_{VKB,1}}(emergency) = \{\{emergency, pinch, exigency\}, \{crisis\}\}$

Similarly, global semantic neighborhood of aid would be:

$$\overline{N_{VKB,1}}(Aid) = N^{\equiv}_{KB,1}(Aid) \cup N^{\prec}_{KB,1}(Aid) \cup N^{\ll}_{KB,1}(Aid)$$

$$\overline{N_{VKB,1}}(Aid) = \{\{Aid, Help, Assistance\}, \{Asset, plus\}\}$$

Definition 3.8 [Restricted Global Semantic Neighborhood]

Given a Knowledge Base KB, a threshold ε and a semantic relationship set R, the *global semantic neighborhood* of a concept C_i *restricted* to the R (where R is a set restricted to the synonymy ($\equiv$), hyponymy ($\prec$) and/or meronymy ($\ll$) semantic relations altogether) is the union of its *semantic neighborhoods* defined with the relation r in R within the same threshold. Formally:

$$\overline{N_{KB,\varepsilon}}^{R}(C_i) = \bigcup_{r \in R} N^{r}_{KB,\varepsilon}(C_i) \quad (3.5)$$

■

Notice that, $\overline{N_{KB,\varepsilon}}^{R}(C_i) \subseteq \overline{N_{KB,\varepsilon}}(C_i)$

Example 3.3: Referring to the value Knowledge Base VKB in Figure 2.2 and using the semantic neighborhood identified in the Example 3.1, the restricted global semantic neighborhood of *windshield* restricted to relation R (hyponymy and meronymy) within a distance of 1 is:

$$\overline{N_{VKB,1}}^{\{\equiv,\prec\}}(Windshield) = N^{\equiv}_{VKB,1}(Windshield) \cup N^{\prec}_{VKB,1}(Windshield)$$

$$\overline{N_{VKB,1}}^{\{\equiv,\prec\}}(Windshield) = \{\{Windshield, windscreen\}\} \cup \{\ \}$$

$$= \{\{Windshield, windscreen\}\}$$

Notice that, to facilitate the readability of the report we use the global semantic neighborhood rather than the restricted global semantic neighborhood. In addition, we flatten the result of the neighborhood of a concept (which is a set of sets) to a flat set.

Hence, the global semantic neighborhood of the concept *emergency* in Example 3.2 becomes: $\overline{N_{VKB,1}}(emergency) = \{emergency, pinch, exigency, crisis\}$.

3.3 Our Concept-based similarity

The concept similarity approaches discussed in Section 2.4 share the following two points:

1) are restricted mainly to the semantic relation *IsA*
2) don't identify the relationship between the concepts which is crucial in our context

Our notion of concept similarity measure is defined on *Knowledge Bases*, *semantic neighborhood*, and *concept enclosure*. We define the *concept enclosure* as follows.

Definition 3.9 [Concept Enclosure]

Given two concepts C_1 and C_2, a threshold ε, and a Knowledge Base KB, C_1 encloses C_2 if the global semantic neighborhood of C_1 within a threshold of i includes the global semantic neighborhood of C_2 within a threshold j $(0 \leq i, j \leq \varepsilon)$ i.e, C_1 encloses C_2 if $\overline{N_{KB,i}}(C_1) \supset \overline{N_{KB,j}}(C_2)$ ■

Definition 3.10 [Ratio Similarity]

Given two concepts C_1 and C_2, and two threshold values i and j associated respectively to C_1 and C_2 and a Knowledge Base KB. The *ratio similarity* between these concepts is defined as a function of the number of common and different concepts of their global semantic neighborhoods. It is denoted as $SimAt(C_1, C_2)_{KB,i,j}$ is defined as:

$$SimAt(C_1, C_2)_{KB,i,j}$$

$$= \frac{\left|\overline{N_{KB,i}}(C_1) \cap \overline{N_{KB,j}}(C_2)\right|}{\left|\overline{N_{KB,i}}(C_1) \cap \overline{N_{KB,j}}(C_2)\right| + \alpha\left|\overline{N_{KB,i}}(C_1) - \overline{N_{KB,j}}(C_2)\right| + \beta\left|\overline{N_{KB,j}}(C_2) - \overline{N_{KB,i}}(C_1)\right|} \quad (3.6)$$

where, $0 \leq \alpha, \beta \leq 1$ and $\alpha + \beta = 1$

Definition 3.11 [Enclosure Similarity]

Given two concepts C_1 and C_2, two threshold values i and j associated respectively to C_1 and C_2 and a Knowledge Base KB. The enclosure similarity between C_1 and C_2 within threshold of i and j is defined as their ratio similarity when $\alpha = 0$. i.e.,

$$EnclosureSimAt(C_1, C_2)_{KB,i,j} = \frac{\left|\overline{N_{KB,i}}(C_1) \cap \overline{N_{KB,j}}(C_2)\right|}{\left|\overline{N_{KB,j}}(C_2)\right|} \quad (3.7)$$

Definition 3.12. [**Similarity**]

Given two concepts C_1 and C_2 and a threshold ε, the *Similarity* between C_1 and C_2 within ε is computed as the maximum enclosure similarity between C_1 and C_2 while varying their neighborhood threshold value between 0 and ε. It is denoted as $Sim_{Enclosure}(C_1, C_2, \varepsilon)$ is defined as:

$$Sim_{Enclosure}(C_1, C_2, \varepsilon) = \max_{0 \leq i,j \leq \varepsilon} \{EnclosureSimAt(C_1, C_2)_{KB,i,j}\} \quad (3.8)$$

■

Our enclosure similarity measure shown in Equation (3.8), is asymmetric. It returns a value of 1 if the two concepts are synonymous or C_1 enclose C_2.

This measure correlates more to the human concept rating (c.f. the relevance of our enclosure measure in Section 6.5.3.1). In addition, it helps us later to identify the similarity and the relationship existing textual values.

Notice that the computation of enclosure similarity ($Sim_{Enclosure}$) is based on maximum similarity value and takes into consideration concepts related with equality, inclusion, overlapping and disjointness relationship.

Example 3.4: Referring to the Knowledge Base KB in Figure 2.2, the enclosure similarity between the concepts *Emergency* and *Crisis* within a threshold of 1 is denoted as:

$$Sim_{Enclosure}(Crisis, Emergency, 1) = \max_{0 \le i,j \le 1} \left(EnclosureSimAt(Crisis, Emergency)_{KB,i,j}\right)$$

Figure 3.4 shows the global semantic neighborhood of *Emergency* and *Crisis* within a threshold of 1 (i.e., global semantic neighborhood at threshold of 0 and 1).

$\overline{N_{KB,0}}(Emergency) = \{Emergency, Exigency, Pinch\}$

$\overline{N_{KB,1}}(Emergency) = \{Emergency, Exigency, Pinch, Crisis\}$

$\overline{N_{KB,0}}(Crisis) = \{Crisis\}$

$\overline{N_{KB,1}}(Crisis) = \{Crisis, Situation\}$

The enclosure similarity of these concepts is computed by varying path length and the result is shown in Table 3.1.

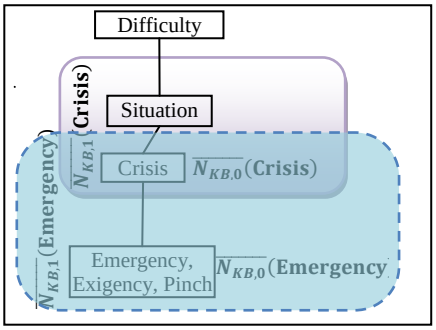


Figure 3.4: Global Semantic neighborhood of *Emergency* and *Crisis* within a threshold of 1

Table 3.1: Enclosure similarity between *Crisis* and *Emergency* within a threshold of 1

A) $EnclosureSimAt(Crisis, Emergency)_{KB,i,j}$

i \ j	0	1
0	0/3	1/3
1	0/3	1/3

B) $EnclosureSimAt(Emergency, Crisis)_{KB,i,j}$

i \ j	0	1
0	0/1	0/2
1	1/1	1/2

Referring to Table 3.1.A, the enclosure similarity between *Crisis* and *Emergency* is:

$$Sim_{Enclosure}(Crisis, Emergency, 1) = \max_{0 \le i,j \le 1} \left(EnclosureSimAt(Crisis, Emergency)_{KB,i,j}\right)$$

$$Sim_{Enclosure}(Crisis, Emergency, 1) = \max\{0/1\,, 1/3\,, 0/3\,, 1/3\} = 0.33$$

Similarly using Table 3.1.B, the enclosure similarity between *Emergency* and *Crisis* within a threshold of 1 is denoted as:

$$Sim_{Enclosure}(Emergency, Crisis, 1) = \max_{0 \le i,j \le 1} \left(EnclosureSimAt(Emergency, Crisis)_{KB,i,j}\right)$$

$$Sim_{Enclosure}(Emergency, Crisis, 1) = max\{0/1\,, 0/2\,, 1/1\,, 1/2\} = 1$$

Notice that, as the $Sim_{Enclosure}$(*Emergency*, *Crisis*,1) is 1 and greater than $Sim_{Enclosure}$(*Crisis*, *Emergency*,1). This value shows that *Emergency* is more similar to *Crisis* as it shares lot of features with Crisis than the reverse (i.e., *Crisis* exhibit some distinct behavior than the lower concept *Emergency*). This shows the asymmetric nature of our measure.

Example 3.5: Identify the similarity between two words *Ambulances* and *Bicycles* within a threshold of 4.

Measuring the similarity between two words comes down to measuring the enclosure similarity of their corresponding concepts. Referring to the Knowledge Base shown in Figure 2.2, measuring the similarity between words starts with mapping each word to the best concept that represents it. Hence, the similarity between the words is computed using

the enclosure similarity between their best concepts. Here, *Ambulances* is mapped to *Ambulance* and *Bicycles* to *Bicycle*. Table 3.2 shows the enclosure similarity between the words within various threshold values.

Table 3.2: Enclosure similarity between two words at different threshold values

A) $\text{EnclosureSimAt}(\text{Ambulances}, \text{Bicycles})_{KB,i,j}$

<Ambulance, i> \ <Bicyle, j>	0	1	2	3	4
0	0/4	0/5	0/6	0/7	0/8
1	0/4	0/5	0/6	0/7	0/8
2	0/4	0/5	0/6	0/7	0/8
3	0/4	0/5	0/6	0/7	0/8
4	0/4	1/5	1/6	1/7	1/8

B) $\text{EnclosureSimAt}(\text{Bicycles}, \text{Ambulances})_{KB,i,j}$

<Bicyle, i> \ <Ambulance, j>	0	1	2	3	4
0	0/1	0/4	0/6	0/7	0/8
1	0/1	0/4	0/6	0/7	1/8
2	0/1	0/4	0/6	0/7	1/8
3	0/1	0/4	0/6	0/7	1/8
4	0/1	0/4	0/6	0/7	1/8

Using Table 3.2.A, we show the enclosure similarity at different levels/thresholds,

$$Sim_{Enclosure}(Ambulances, Bicycles, 4) = \max_{0 \le i,j \le 4} \{EnclosureSimAt(Ambulances, Bicycles)_{KB,i,j}\}$$

$Sim_{Enclosure}(\text{Ambulances}, \text{Bicycles}, 4) = 1/5 = 0.2$

Similarly, using Table 3.2.B,

$$Sim_{Enclosure}(Bicycles, Ambulances, 4) = \max_{0 \le i,j \le 4} \{EnclosureSimAt(Bicycles, Ambulances)_{KB,i,j}\}$$

$Sim_{Enclosure}(Bicycles, Ambulances, 4) = 1/8 = 0.125.$

This example shows that the '*Ambulances'* is more similar to '*Bicycles'* than '*Bicycles'* and '*Ambulances'*.

3.3.1 Properties of our concept similarity measure

Here, we present the property of our enclosure similarity measure.

Referring to our concept similarity measure provided in Definition 3.12 we identify the following two basic properties which held True:

1. $Sim_{Enclosure}(A, A, \varepsilon) = 1$, the similarity of a concept with itself is 1
2. $Sim_{Enclosure}(A, B, \varepsilon) \neq Sim_{Enclosure}(B, A, \varepsilon)$, the similarity between two different concepts is not same i.e., the similarity measure is asymmetric

Proposition 1. Given two concepts C_1 and C_2, and a threshold ε; C_1 encloses C_2 if and only if the corresponding enclosure similarity is 1.

i.e., $C_1\ encloses\ C_2 \Leftrightarrow Sim_{Enclosure}(C_1, C_2, \varepsilon) = 1$

Proof: To prove this expression, first let us consider the forward expression

Assume that $C_i\ encloses\ C_j$ Wnt[38] $Sim_{Enclosure}(C_1, C_2, \varepsilon) = 1$

C_1 encloses C_2

$\Rightarrow \exists i, j \leq \varepsilon\ /\ \overline{N_{KB,i}}(C_1) \supset \overline{N_{KB,j}}(C_2)$ using Definition 3.9

$\Rightarrow \overline{N_{KB,i}}(C_1) \cap \overline{N_{KB,j}}(C_2) = \overline{N_{KB,j}}(C_2)$ and $\overline{N_{KB,i}}(C_2) - \overline{N_{KB,i}}(C_1) = \emptyset$

$\Rightarrow Sim_{Enclosure}(C_1, C_2, \varepsilon) = \max_{0 \leq i,j \leq \varepsilon} \{\text{EnclosureSimAt}(C_1, C_2)_{KB,i,j}\} = 1$

$\therefore Sim_{Enclosure}(C_1, C_2, \varepsilon)$=1

Let us prove the converse of the expression.

Assuming that $Sim_{Enclosure}(C_1, C_2, \varepsilon) = 1$, wnt $C_1\ encloses\ C_2$

$Sim_{Enclosure}(C_1, C_2, \varepsilon) = 1$

[38] We need to show that.

$\Rightarrow \exists i, j \leq \varepsilon \; / \; \overline{N_{KB,\iota}}(C_1) \cap \overline{N_{KB,J}}(C_2) = \overline{N_{KB,J}}(C_2)$, i.e., there exist two thresholds i and j and the intersection of the global semantic neighborhood is the same as the neighborhood of C_j.

$\Rightarrow 0 \leq j \leq i \leq \varepsilon$ and here there are two cases,

Case 1: $j = i \; \wedge \left|\overline{N_{KB,\iota}}(C_1)\right| = \left|\overline{N_{KB,J}}(C_2)\right| \Rightarrow C_1 = C_2$. i.e., the two concepts are equal. Thus, C_i encloses C_j and also C_j encloses C_i

Case 2: $j \leq i \; \wedge \left|\overline{N_{KB,\iota}}(C_1)\right| > \left|\overline{N_{KB,J}}(C_2)\right| \Rightarrow \overline{N_{KB,\iota}}(C_1) \supset \overline{N_{KB,J}}(C_2)$ i.e.,$C_1 \; encloses \; C_2$

$\therefore C_1 \; encloses \; C_2$ ■

3.4 Text representation and relations

As illustrated previously in the Motivating Section (cf. Section 1.2), assessing the relatedness and identifying the relationships between two RSS items amounts to comparing corresponding elements, which in turn come down to comparing corresponding element labels and textual values (contents). It is to be recalled that, RSS (simple) element labels and contents underline basic text (labels assuming atomic textual values, whereas contents underline sentences, c.f. Definition 3.2). Thus, herewith, we define the idea of *Concept Set* to represent a piece of text. It will be exploited in representing (and consequently comparing) RSS elements labels and contents. We also detail the different relationships that might occur between texts.

Definition 3.13 [Concept Set]

Given a text *T* (i.e., phrase, sentence, etc.), its *Concept Set* denoted as *CS*(*T*), is a set {C_1, …, C_m}, where each C_i represents a concept related to at least a word in *T*. Each concept C_i is assumed to be obtained after several textual pre-processing operations such as stop-

words removal[39], stemming[40], and/or mapping to the value Knowledge Base, and grouping. ■

Example 3.6: The content of the title element from RSS item *CNN4 in* Figure 1.1 "*U.N. chief launches $613M*[41]*Gaza aid appeal"* can be described by the following concept set: $CS(CNN4) = \{\{UN\}, \{chief\}, \{launch\}, \{Gaza\}, \{aid\}, \{appeal\}\}$.

Similarly, the concept set for the content of the title element from RSS item *BBC3* "*UN launches $613m appeal for Gaza"* is described as: $CS(BBC3) = \{\{UN\}, \{launch\}, \{appeal\}, \{Gaza\}\}$

Definition 3.14 [Concept Membership]

Given a concept *C* and a *Concept Set CS*, *C belongs* to *CS* denoted as $\boldsymbol{C \in CS}$, if *C* exists as member of one of the concepts in the concept set *CS*. ■

Example 3.7: The concept *Gaza belongs to* the *Concept Set* of the content of title in CNN4 of the Figure 1.1 (c.f. Example 3.6),

i.e., $Gaza \in \{\{UN\}, \{chief\}, \{launch\}, \{Gaza\}, \{aid\}, \{appeal\}\}$.

Definition 3.15 [Global Semantic Neighborhood of Concept Set]

The global semantic neighborhood of a *Concept Set CS* within a threshold of ε is the union of the global semantic neighborhoods of its concepts within the same threshold ε.

[39] Stop-words identify words/expressions which are filtered out prior to, or after processing of natural language text which is done using stop list (e.g., *a, an, so, the*, …). However, those words that would change the meaning of the text such as but, not, neither, nor … are not considered as stop words.

[40] Stemming is the process for reducing inflected (or sometimes derived) words to their stem, i.e., base or root (e.g., "*housing*", "*housed*" → "*house*").

[41] The concept set of a text considers only textual values and hence other types of values are ignored.

$$\overline{N_{KB,\varepsilon}}(CS) = \bigcup_{c \in CS} \overline{N_{KB,\varepsilon}}(c) \tag{3.9}$$

Definition 3.16 [Text Disjointness]

Given two texts T_1 and T_2, and a threshold ε; they are *disjoint*, denoted as $T_1 \bowtie T_2$, if the global semantic neighborhoods of these concept sets within the same threshold ε doesn't intersect. Formally:

$$T_1 \bowtie T_2 \; if \; \overline{N_{KB,\varepsilon}}\big(CS(T_1)\big) \cap \overline{N_{KB,\varepsilon}}\big(CS(T_2)\big) = \emptyset \tag{3.10}$$

■

Example 3.8: The title texts of RSS items *CNN1* and *CNN2* in Figure 1.1, described respectively by the following *Concept Sets*:

$CS(CNN1) = \big\{\{Minster\}, \{Somalia\}, \{blast\}, \{dead\}\big\}$ and

$CS(CNN2) = \big\{\{Bin\ Laden\}, \{Pakistan\}, \{PM\}, \{say\}\big\}$, are disjoint as their global semantic neighborhoods (within threshold of 1for instance) do not overlap.

Definition 3.17 [Text Overlapping]

Given two texts T_1 and T_2, and a threshold ε; T_1 overlap with T_2, denoted as $T_1 \cap T_2$, if the global semantic neighborhoods of these concept sets within the same threshold ε overlap/intersect. Formally:

$$T_1 \cap T_2 \; \; if \; \; \overline{N_{KB,\varepsilon}}\big(CS(T_1)\big) \cap \overline{N_{KB,\varepsilon}}\big(CS(T_2)\big) \neq \emptyset \tag{3.11}$$

■

Example 3.9: The title texts of *CNN5* in Figure 1.1 and *BBC4* in Figure 1.3, described respectively by the following *Concept Sets*:

$CS(CNN5) = \big\{\{Al-Jazeera\}, \{Cameraman\}, \{home\}, \{Gitmo\}\big\}$ and

$CS(BBC4) = \big\{\{Free\}, \{Guantanamo\}, \; \{prisoner\}, \{home\}\big\}$, overlap since they have:

- an identical concept *home* and
- common synonyms *Guantanamo* and *Gitmo*

As a result, the global semantic neighborhood of their concept sets (within a threshold of 0 for instance) overlap.

Definition 3.18 [Text Inclusion]

Given two texts T_1 and T_2, and a threshold ε; T_1 *include* T_2, denoted as $T_1 \supset T_2$, if the global semantic neighborhood of $CS(T_2)$ is included in the global semantic neighborhood of $CS(T_1)$ within the same threshold ε. Formally:

$$T_1 \supset T_2 \; if \; \overline{N_{KB,\varepsilon}}\big(CS(T_1)\big) \supset \overline{N_{KB,\varepsilon}}\big(CS(T_2)\big) \tag{3.12}$$

■

Example 3.10: The text T_1: "Hong Kong Cheer Olympic Torch" and T_2: "Hong Kong Cheer Torch", described respectively by the following *Concept Sets*:

$CS(T_1) = \big\{\{Hong\ Kong\}, \{Cheer\}, \{Olympic\}, \{Torch\}\big\}$ and

$CS(T_2) = \big\{\{Hong\ Kong\}, \{Cheer\}, \{Torch\}\big\}$.

T_1 include T_2 as the global semantic neighborhood of $CS(T_1)$ includes the global semantic neighborhood of $CS(T_2)$ within a threshold of 0 for instance.

Definition 3.19 [Text Equality]

Given two texts T_1 and T_2, and a threshold ε; T_1 is equal to T_2, denoted as $T_1 = T_2$, if $T_1 \supset T_2$ and $T_2 \supset T_1$. In other words, $T_1 = T_2$ if the global semantic neighborhoods of their concept sets within the same threshold ε are equal. Formally:

$$T_1 = T_2 \; if \; \overline{N_{KB}}(c) = \overline{N_{KB}}\big(CS(T_2)\big) \tag{3.13}$$

■

Example 3.11: The title texts of *CNN2* in Figure 1.1 and *BBC2* in Figure 1.3, described respectively by the following *Concept Sets*:

$CS(CNN2) = \big\{\{Bin\ Laden\}, \{Pakistan\}, \{PM\}, \{say\}\big\}$ and

$CS(BBC2) = \{\{Bin\ Laden\}, \{Pakistan\}, \{say\}, \{PM\}\}$, are equal as their corresponding global semantic neighborhood at threshold of 0 are identical.

Definition 3.20 [Text Representation Model]

Given two texts T_1 and T_2 described by their respective *Concept Set* $CS(T_1)$ and $CS(T_2)$, we represent each text T_i using the vector space model in information retrieval. A vector $\vec{V}(T_i)$ is represented in an *n*-dimensional space with one component in the vector for each concept in the concept set of both texts. The vector space dimension represents distinct concepts as axis, associated with a weight score and denoted as:

$$\vec{V}(T_i) = \langle w_1, \cdots, w_m, \cdots, w_n \rangle \tag{3.14}$$

where: w_m is the weight score associated to concept $C_m \in \big(CS(T_1) \cup CS(T_2)\big)$, $1 \leq mn$ and $n = |CS(T_1) \cup CS(T_2)|$

The weight w_m associated to a concept C_m in $\vec{V}(T_i)$ (where i =1or 2) is calculated as the maximum *enclosure similarity* it has with another concept C_j from the *Concept Set* of the other text - T_j (j = 2, if i =1, otherwise j = 1). Notice that w_m = 1 if the concept C_m is member of the *Concept Set* of the text T_i, i.e., $CS(T_i)$. Formally, it is defined as:

$$w_m = \begin{cases} 1 & \text{if } C_m \in CS(T_i) \\ \max\limits_{j=1..|CS(T_j)|} \Big(Sim_{Enclosure}\big(C_m, C_j, \varepsilon\big)\Big) \wedge C_j \in CS(T_j) & otherwise \end{cases} \tag{3.15}$$

where, $Sim_{Enclosure}$ is the enclosure similarity measure (c.f. Definition 3.12)

Example 3.12: Let us consider T_1: "Ford Motor reported that its ongoing losses soared in the fourth quarter, but the company reiterated it still does not need the federal bailout already received by its two U.S. rivals." and T_2: "US carmaker Ford reports the biggest full-year loss in its history, but says it still does not need government loans". The corresponding vector representations $\vec{V_1}$ and $\vec{V_2}$ are shown in Figure 3.5.

	Ford	Motor	report	ongo	loss	soar	Fourth-quarter	company	reiterate	still	need	Federal	Bailout	receive	US	…	Big	Full-year	history	say	govemment
$\overrightarrow{V_1}$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		1	0.5	0	**1**	**0.5**
$\overrightarrow{V_2}$	1	1	1	0	1	0.33	**1**	0	**1**	1	1	**1**	**1**	0	1		0	1	1	1	1

Figure 3.5: Vectors obtained when comparing two texts

Vector weights are evaluated in two steps:

- First, for each concept C_i in $\overrightarrow{V_1}$ i.e., $\vec{V}(T_1)$; and $\overrightarrow{V_2}$, i.e., $\vec{V}(T_2)$ we check the existence of C_i in each of the concept sets corresponding to the texts being compared.
- Second, we update the weight of those concepts having value of zero with maximum *enclosure similarity* value.

Following the WordNet subset extract in Figure 2.2, the concept Government is included in the global semantic neighborhood of *Federal* within threshold of 1, i.e.,

$government \in \overline{N_{KB,1}}(federal)$. Hence, it has the maximum enclosure similarity value with *federal*, i.e., $Sim_{Enclosure}$ (*federal, government*,1) = 1 in $\overrightarrow{V_2}$, However, in $\overrightarrow{V_1}$, $Sim_{Enclosure}$(*government, federal*,1) = 0.5 .

Similarly, *loan* is included in the global semantic neighborhood of *bailout* within threshold of 1, i.e., $loan \in \overline{N_{KB,1}}(bailout)$. In this way, $Sim_{Enclosure}$(*bailout, loan*,1) = 1 in $\overrightarrow{V_2}$ and $Sim_{Enclosure}$(*loan, bailout*, 1) = 0.5 in $\overrightarrow{V_1}$.

3.5 Texts relatedness

In RSS document context, both the tag name of an element and the content of a simple element refer to textual values. Hence, texts relatedness refers to either relatedness between element names (tags) or relatedness between contents of simple elements. To accurately capture the semantic relatedness between two textual values, we exploit the classical vector space model used in information retrieval (MCGILL, M. J., 1983),

integrating the notion of semantic neighborhood with our enclosure similarity (cf. Section 3.3). In detail, we proceed as follows.

When comparing two texts T_1 and T_2, each would be represented as a vector $\vec{V}$ ($\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ respectively) with weights underlining concept occurrences and descriptive degrees in their corresponding *Concept* Sets, $CS(T_1)$ and $CS(T_2)$, taking into account global semantic neighborhood.

The texts relatedness algorithm accepts four basic kind of information as parameters (Lines 1 to 4) and returns a tuple containing the semantic relatedness and relationship values. The parameters are:

- the two texts to be compared T_1 and T_2
- two threshold values $T_{Disjointness}$ and T_{Equal}
- semantic ag
- a Knowledge Base that would be used to identify the semantic neighborhood of a concept

The algorithm identifies the concept sets CS_1 and CS_2 of T_1 and T_2 respectively using a function *CS* (Lines 11 to 12, following Definition 3.9), builds the vector space corresponding to T_1 and T_2, computes the cosine measure and identifies the exclusive relationship (i.e., Equal, Include, Overlap or Disjoint). In lines 14 to 19, T_1 and T_2 are represented as vectors $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ respectively with weights underlining concept existence, and maximum similarity in both CS_1 and CS_2. The text relatedness algorithm can be easily tuned to work either with syntactic or semantic similarity. If the semantic flag is set, the procedure *weight* accepts the concept whose weight to be computed C_i, the concept set of the text containing C_i, the concept set of the other text, and a Knowledge Base *KB*, and it returns a weight that reflects the maximum enclosure similarity value computed using Equation (3.15). In computing the weight score of a concept, any semantic similarity measure discussed in Section 2.4 could be used. But, if other

measures are adopted the *include* relationship won't be recognized. In Line 21, the semantic relatedness *SemRel* between two texts is quantified using a measure of the similarity between vectors $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ implemented in *Vector-Based-Similarity-Measure* function. In this study, we use the *cosine measure:*

$$SemRel = SemRel(T_1, T_2) = Cos(\overrightarrow{V_1}, \overrightarrow{V_2}) = \frac{\overrightarrow{V_1} . \overrightarrow{V_2}}{|\overrightarrow{V_1}| \times |\overrightarrow{V_2}|} \in [0,1] \qquad (3.16)$$

Semantic relatedness is consequently exploited in identifying three relations (i.e., *disjoint, overlap* and *equal*) between the two texts. Our method (*Relation* in Lines 22 to 32) for identifying the basic relationships is based on the use of threshold values so as to overcome the often imprecise descriptions of texts. For instance, texts (likewise RSS items) that describe the same issue are seldom exactly identical. They might contain some different concepts, detailing certain specific aspects of the information being described, despite having the same overall meaning and information substance (cf. Chapter 1, Example 1.2). In addition, two texts and news items might shares few concepts, for instance, the content of title element in CNN3 of Figure 1.1, *Bus blast kills, hurts dozens in Syria***,** and the content of title element in BBC3 of Figure 1.3, *UN launches $613m appeal for Gaza,* overlap as the global semantic neighborhood of their corresponding concept sets overlap as the global semantic neighborhood of the concept *Syria* and *Gaza* overlap. But each text addresses totally different issues. Thus, we address the fuzzy nature of textual content in identifying relations by providing pre-defined/pre-computed and user configurable similarity thresholds $T_{Disjointness}$ and T_{Equal}, as shown in Figure 3.6.

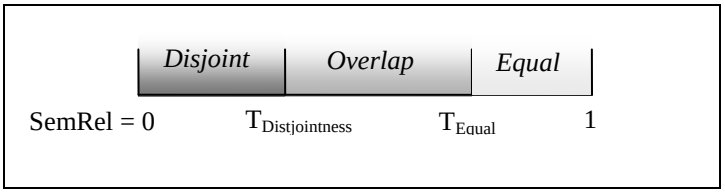


Figure 3.6: Basic text relationships and corresponding thresholds

Given two threshold values $T_{Disjointness}$, T_{Equal} and a similarity value *SemRel*; the relationship (*Equal*, *Overlap*, *Disjoint*) between T_1 and T_2 is identified following Equation (3.17). We suggest using the following rules to identify the basic relationships - Disjoint, Overlap or Equal- existing between two texts T_1 and T_2.

$$Relation(T_1, T_2, T_{Disjointness}, T_{Equality}, SemRel) = \begin{cases} Equal & If\ SemRel \in [T_{Equal}, 1] \\ Overlap & Else\ If\ SemRel \in]T_{Disjointness}, T_{Equal}[\\ Disjoint & Otherwise\ i.e., SemRel < [0, T_{Disjointness}] \end{cases} \quad (3.17)$$

While the relations -*Disjoint*, *Overlap* and *Equal*- can be defined using the semantic relatedness (in the context of fuzzy relations), the *include* relationship is computed differently as follows:

Referring to Definition 3.18 a text T_1 includes another text T_2 if within a given threshold value, every concept in T_2 is included in the global semantic neighborhood of the concept set of T_1. Thus, the corresponding weight score of these concepts in the vector have a value of 1. We define it as:

Relation(T_1, T_2) is *Include*, i.e., $T_1 \supset T_2$, if the product of the weight score w_p of all concepts represented in the vector $\overrightarrow{V_1}$ (describing T_1) is equal to 1, i.e.,

$$Relation(T_1, T_2) = Include\ if \prod_{p=1}^{|CS(T_1)|} w_p = 1 \quad (3.18)$$

where, $\prod w_p$ is the product of weight scores w_p. It underlines whether or not T_1 encompasses all concepts in T_2.

Notice that the relationship between text values is identified on best value following the partial order shown below in Equation (3.19) and implemented in lines 22 to 32 respectively.

$$Equal > Include > Overlap > Disjoint \quad (3.19)$$

Pseudo Code 1: TR Algorithm

```
Input:
1.   T1,T2: String                          // two input texts
2.   TDisjointness, TEqual : Decimal        // threshold values
3.   flag: Boolean                          // consider semantic or not flag
4.   KB: Knowledge_Base                     // semantic knowledge base proposed by the user
Variable:
5.   V1: Vector                            // vector for T1
6.   V2: Vector                            // vector for T2
7.   CS1, CS2: Set                         // concept set of T1 and concept set of T2
8.   C: Set                                // C is concept
Output:
9.   SemRel: Decimal                       //relatedness value between T1,and T2
10.  Rel: String                           //topological relationships between T1,and T2
Begin
11.  CS1 = CS(T1)                          // CS– returns the concept set of the text T1
12.  CS2 = CS(T2)                          // CS – returns the concept set of the text T2
13.  C = CS1 ∪ CS2
14.  V2 = V1 = Vector_Space_Generator(C) //generate vector space having C as concepts
15.  If flag = True Then                   // is semantic flag set?
16.    For each Ci in C
17.      V1[Ci] = Weight(Ci, CS1, CS2, KB)   // computes the weight of concept Ci in V1
18.      V2[Ci] = Weight(Ci, CS2, CS1, KB)   // computes the weight of concept Ci in V2
19.    Next
20.  End IF
21.  SemRel = Vector-Based-Similarity-Measure(V1, V2) //using cosine similarity
22.  If SemRel ≥ TEqual Then
23.     Rel = "Equal"
24.  Else If ∏V2(wp) = 1 Then  // is product of weight values in v2 is 1?
25.     Rel = "Included in"
26.  Else If ∏V1(wp) = 1 Then  // is product of weight values in v1 is 1?
27.     Rel = "Include"
28.  Else If TDisjointness < SemRel < TEqual Then
29.     Rel = "Overlap"
30.  Else If SemRel ≤ TDisjointness Then
31.     Rel = "Disjoint"
```

Pseudo Code 1: TR Algorithm

```
32.     End If
33.     Return ⟨SemRel, Rel⟩
        End
```

Example 3.13: Considering the Example 3.12, vectors shown in Figure 3.5, and using threshold values: $T_{Disjointness}$= 0.1 and T_{Equal}= 0.9.

$$SemRel(T_1, T_2) = Cos(\overrightarrow{V_1}, \overrightarrow{V_2}) = \frac{\overrightarrow{V_1}.\overrightarrow{V_2}}{|\overrightarrow{V_1}| \times |\overrightarrow{V_2}|} = 0.86$$

As in both $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ some of the weight scores have a value of zero and

$SemRel(T_1, T_2) \in]0.1, 0.9[$, the $Relation\ (T_1, T_2) = Overlap$.

Hence, $TR\ (T_1, T_2) = \langle 0.86, Overlap \rangle$.

3.6 RSS relatedness

As discussed previously in Section 3.1, the natural way to compute similarity between two objects is to aggregate the similarity between their corresponding components. Hence, quantifying the semantic relatedness and identifying the relationship between two RSS items amounts to comparing corresponding elements. This in turn comes down to comparing corresponding simple elements labels and contents, which simplifies to basic pieces of text. As a result computing item relatedness and relationship is done by combining the relatedness and relationship value between all sub-elements of the items. However, doing so is complex and might generate irrelevant result. The item relatedness computation involves handling three major challenges:

1. identifying the set of elements that could be used in computing relatedness.
2. computing the semantic relatedness between a pair of elements
3. combining the semantic similarity and relationship value of the elements so as to get the relatedness between the items.

First, we allow a user to specify her personal choice or preference of tag names that would be used in computing the items relatedness and/or creating a link between these items (which is defined hereafter as *Item Connector*).

Definition 3.21 [Item Connector – ic]

Given an item *I*, the *item connector ic* is a collection of tag names whose content is used to measure semantic relatedness value. It is denoted as:

$$ic ::= e_i.name + \ \ /e_i \in I \tag{3.20}$$

Having the set of elements that could be used to compute the items relatedness is only the first step. The computation involves identifying the correspondence, a pair of tag names from *item connector*, to be used in the computation process as defined next.

Definition 3.22 [Item Connector Association Matrix – ic_matrix]

Given two items I_1 and I_2 and an *item connector ic*, item connector association matrix *ic_matrix* is an $N \times N$ binary matrix (contains only zeros or ones) that associates an *ic* to *ic* (i.e., $ic \times ic$). The matrix shows the possible pair of tag names that can be used in computing the semantic relatedness. A value of 1 (True) for a pair (tag_i, tag_j) – tag_i refers to tag name of an element from item I_1 (row) and tag_j refers to tag name of an element in item I_2 (column) - signifies that an element named tag_i and an element named tag_j are chosen to be used in computing relatedness.

$$ic\_matrix = \begin{array}{c} \\ e_{11} \\ \vdots \\ e_{1n} \end{array} \begin{array}{c} \begin{array}{ccc} e_{21} & \cdots & e_{2n} \end{array} \\ \begin{pmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{pmatrix} \end{array}$$

Thus, computing the item relatedness (IR) is related to aggregate the relatedness value of those sub-elements having a value of 1 in the accompanying *ic_matrix*.

In the next two sub-sections, we present the simple elements and items relatedness computation approaches that handle the second and the third challenges stated above.

3.6.1 Simple element relatedness

The relatedness between two simple elements (*ER*) is computed using Pseudo Code 2. It accepts two elements e_1 and e_2 as input (Line 1) and returns a tuple quantifying *SemRel* and *Relation* values between e_1 and e_2 based on corresponding labels and values relatedness.

In lines 9–10, as simple elements are composed of textual values, labels and contents, the semantic relatedness is computed respectively using the *TR* algorithm (Pseudo Code 1), with a dedicated Knowledge Base – label (LKB) and value (VKB) respectively (cf. 3.2.2).

Simple Element relatedness computation involves combining the semantic relatedness and relationship value of their textual contents and labels. This demands the use of a method for combining the label and the value semantic relatedness results, among which the maximum, minimum, average and weighted sum functions are possible candidates. Nonetheless, the latter provides flexibility in performing the match operation and adapting the process w.r.t. the user's perception of element relatedness. In particular, it enables the user to assign more importance to either the label semantic relatedness or value semantic relatedness values. In Line 11, the method E_{SemRel} quantifies the relatedness value between elements, as the *weighted sum* of label (LB_{SemRel}) and content/value (VR_{SemRel}) semantic relatedness, such as:

$$SemRel(e_1, e_2, \alpha) = E_{semRel}(LB_{SemRel}, VR_{SemRel}) = \alpha \times LB_{SemRel} + (1 - \alpha) \times VR_{SemRel} \quad (3.21)$$

where, $\alpha \in [0,1]$: the label similarity tuning weight

Providing different weight parameters would empower the users to customize the RSS similarity computation to the scenario at hand stressing only on the structure, the content or the combination of both while considering semantic or not. For instance:

- for $\alpha = 1$, E_{semRel} will consider only the label semantic relatedness and hence text relatedness wouldn't contribute to the overall elements relatedness. This comes down to the tag similarity (BUTTLER, D., 2004)
- for $\alpha = 0$, E_{semRel} will consider only the text semantic relatedness while ignoring the semantic relatedness between labels. Hence, this comes down to semantic-based content similarity measure in Information Retrieval (BAEZA-YATES, R. and Ribeiro-Neto, B., 1999).

In Line 12, the hard-coded rule-based method $E_{Relation}$ is used for combining label and content relationships as follows:

$$Relation(e_1, e_2) = E_{Relation}(LB_{Relation}, VR_{Relation}) = \begin{cases} Equal & if \quad LB_{Relation} = \text{"Equal"} \ \wedge \ VR_{Relation} = \text{"Equal"} \\ Include & if \quad ((LB_{Relation} = \text{"Include"} \vee LB_{Relation} = \text{"Equal"}) \wedge VR_{Relation} = \text{"Include"}) \vee (LB_{Relation} = \text{"Include"} \wedge VR_{Relation} = \text{"Include"}) \\ Overlap & if \quad LB_{Relation} = \text{"Overlap"} \vee VR_{Relation} = \text{"Overlap"} \\ Disjoint & if \quad LB_{Relation} = \text{"Disjoint"} \vee VR_{Relation} = \text{"Disjoint"} \end{cases} \tag{3.22}$$

- *Relation*(e_1, e_2) is *Equal*, i.e., $e_1 = e_2$, if their corresponding labels and their contents are related with equality.
- *Relation*(e_1, e_2) is *Include*, i.e., $e_1 \supset e_2$, if either the label of e_2 is the redundant of the label of e_1 and the content of e_1 include the content of e_2 or the label of e_1 include the label of e_2 and their contents are related with equality.
- *Relation*(e_1, e_2) is *Overlap*, i.e., $e_1 \cap e_2$, if either their corresponding labels or contents overlap.

- *Relation*(e_1, e_2) is *Disjoint*, i.e., $e_1 \rhd\lhd e_2$, if either their corresponding labels or contents are disjoint.

Pseudo Code 2: ER Algorithm

```
Input:
1.   e1, e2: Element                    // two simple elements
2.   flag: Boolean                      // semantic flag
3.   T_Disjointness , T_Equal : Decimal // threshold values
4.   α: Decimal                         // label similarity tuning weight
Variable:
5.   LB_SemRel, VR_SemRel: Decimal      // label and value semantic relatedness values
6.   LB_Relation, VR_Relation: String   // Label and value relationship values
Output:
7.   SemRel: Decimal                    // relatedness value between e1 and e2
8.   Relation: String                   // relationship value between e1 and e2
Begin
9.   ⟨LB_SemRel, LB_Relation⟩ = TR(e1.name, e2.name, T_Disjointness, T_Equal, flag, LKB)
                                        //relatedness between labels
10.  ⟨VR_SemRel, VR_Relation⟩ = TR(e1.value, e2.value, T_Disjointness, T_Equal, flag, VKB)
                                        //relatedness between values/contents
11.  SemRel = E_semRel(LB_SemRel, VR_SemRel, α)
                                        //E_semRel – combines the label and value relatedness values
12.  Relation = E_Relation(LB_Relation, VR_Relation)
                                        //E_Relation– combines the label and value relationships values
13.  Return ⟨SemRel, Relation⟩
End
```

Notice that, it is very unlikely for labels to be related with *Disjoint* relationship and hence the *Disjoint* relationship between two simple elements is dependent on the *Disjoint* relationship between their contents. Table 3.3 show the summary of applying the hard-coded rules in $E_{Relation}$.

Table 3.3: Summary of heuristic based relationship aggregation rules

$LB_{Relation}$ / $VR_{Relation}$	Equal	Include	Overlap	Disjoint
Equal	Equal	Include	Overlap	Disjoint
Include	Include	Include	Overlap	Disjoint
Overlap	Overlap	Overlap	Overlap	Disjoint
Disjoint	Disjoint	Disjoint	Disjoint	Disjoint

3.6.2 Item Relatedness

Having identified the semantic relatedness and relationships between simple elements, the item relatedness value is identified by combining the semantic relatedness and relationship of their corresponding sub-elements.

Given two RSS items I_1 and I_2, each made of collection of simple elements $\{e_i\}$ and $\{e_j\}$ respectively, the *Item Relatedness* (*IR*) algorithm, Pseudo Code 3, returns a tuple quantifying semantic relatedness, *SemRel*, value as well as the *Relation* between I_1 and I_2 based on corresponding element relatedness (lines 16– 23).

IR algorithm consists of three main steps:

- Step 1 (Line 18) involves identifying the pair of sub-elements that would be used to compute elements relatedness while considering semantic information or not.
- Step 2 (Line 19) involves computing the simple element relatedness value (using ER algorithm in Pseudo Code 2).
- Step 3 (Lines 26 to 27) involves aggregating the simple elements semantic relatedness values and combining simple elements relations to get the item relatedness value.

Step 1 is accomplished with the help of item connector association matrix *ic_matrix* (containing the possible pair of tag names that would be used to compute the item relatedness). Hence, for each element e_i in I_1, look for an element e_j in I_2 having a True or

1 value in the item connector association matrix, *ic_matrix* (Line 18).

In Line 19, the computed semantic relatedness value is aggregated (summed to eij_{SemRel}) so as to compute later on the item semantic relatedness value. Similarly, the semantic relation between the simple elements (e_i and e_j) $eij_{Relation}$ is accumulated temporarily until the relationship between all pair of sub-elements is identified in Line 21. In Line 26, the semantic relatedness value between I_1 and I_2 is computed as the average of the aggregated semantic related values between corresponding element sets of I_1 and I_2:

$$SemRel(I_1,\ I_2) = \frac{\sum_{e_i \in I_1} \sum_{e_j \in I_2} SemRel(e_i, e_j)}{count} \tag{3.23}$$

where, *count* is the number of pair of sub-elements with value of true for $ic\_matrix[e_i.name][e_j.name]$

The relationship between two items is identified with the heuristic based hard-coded rule in $I_{Relation}$ (c.f. Line 27) for combining sub-element relationships stored in $eij_{Relation\_set}$ (which is the relationship between e_i and e_j) as defined below (let i and j be the cardinality of I_1 and I_2 respectively):

- *Relation*(I_1, I_2) is Disjoint, denoted as $I_1 \rhd\lhd I_2$, if all elements $\{e_i\}$ and $\{e_j\}$ are disjoint. i.e.,

  $$Relation(I_1,\ I_2) = Disjoint\ if\ \forall e_1 \in \{e_i\}, \forall e_2 \in \{e_j\}, e_1 \rhd\lhd e_2$$

- *Relation*(I_1, I_2) is *Equal*, denoted as $I_1 = I_2$ if all their elements in $\{e_i\}$ are equal to all those in $\{e_j\}$ i.e.,

  $$Relation(I_1, I_2) = Equal\ if\ i = j\ \wedge \forall e_2 \in \{e_j\}, \exists e_1 \in \{e_i\} / e_1 = e_2$$

- *Relation*(I_1, I_2) is Include, denoted as $I_1 \supset I_2$ if all elements in $\{e_i\}$ include or equal to those in $\{e_j\}$ i.e.,

$$Relation(I_1, I_2) = Include\ if\ i \geq j\ \wedge\ \forall e_2 \in \{e_j\},$$
$$\exists e_1 \in \{e_i\}\ /\ e_1 \supset e_2 \vee e_1 = e_2$$

- *Relation*(I_1, I_2) is *Overlap,* denoted as $I_1 \cap I_2$, if at least a pair of sub-element of these items is related with overlap, equal, or include, i.e.,

$$Relation(I_1, I_2) = Overlap\ if\ \exists e_1 \in \{e_i\}, \exists e_2 \in \{e_j\}\ /\ Relation(e_1, e_2) = r,$$
$$r \in \{Overlap, Equal, Include\}$$

Pseudo Code 3: IR Algorithm

```
Input:
1.    I1, I2 : Item                    // two input items (Complex elements)
2.    flag : Boolean                   // semantic flag
3.    ic[]: String                     //item connector
4.    ic_matrix[][]: Boolean           //determines which elts would be used for similarity
5.    TDisjointness, TEqual : Decimal  // threshold values
6.    α: Decimal                       // label similarity tuning weight
Variable:
7.    SumRel : Decimal                 //accumulate the running sum
8.    eijSemRel : Decimal              // semantic relatedness values ei and ej
9.    eijRelation : String             // relationship value between ei and ej
10.   eijRelation_set : Set            // would contain sub-elements relationship values
11.   count : Integer                  // controls the number of sub_elts
Output:
12.   SemRel: Decimal                  // relatedness value between I1 and I2
13.   Relation: String                 // relationship value between I1 and I2
Begin
14.   SumRel = 0
15.   eijRelation_set = ∅
16.   For each ei In I1
17.    For each ej In I2
18.     If (ic_matrix[ei.name][ej.name] == True) Then
19.        ⟨eijSemRel, eijRelation⟩ =ER(ei, ej, TDisjointness, TEqual, flag, α)
20.        SumRel = SumRel + eijSemRel
21.        eijRelation_set = eijRelation_set ∪ eijRelation
22.        count ++
23.     End If
24.    Next
```

Pseudo Code 3: IR Algorithm

```
    Next
    SemRel = SumRel / count          // average semantic similarity value
    Relation = I_Relation(eij_Relation_set)
    Return ⟨SemRel, Relation⟩
End
```

Example 3.14: Let us consider RSS items *CNN4* and *BBC3* (Figure 1.1 and Figure 1.3 respectively). The corresponding item relatedness is computed as follows: weighting factor of α = 0.5 is assigned to label while evaluating simple element relatedness. Thresholds $T_{Disjointness}$ = 0.1 and T_{Equal} = 0.9 are used in getting the relationship value and using a pair of elements having similar tag name to compute the item relatedness as shown in Table 3.4. The result of computing the simple element relatedness between pair of elements with value of 1 in *ic_matrix* is shown in Table 3.5.

Table 3.4: Sample ic_matrix

CNN4 \ *BBC3*	*title*	*description*
title	1	0
description	0	1

Using Equation (3.23), *SemRel*(*CNN4*, *BBC3*) = (0.908 + 0.832)/2 = 0.87, where count is equal to 2. Notice that, the sub-element relatedness between those with different tag names is not used as the corresponding entry in the ic_matrix is zero.

Table 3.5: Element relatedness matrix

ER	$title_{BBC3}$	$description_{BBC3}$
$title_{CNN4}$	<0.908, Equal>	-
$description_{CNN4}$	-	<0.832, *Overlap*>

$\text{Relation}(\text{CNN4}, \text{BBC3}) = \text{Overlap}$, since

$\text{Relation}(\text{description}_{\text{CNN4}}, \text{description}_{\text{BBC3}}) = \text{Overlap}.$

Hence, $\text{IR}(\text{CNN4}, \text{BBC3}) = \langle 0.87, \text{Overlap} \rangle$.

3.7 Computational complexity

The computational complexity of our relatedness algorithms are identified using the worst case analysis and using the RAM machine (SKIENA, S. S., 1998) (c.f. Annex 2 for detail fundamental assumptions of RAM). Suppose T_1 and T_2 are two texts with their corresponding *concepts sets* CS_1 and CS_2, let n and m be the number of concepts in CS_1 and CS_2 respectively. Let d and n_c refer to the depth of the Knowledge Base and the maximum number of words per concept/synset.

3.7.1 Complexity of enclosure similarity measure

It is to be recalled that the enclosure similarity of a pair of concepts depends on the number of shared and individual concepts of their global semantic neighborhood within various threshold values (cf. Definition 3.7 and Definition 3.8). Given two concepts C_1 and C_2, and a threshold $\varepsilon = d$, to identify the global semantic neighborhood of a concept depends on:

- identifying the global semantic neighborhood of a concept related with the relation $R \in \{\equiv, \prec, \ll\}$ within a given threshold i, $0 \leq i \leq d$, involves $(1 + 2 \times i) \times n_c$ time.
- identifying the number of concepts shared by the global semantic neighborhood of C_1 within threshold i and that of C_2 within threshold j, depends on the size of the global semantic neighborhoods which is $n_c{}^2(1 + 2 \times i)(1 + 2 \times j)$. Assuming that, the number of words in a concept is uniform (i.e. n_c) which might not be always true.

The enclosure similarity between the C_1 and C_2 within a maximum threshold d is the maximum enclosure similarity the concepts have while varying the threshold between 0 and d for each concept (c.f. Equation (3.8)). For a fixed threshold i, it involves computing the enclosure similarity of C_1 within i while varying threshold j of C_2 between 0 and d. It involves:

$\sum_{i=0}^{d} \sum_{j=0}^{d} (n_c{}^2(1 + 2 \times i)(1 + 2 \times j))$ time units

$$= {n_c}^2 \sum_{i=0}^{d}(1 + 2 \times i) \sum_{j=0}^{d}(1 + 2 \times j)$$

$$= {n_c}^2 (1 + d)^2 (1 + d)^2$$

$$\therefore O(sim_{Enclosure}) = O({n_c}^2 \times d^4)$$

3.7.2 Complexity of RSS relatedness

Suppose, I_1 and I_2 are two items (elements), n_{e_1} and n_{e_2} are the number of sub-elements, t_1 and t_2 are the corresponding content of sub-elements, n and m represents the number of concept sets in the vector spaces of $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$. Item relatedness is computed in a polynomial time complexity of $O(n \times m \times {n_c}^2 \times d^4)$ since:

- Text relatedness (*TR*) is computed with time complexity dependent on complexity of:
  - (i) building the vector space – that depends on the size of the Knowledge Base and the number of *concept sets* and complexity of computing enclosure similarity, i.e., $O(n \times m \times {n_c}^2 (1 + d)^4)$
  - (ii) detecting the relationship is done in $O(n + m)$.

  Hence, $O(TR) = O(n \times m \times {n_c}^2 \times d^4)$ as the complexity of detecting relationship don't contribute the final complexity.
- The complexity of simple element relatedness is dependent mainly on *O*(*TR*).
- The complexity of item relatedness is dependent on the number of sub-elements that would be used to compute the relatedness value (in the worst case all sub-elements of an item is used as item connector) and simple element relatedness, i.e.,

  $$O(IR) = O\left(n_{e_1} \times n_{e_2} \times O(ER)\right)$$

  $$= O\left(n_{e_1} \times n_{e_2} \times n \times m \times {n_c}^2 \times d^4\right)$$

  As the number of sub-elements in each item I_1 and I_2 is constant, and the highest-order term, $n \times m \times {n_c}^2 \times d^4$, dominates the growth rate.

$$O(IR) \leq O\ (n \times m\ \times {n_c}^2 \times d^4)$$

Therefore, we conclude that IR is computed with complexity of $O(n \times m\ \times {n_c}^2 \times d^4)$, that depends on the number of concept sets in each texts, (n and m) and Knowledge Base information (depth d and number of concepts n_c).

3.8 Summary

In this chapter, we have presented our semantic relatedness measures (algorithms) dedicated mainly to RSS news items. In developing our relatedness measure, we followed bottom-up design strategy, which is summarized in Figure 3.7. The approach starts with measuring the similarity between concepts or keywords extracted from texts (i.e, labels and content of simple elements).

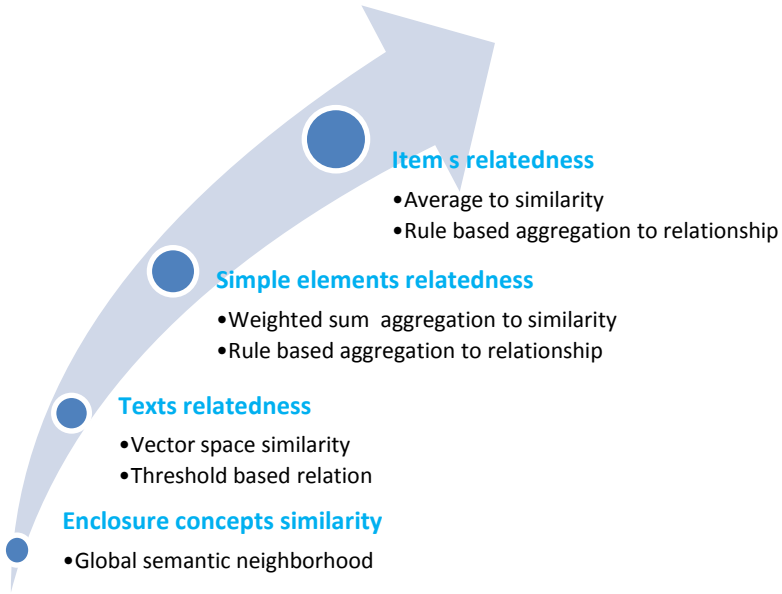


Figure 3.7: Summary of our semantic measures

The text relatedness algorithm identifies the semantic similarity value computing the angle separating the vectors containing their concept weight computed with enclosure measure. The relationship between texts is identified using intervals delimited by two

threshold values, disjointness and equality, and the content of the vector. The simple element and item relatedness algorithms combine the similarity and relationship value of their corresponding building block, using weighted sum or average for similarity and hard-coded rule for relationships.

The contribution of this chapter can be categorized as follows:

1. We proposed a generic concept-based similarity measure. Our measure identifies the semantic similarity value and relationship that exists between a pair of concepts, using the global semantic neighborhood of each concept. The proposed measure can be easily customized and configured by specifying the set of semantic relations and/or the threshold between concepts while identifying the semantic neighborhood and global semantic neighborhood. In addition, the applicability of this measure is not restricted only to RSS.
2. We provided three semantic relatedness algorithms (TR, ER, and IR) that work at different level of granularity-text, simple elements and item following the bottom-up design principle. Each of the algorithms returns a pair containing semantic similarity and relation values.
3. The provided measures are easily configurable and customizable by users. The similarity measures use extensively Knowledge Base that can be tuned and adapted by the application domain. In addition, the measures allow users to specify their notion of similarity by specifying parameter such as tuning threshold, and item connectors.
4. Our semantic relatedness measures run in polynomial time complexity that depends on the syntactic information (i.e., the number of concepts) and semantic information (i.e., the maximum number of words per synset, and the depth of the Knowledge Base).

5. We published extract of this chapter in international conferences (GETAHUN, F. et al., 2009; GETAHUN, F. et al., 2009) and WWW journal (GETAHUN, F. et al., 2009).

CHAPTER 4
CONTEXT-AWARE NEWS FEEDS MERGING FRAMEWORK

Abstract

The purpose of this chapter is to present the design of our context-aware RSS feeds merging framework. The framework allows a user to fuse/integrate RSS news items collected from a set of user's favorite sources using easily configurable merging rules. Here, we represent the user context and preferences as a Knowledge Base, and merging rules using Horn clause in First Order Logic. We categorize our merging rules into two: simple elements and items merging rules. In this chapter, we also present our adaption of the link hierarchical clustering algorithm used to facilitate the merging process.

4.1 Introduction

As discussed in Chapter 1, semantic- and context -aware merging of RSS feeds have been considered as a fundamental requirement for an integrated feed aggregation in distributed and heterogeneous environment. Up to now, four approaches have been proposed to merge semi-structured/XML documents: template-based (TUFTE, K. and Maier, D., 2001; TUFTE, K. and Maier, D., 2002; WEI, W. et al., 2004; LAU, H. and Ng, W., 2005), 2-ways (FONTAINE, R.L., 2002) and 3-ways merging (LINDHOLM, T., 2003; LINDHOLM, T., 2004), and propositional fusion-rule (HUNTER, A. and Liu, W., 2006; HUNTER, A. and Liu, W., 2006). Each of these approaches promotes the use of hard-coded merging rules. The merging rules decide on what to do when a particular condition is satisfied (such as a node is inserted, deleted, moved, or updated). Unfortunately, in these approaches, the actions are restricted to join the sub-documents with inner or outer join type and aren't flexible.

In this chapter, we present a *context-aware* and *rule-based* RSS merging approach that empowers a user in writing her perception of merging feeds by combining a set of pre-defined rules.

The rest of this chapter is organized as follows. In Section 4.2, we present the architecture of RSS merger. Section 4.3 details our relationship-aware adaptation of hierarchical clustering algorithm that assists the merging process. In Section 4.4, we present a Knowledge Based model to represent both user context and user preferences followed by a context-aware merging rule in Section 4.5. In Section 4.6, we detail our context-aware feed merger. Section 4.7 details the set of actions used to generate the output of feed merging and we conclude the chapter by providing the summary in Section 4.8.

4.2 News feed merging architecture

As presented in the Motivation Section of Chapter 1 (cf. Section 1.2), a user might use different kind of devices (e.g., PC, PDA, Smartphone, etc.) at different moments (morning, tea break, week-ends, etc.) within different locations (office, home, etc.) to read integrated news items extracted from her set of feed sources. Thus, providing integrated news to a user has to be adapted according to her context.

The motivation behind the merging of feeds is to provide a new way that integrates all feeds collected from distributed and heterogeneous sources. The merging process takes into consideration the device type, the context and the preferences of the user. The architecture of our RSS merging framework is shown in Figure 4.1. It is composed of:

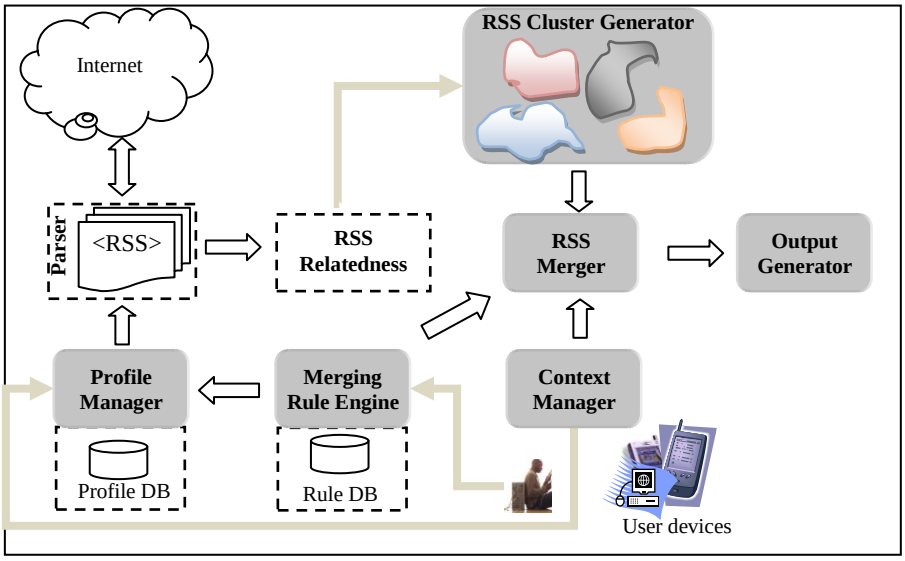


Figure 4.1: Architecture of RSS merging framework

- *RSS Cluster Generator*: it is an optional component used to put together related news items so as to facilitate and improve the merging quality. This component uses the result of our *RSS relatedness* measure detailed in Chapter 3 together with our adaptation of link clustering algorithm. This component is detailed in Section 4.3.

- *Context Manager*: it captures the context of the user interacting with the system. A context (DEY, A. K., 2001) refers to any information that characterizes the situation of an entity (user) such as the whereabouts information -the location of the user-, the resource used to interact with the system -device information-, timing, and the most frequently accessed information. The context manager automatically captures this information and communicates with the *Profile Manager* to store it.
- *Profile Manager*: it handles the management of user's profile. A profile refers to a user information, her contextual information identified with the help of context manager and her set of preferences.
- *Merging Rule Engine*: it is responsible for managing the different merging rules proposed by the user. A user provides a set of merging rules at least once (for instance, when the system is initialized for the first time) and can modify them later whenever necessary. This component is detailed in Section 4.4 and 4.5. These rules are part of the user profile and hence handled by the *Profile Manager*.
- *RSS Merger*: it aggregates/fuses news items, within the same cluster, using the set of merging rules provided by the user and fits with the current user context. This component is detailed in Section 4.6.
- *Output Generator*: it accepts the result of the *RSS Merger* component and generates a result in the format requested by the user (for instance, RSS, Atom, XHTML, etc.). It is composed of a set of functions or actions that decide the order in which an output would be generated. It is detailed in Section 4.7.

4.3 Clustering

Clustering is a method for grouping similar data together. In our framework, it is a pre-processing step that facilitates the merging process. Please recall that, the different clustering approaches are divided into two broad categories: Hierarchical and Non-

hierarchical.

1) Hierarchical (agglomerative and divisive) (GOWER, J. C. and Ross, G. J. S., 1969; JARDINE, N. and Sibson, R., 1971) clustering algorithms produce nested sets of data (hierarchies), in which pairs of elements or clusters are successively linked until every element in the data set becomes connected. Single link (SNEATH, P. H. A and Sokal, R.R., 1973), complete link (KING, B., 1967) and group average link (ALDENDEFER, M. S. and Blashfield, R. K., 1984) algorithms are the known hierarchical clustering methods.
2) Non-hierarchical methods group a data set into a number of clusters irrespective of the route by which they are obtained (e.g., K-means (HARTIGAN, J. A. and Wong, M. A., 1979)).

Independent of the clustering categories, a clustering algorithm group only highly related documents/items. Hence, applying such algorithms in our feed context would result in grouping mainly highly overlapping news in the same cluster as they disregard relationships. In other words, those news items related with the inclusion relationship for instance, and having lesser relatedness/similarity scores, would be put in different clusters. However, such items should naturally belong to the same cluster so as to be subsequently merged together.

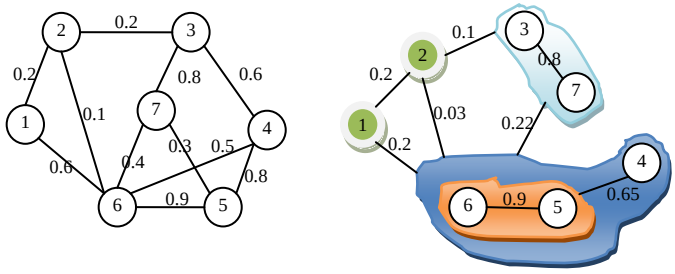


Figure 4.2: Group-average link clustering at level 0.6

To avoid this, we have adapted in this work, the graph-based agglomerative group

average-link clustering method to consider RSS item relationships. Given n RSS news items, we form a fully connected graph G with n vertices and $n(n-1)/2$ weighted edges. The vertices represent the news items/clusters, and the weight of an edge corresponds to the item semantic relatedness/similarity value between vertices connected with this edge. The group link clusters for a clustering level l_i (i.e., c_{l_i}) can be identified by combining those vertices with weights $w \geq l_i$ from the graph G. The clustering level is a threshold value used to combine clusters including single news clusters.

Figure 4.2.A represents a graph with seven nodes that correspond to single news clusters: the number in the circle represents the *id* of member news item and the weight corresponds to the semantic relatedness value between the news items. The missing edges have a semantic relatedness value of 0. Figure 4.2.B presents the remaining graph after deleting all edges with weight < 0.6 (i.e., combining those vertices with weight ≥ 0.6). There are four clusters C_1={1}, C_2={2}, C_3={3, 7}, C_4={6, 5, 4} representing clustering at level 0.6. In general, the resulting of clustering at clustering level l_i, C_{l_i}, contains all news items I with semantic relatedness value greater than or equal to l_i. Formally:

$$C_{l_i} = \{I \mid \exists I_j \in C_{l_i}, SemRel(I, I_j) \geq l_i\} \quad (4.1)$$

where, *SemRel* returns the semantic relatedness between two items.

The Pseudo Code 4 represents our relationship-aware group average link level based clustering algorithm called *RaGALL*. It groups together all nodes with higher similarity value and also those related with inclusion relationships. The edge connecting a pair of clusters C_i and C_j represents an average relatedness/similarity and is computed using Unweighted Pair Grouping Method (UPGM) (SNEATH, P. H. A and Sokal, R.R., 1973):

$$AvgSemRel(C_i, C_j) = \frac{\sum_{k=1}^{|C_i|} \sum_{l=1}^{|C_j|} SemRel(I_k^{C_i}, I_l^{C_j})}{|C_i| \times |C_j|} \quad (4.2)$$

where:

- $I_k^{C_i}$ and $I_l^{C_j}$ represent the k^{th} and l^{th} member news item of clusters C_i and C_j respectively
- $|C_i|$ and $|C_j|$ represent the size of the cluster C_i and C_j respectively
- *SemRel* returns semantic relatedness value between the two items.

Example 4.1: For instance, in Figure 4.2.B, the weight of the edge connecting cluster C_3 = {3, 7} and C_2 = {6,5,4}, is computed as:

$$AvgSemRel(C_3, C_4) = \frac{SemRel(3,4) + SemRel(7,6) + SemRel(7,5)}{2 \times 3}$$

$$AvgSemRel(C_3, C_4) = \frac{0.6 + 0.4 + 0.3}{6} = 0.22$$

In the same way, the weight of all the remaining edges connecting pair of clusters is computed.

The result of *RaGALL* clustering C_{l_i} is similar to the α cut clustering result of Gracia and Ng in (GARCIA, I. and Ng, Y., 2006). In (GARCIA, I. and Ng, Y., 2006), a news item may belong to different clusters, and a cluster contains a set of related articles. The redundant (identical and subsume) and less-informative articles are removed with the help of a fuzzy equivalence relation. However, our algorithm generates independent clusters (i.e., a pair of news items from two different clusters is related only with a *disjoint* relationship).

The algorithm *RaGALL* generates clusters by varying the clustering level between 1 and 0, at a constant decrement pace of *Dec-value*. Lines 7 and 8 show clustering at level 1 which generates the initial clusters for each individual news items and groups those items

that are related with equality and/or inclusion relationships. It results in grouping redundant news items. Lines 11 to 15 show clustering at level l_i which involve two steps: firstly, computing the semantic relatedness between the two clusters using UPGM; and secondly, grouping the clusters if their corresponding weight is greater than or equal to l_i.

Pseudo Code 4: RaGALL Algorithm

```
    Input:
1.    Sem_Rel[][]: Decimal   //a matrix containing semantic relatedness value of pair of items
    Variable:
2.    Dec-value: Decimal            // constant clustering level decrement value (e.g., -0.1)
3.    li: Decimal                   // clustering level
4.    cl: Decimal                   // stopping clustering level
    Output:
5.    Clusters: Collection          // contain the result of clustering
    Begin
6.    For li = 1 Down to 0 Step Dec-value
7.      If li = 1 Then
8.        Clusters=Generate_Initial_Clusters_Grouping_Redundancy(Sem_Rel)
9.      Else
10.       For each pair of clusters (ci, cj) in Clusters
                                    //Clusters contains group of items at level li-1
11.         Average-Relatedness = UPGM(ci, cj)  //computed using Equation (4.2)
12.         If Average-Relatedness ≥ li Then
13.           group ci and cj in the same cluster
14.         End If
15.       Next
16.     End if
17.   Next
18.   cl =C-Index(Clusters)     // stopping rule for clustering
19.   Return clusters[cl]
    End
```

A stopping rule is necessary to determine the most appropriate clustering level for the link hierarchies. Milligan & Cooper (MILLIGAN, G. W. and Cooper, M. C., 1985) present 30 of such rules. Among these rules, C-index (HUBERT, L.J. and Levin, J.R., 1976) exhibits excellent performance (found in the top 3 stopping rules). Here, in line 19,

we use an adaptation of C-index, provided by Dalamagas *et al.* (DALAMAGAS, T. et al., 2006) and detailed in Annex 3.

4.4 User context modeling

We recall that a context is any information that describes the situation of an entity. It might be extracted automatically using widget (DEY, A. K. et al., 2001; EJIGU, D. et al., 2008) or manually. An entity refers to a person, an object, a device, etc. that interacts with the system and has a set of attributes that describe it.

A user preference refers to what a user likes or dislikes and her favorite set of RSS feed addresses. This turns out to be the association between the user and other entities.

In our study, the captured context information and user preferences are stored as part of the user's profile for later use. Here, we represent both following CONtext ONtology – CONON (WANG, X. H. et al., 2004) and EHRAM (EJIGU, D. et al., 2008) with a user context Knowledge Base. Formally, the user context Knowledge Base *CKB* is represented as a collection of related concepts (entities) and denoted as:

$$CKB = \langle C, E, R, f \rangle \tag{4.3}$$

where:

- C is a collection of concepts. A concept represents an entity or instance of an entity. Each entity has a uniform resource identifier that can uniquely identify and relate it with other entities.
- E is an edge that connects two related concepts i.e., $E \in C \times C$.
- R is the set of relationships associated to an edge i.e., $R = \{isa,\ ismemberof,\ uses, in, at, instanceof, like, dislike\}$
- f is a function denoting the nature of the edge i.e., $f: E \rightarrow R$ to associate an edge with a relationship.

Figure 4.3 shows a sample multi-rooted user context Knowledge Base. The root *usercontext* references a user together with her context such device, and location information, whereas the other root *user preference* refers to a user and her preferences restricted to favorite news sources.

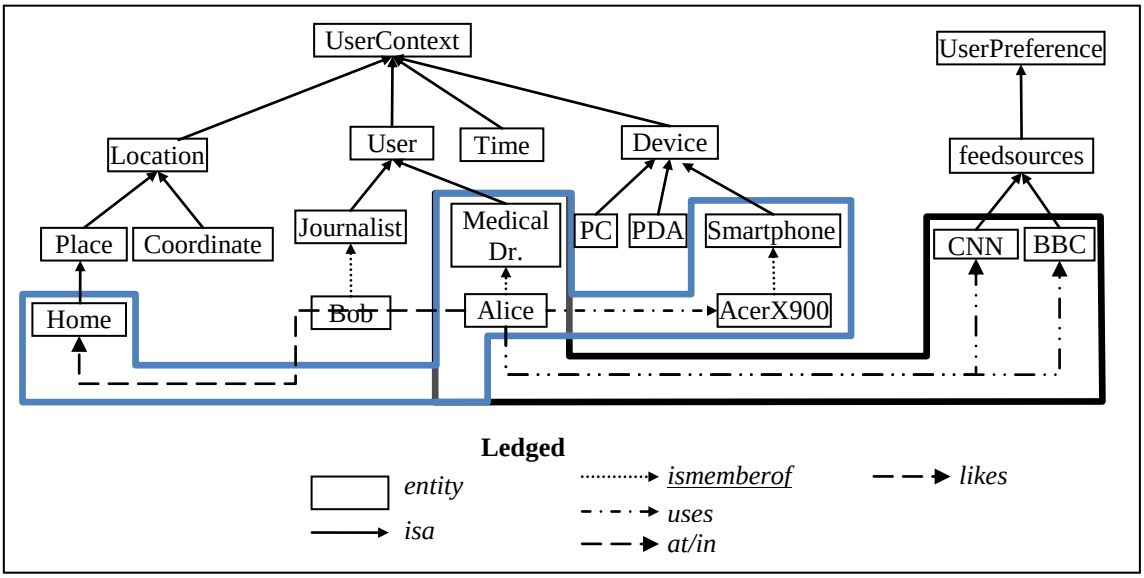


Figure 4.3: A sample user context Knowledge Base

The relationship between concepts is interpreted as a predicate or logic clause accepting the concepts as parameters as demonstrated in Example 4.2 and Example 4.3. Table 4.2 contains some of the functions related to user context and preference modeling.

Example 4.2: For instance, Alice is a medical doctor that uses Acer X900, Smartphone, at the time she accesses the RSS merger. This contextual statement is represented in the Knowledge Base shown in Figure 4.3 using the entities and their relationship. We represent this contextual information with conjunction of atoms/propositions in FOL format that uses relationship as predicate and concepts/entities as parameters i.e.,

$$ismemberof(Medical\ Dr., Alice) \wedge uses(AcerX900, Alice) \wedge$$
$$ismemberof(SmartPhone, AcerX900) \wedge at(Home, Alice).$$

Notice that, a member of a class (or the instance of a class) has all the attributes and operations of the class it is created from and may transitively inherit the behavior of its super-classes or ancestors provided that there is a defined relationship between the child and the ancestors (c.f. Table 4.1 for the transitivity relationships).

For instance, Alice *ismemberof* Medical Doctor and A Medical Doctor *isa* User. Hence, transitively, Alice *isa* user.

Table 4.1: Intra transitivity relationship between entities

R_{ij} \ R_{jk}	*uses*	*at/in*	*ismemeberof*	*isa*
uses	uses			uses
at/in		at/in	at/in	at/in
ismemeberof	uses		ismemeberof	isa
isa	uses		isa	isa

In general, given three entities C_i, C_j and C_k, and two relationships R_{ij} and R_{jk} between C_i and C_j and C_j and C_k respectively, the transitive relationship is denoted as R_{ik}. Table 4.1 shows the transitive relationship between the entities which is not limited to the same relationships.

Table 4.2: Sample list of functions associated to context modeling

Function	Description
Boolean *feedsource*(sources S, feed F)	Returns True if the feed F is among the web feed sources S
Boolean *content*(feed F, string C)	Returns True if the feed F is about the string C
Boolean *r*(concept C_1, concept C_2)	Returns True if the $relation(C_1, C_2) = r$, $r \in \{isa, ismemeberof, uses, instanceof, like, dislike, \dots\}$
Feed[ ] *getsources*(string X)	Crawls the WWW extracts list of web feed addresses that describe issues about *X*
String gettimeperiod(user U, date D, time T)	Returns a string value that represents the context information time, by interpreting the date *d* and time *t*, particularly to the user *u*. It might refer to the user agenda, and a table of conditions and the associated interpretation. For instance, 10 AM might be interpreted as a coffee break for Alice.
String *gps2place*(coordinate GPS)	Convert the actual GPS coordinate into a text describing the place
Feed[ ] *getfeeds*(user U)	Returns the favorite feeds of the user *u*
Context *getcontext*( )	Returns context using a specialized widget installed in the device

Example 4.3: For instance, Alice has registered CNN and BBC as her favorite news feed sources. This is directly represented in the context Knowledge Base shown in Figure 4.3, with the edge connecting the Alice with feed sources CNN and BBC with like relationship.

Example 4.4: Alice wants to read only the sport news of CNN and BBC, focusing on "football" and "basketball". This preference can be represented as set of rules using the functions listed in Table 4.2. i.e.,

$$\forall F \in getfeeds(alice), \big(feedsource(cnn, F) \vee feedsource(bbc, F)\big) \wedge \big(content(F, "football") \vee content(F, "basketball")\big)$$

Notice that using the user contextual Knowledge Base shown in Figure 4.3, *getfeeds*(Alice) returns BBC and CNN.

The next section details merging rules.

4.5 Merging rule

A merging rule is an expression that determines when an action would be done. Here, the merging of news items depends on the context of the user and her personalized set of merge rules. We represent a merging rule using FOL (SMULLYAN, M. R., 1995) Horn clause (HORN, A., 1951). The Horn clause controls merging elements depending on the result of antecedent expression (i.e., a set of functions/predicates that access the user contextual information, the relationship or similarity between these elements). Given two terms $term_1$ and $term_2$ referring to either simple elements or items extracted from a feed, the merging rule is denoted as:

$$Rule ::= predicate\ (term_1, term_2)\ [Connective\ \ predicate(term_1, term_2)] \Rightarrow action(term_1, term_2) \quad (4.4)$$

where:

- $predicate$ is a Boolean function. It can be relationship, similarity or one of context manipulation functions shown in Table 4.2

- $Connective$ is the logical connector AND or OR
- $term_1$ and $term_2$ are either simple-elements or items
- *action* represents merging function that would be executed (cf. list of merging functions in Table 4.3) as soon as the predicate (antecedent) is True.

Example 4.5: For instance, for any two items extracted from the favorite source of the user *u* keep the first element if they are related with equality or inclusion relationship. This statement is represented with the following rule: $\forall X, Y \in getfeeds(u)$

$$ro ::= equal(X,Y) \vee includes(X,Y) \Rightarrow keepfirst(X,Y)$$

The predicate *equal(X,Y)* returns True only if *Y is equal to X*. Similarly, *includes*(*X,Y*) returns True only if *X includes Y*. The action function, *keepfirst,* keeps the first item i.e., *X*.

Table 4.3: List of merging actions

Merging function	Description
$item$ keeplatest ($item\ i_1, item\ i_2$)	returns the latest of the two news items
element keepfirst ($element\ e_1, element\ e_2$)	returns the first element
$element$ keepsecond ($element\ e_1, element\ e_2$)	returns the second element
item keepboth ($item\ i_1, item\ i_2$)	keep both items
getcorrespondence(itemi_1, $item\ i_2$)	returns the correspondence between pair of items
concat($element\ e_1$, element e_2, string c)	returns the concatenation of two elements delimited by the string c

Merging two items using our merging rule is similar to the propositional fusion rule of Hunter (HUNTER, A. and Liu, W., 2006). However, our merging process is dependent on the relationship/similarity existing between elements or items. Figure 4.4 shows the set of default merging rules (i.e., rules used only if a user didn't provide any personalized rules) categorized into simple elements and items merging rule.

On one hand, Figure 4.4.A contains the list of simple elements merging rules $Merge_{Simple}$. It makes sure that, given two simple elements e_1 and e_2, it produces another element, as detailed below.

- The *Rule-S1* returns the result of `keepfirst` which is the first element of those related with equality or inclusion relationship, as a merge result.
- The antecedent formula in *Rule-S2*, is always True (i.e., fact) and hence the function `concat`(*e*$_1$, *e*$_2$, *delimiter*) returns the result of concatenating the two elements separated by *delimiter*. However, such a rule is applied only if there is no other rule to be used. In this particular merging case, the *Rule-S2* is used when the elements are related with either *Overlap* or *Disjoint* relationships.

  The function `Concat` creates a new element e_k having the least common ancestor of e_1 and e_2 tag names, and the content can be build in two ways:

  1) by concatenating the common and different part of e_1 and e_2 contents separated by the *delimiter* i.e.,

     e_k::=<lca(e_1.name, e_2.name)> concat(getcommon(e_1.content,e_2.content) $getdifference(e_1.content, e_2.content), delimiter)$ </$lca(e_1.name, e_2.name)$>

  2) looking for the synopsis of the first www document (for instance using Google API) that contains all common and different concepts of the elements.

It is to be noted that the result of merging two simple elements might fail to consider two important issues:

(1) merging attributes, and
(2) handling the issue of merging an element without correspondences.

For instance, an element named *category* exists only in BBC3 news (cf. Table 4.4) and doesn't have any corresponding element in CNN4; as result the correspondence is *NULL*.

To handle the first issue, i.e., merging attributes, we consider three cases:

1) if the attributes have similar name and similar value, then keep only one of the attributes as a result;

2) if the attributes have similar name but different values then store the concatenation of their values separated by '|' to reflect the conflict;
3) otherwise, add each attribute to the merged element.

To handle the second issue, we return the known element as merged result.

On the other hand, Figure 4.4.B contains the list of items merging rules, as follows.

- *Rule-I1*, returns the latest of the equal news items as a final result. Notice that, each news item has a timestamp which is added either at the time of creation or transmission. The function `keeplatest` keeps the recent news items taking into consideration the difference in time zone and time format.
- *Rule-I2*, returns the first item as the result of merging news items related with the *include* relationship using the `keepfirst` action function.
- *Rule-I3* is used when the news items are overlapping. Merging such news items come down to merging recursively the corresponding contents (i.e., sub-elements). This is achieved in three steps:
    1) identify the correspondence matrix, containing matching sub-elements of each item, using `getcorrespondence` function. This function returns a set of elements; each member *e* has three members (accessed with an index of counting number type) referring to the name of sub-elements of each item and the relationship in-between. Table 4.4 shows a sample result of the `getcorrespondence` operator.
    2) merge each matching pair of correspondence matrix using the simple elements merging rule, $Merge_{Simple}$ (shown in Figure 4.4.A or personalized by the user), and accumulate the result until all the elements are merged.
    3) add the accumulated elements as the children of a new item and return it as the final merged result.

- The *Rule-I4* is used to merge disjoint news items. It comes down to keeping both items with `keepboth` function. The function `keepboth` generalizes Lau & Ng (Lau & Ng, 2007) notion of merging disjoint elements by creating a pre-defined element, *m*, having both elements as children or returning an item that aggregates the result of concatenating corresponding sub-elements of both items. It is formalized as follows:

$$KeepBoth(I_1, I_2, deepconcat, delimiter, m) ::=$$

$$\begin{cases} \text{<item>} \left(\sum_{e \in getCorrespondence(I_1, I_2)} concat(e[1], e[2], delimiter) \right) \text{</item>}, deepconcat=True \\ \text{<m>} concat(I_1, I_2, delimiter) \text{</m>} \qquad otherwise \end{cases} \tag{4.5}$$

$\forall e_1, e_2 \in Simple - Element$

$Rule\text{-}S1 ::= equal(e_1, e_2) \vee include(e_1, e_2) \Rightarrow keepfirst(e_1, e_2)$

$Rule\text{-}S2 ::= True \Rightarrow concat(e_1, e_2, delimiter)$

A. Simple element merging rule: $Merge_{Simple}$

$\forall I_1, I_2 \in Item$

$Rule\text{-}I1 ::= equal(I_1, I_2) \Rightarrow keeplatest(I_1, I_2)$

$Rule\text{-}I2 ::= include(I_1, I_2) \Rightarrow keepfirst(I_1, I_2)$

$Rule\text{-}I3 ::= overlap(I_1, I_2)$

$$\Rightarrow \sum_{\forall e \in getCorrespondences(I_1, I_2)} Merge_{Simple}(e[1], e[2])$$

$Rule - I4 ::= disjoint(I_1, I_2) \Rightarrow keepboth(I_1, I_2, deepconcat, delimiter, m)$

B. Item merging rule : $Merge_{Item}$

Figure 4.4: Some of the default merging rules

A merging rule is context-aware, if the antecedent formula in Equation (4.4) uses the context of a user and the consequence is a *merging rule*. It is denoted as follows:

$$\text{uc} \Rightarrow \text{merging} - \text{rule} \tag{4.6}$$

where:

- *uc* is the user context (c.f. Section 4.4). It is a formula in FOL
- *merging-rule* is expressed in Equation (4.4). Triggered only if *uc* is True.

Example 4.6 The following is an example of context-aware merging rule of Alice used only when she is in her office using a PDA.

$\forall I_1, I_2 \in Item, u \in User$

$getContext(U) = \text{«}\, office \,\text{»} \wedge Device(U) = \text{«}\, pda \,\text{»}$

$$\Rightarrow \begin{cases} Equal\ (I_1, I_2) & \Rightarrow & keeplatest(I_1, I_2) \\ Include(I_1, I_2) & \Rightarrow & keepfirst(I_1, I_2) \\ Overlap(I_1, I_2) & \Rightarrow & \sum_{\forall e \in getCorrespondences(I_1, I_2)} Merge_{Simple}(e[1], e[2]) \\ Disjoint\ (I_1, I_2) & \Rightarrow & keepboth(I_1, I_2, deepconcat, delimiter, m) \end{cases}$$

Recall that the RSS merger communicates with the *rule engine* to get a set of merging rules that satisfies the user context. The rule engine extracts the rules following four steps as shown in Pseudo Code 5 below.

Pseudo Code 5: Get_Merging_rule

Step 1. Get the context of the user using `getContext` (part of widget install on the device),

Step 2. Select the personalized merging rules of the user that satisfy the recent recommendation/preference defined on contextual information.

Step 3. If there is any

- identify the personalized merging rules stored in the user profile satisfying the recommendation identified in Step 2

Step 4. Otherwise,

- the rule engine infers the most probable rules using the historical contextual recommendation (e.g., returning the popular recommendations or returns the default merge rules).

Example 4.7: Referring to the Scenario 2, assume Alice is at home using her PC and wants to read the different perspective of each news author. Using Pseudo Code 5, this preference is interpreted as merging the news items using the default merging rules as Alice didn't present her personalized rule.

Let us consider the RSS news items relatedness between *CNN4* and *BBC3* detailed in Example 3.14. These items are related with *overlap* relationship using *title* and *description* as item connector.

Referring to the items merging rule, $Merge_{Item}$, *Rule-I3* is the best rule that applies to merge these news items. Merging these items come down to the merging of their corresponding sub-elements. The correspondence between sub-elements (i.e., the result of `getCorrespondence` operator) is shown in Table 4.4. Notice that, the operator `getCorrespondence` identifies the best correspondence using the maximum relatedness value. Otherwise, tag name similarity is used. A `Null` valued relationship signifies either the elements are not part of item connector or the element exists only in one item. Merging *CNN4* and *BBC3* is represented as:

$Merge_{Items}(CNN4, \quad BBC3)$ ::=< *item* >

$$\left(\sum_{\forall e \in getCorrespondences(CNN4,BBC3)} Merge_{Simple}(e[1], e[2])\right) \text{</item>}$$

$Merge_{Items}(CNN4,\ BBC3)$ ::=< *item* >

$Merge_{simple}(title_{CNN4},\ title_{BBC3})\ Merge_{simple}(description_{CNN4},$
$description_{BBC3}) Merge_{simple}(link_{CNN4},\ link_{BBC3})\ Merge_{simple}(guid_{CNN4},$
$guid_{BBC3})\ Merge_{simple}(null,\ category_{BBC3})$</item>

The result is shown in Figure 4.5.

Table 4.4: Correspondence between CNN4 and BBC3: getCorrespondence(CNN4, BBC3)

e_i	e_j	*Relation*
$title_{CNN4}$	$title_{BBC3}$	*Equal*
$description_{CNN4}$	$description_{BBC3}$	*Overlap*
$link_{CNN4}$	$link_{BBC3}$	*Null*
$guid_{CNN4}$	$guid_{BBC3}$	*Null*
Null	$Category_{BBC3}$	*Null*

4.6 RSS merger

Merging RSS news items collected from one or more sources can be done after grouping items using our relationship-aware clustering algorithm - *RaGALL*. Recall that merging could be done without performing clustering, in such a case there is only one cluster that contains all news items. Nonetheless, clustering would provide more relevant merging candidates, and thus would amend merging results (cf. Section 4.3).

Hereunder, we start by defining an item neighborhood to be exploited in applying the merging rules, and performing RSS news items merging.

Definition 4.1. [Item Neighborhood]

The neighborhood of news item I_i refers to a set of news items I_j related with *equality* or *inclusion* relationship. Formally, it is denoted as:

$$N(I_i) = \{I_j | I_i = I_j \vee I_i \supset I_j\} \quad (4.7)$$

Once the neighborhood is identified, all items in $N(I_i)$ can be collapsed and represented by I_i without losing information.

Example 4.8: For instance, $N(CNN2) = \{I_j | CNN2 = I_j \vee CNN2 \supset I_j\}$ returns all news related with equality or inclusion with CNN2.

Considering the sample news feeds in Figure 1.1 and Figure 1.3, the neighborhood of CNN2 returns BBC2 i.e., $N(CNN2) = \{BBC2\}$. Notice that *CNN2* is considered as the representative or centroid of the resulting set.

Here, we provide an algorithm represented as a Pseudo Code 6 that handles merging of news items collected from a set of distributed sources. The algorithm accepts a cluster of news items with the accompanying semantic relatedness matrix i.e., *sem_rel* and generates a merged version. For any pair of news items *i* and *j*, *sem_rel*[*i*][*j*].value and *sem_rel*[*i*][*j*].*rel* represent respectively the relatedness and the relationship components of the item relatedness measure. In addition, the algorithm accepts user information such as her/his personal identifier.

The RSS merger communicates with the rule engine presented above in Pseudo Code 5 (Line 7) to extract the set of merging rules associated to the user *U*. In Line 8, an empty document is created using the initialize action. Then, in Line 10, the item neighborhood of a news item is identified so as to produce a special item, I_r, which can represent the merged result of all news belonging to the same item neighborhood using *Merge-Items-*

Neighborhood[42]. Then in Line 11, the semantic relatedness matrix is updated by deleting the rows and columns of all items included in the neighborhood of I_i and add I_r into the output file. Lines 15 to 23 are used to merge all the remaining news items. The merging process is conducted incrementally.

In Line 16, any two highly related news items (I_s and I_r) over all pair of items are identified. These news items are merged using the merging rule provided by the user (Line 17). The resulting news item is added to the output file (Line 18). In Line 19, the *sem_rel* matrix is updated by removing rows and columns of I_s and I_r by adding the newly merged news item I_k. Item relatedness between I_k and those related with its constituting components (i.e., I_s and I_r) is computed by aggregating the relatedness between sub-elements of I_s and I_r. The semantic relatedness between sub-elements E_{ik} of I_i and I_k is computed as the average semantic similarity value of $SemRel(E_{ik},E_{sj})$ and $SemRel(E_{ik},E_{rj})$ where E_{sj} and E_{rj} are sub-elements of I_s and I_r respectively. The relation between sub-elements is identified using the semantic relatedness value and two threshold values, $T_{Disjointness}$ and T_{Equal}, as shown in Line 21. In Line 22, the semantic relatedness and relationship between items are computed by combining the semantic relatedness and relationship values using the Item relatedness algorithm (c.f. Pseudo Code 3).

[42] Merge-Items-Neighborhood merges news items redundant news items based on the equality and inclusion merging rule of the user.

Pseudo Code 6: Merging RSS news Items

```
Input:
1.      Ci: {I1, I2, …, Im}                    // Ci is a cluster having Ik items  1 ≤ k ≤ m
2.      sem_rel[][]                            // it contains items relatedness value
3.      U : User                               // user information
Variable:
4.      User-merging-rule[]: String            // list of action to be done based on the relationship
5.      r, s: Integer
OutPut:
6.      Doc: Document                          // file containing merged news items
Begin
7.      User-merging-rule = Pseudo Code 5(Getcontext(U)) //get merging rule of u
8.      Doc = Initalize ("RSS")
9.      For each Ii in Ci
10.        N = GetNeighborhood (Ii )           //cf. Def. 4.6.
11.        Update(sem_rel)                     //deleting news included in neighborhood of Ii
12.        Ir = Merge-Items-Neighborhood (N, user-merging-rule)
13.        AddElement (Ir, Doc.DocumentRoot)
14.     Next
15.     Do
16.        (r, s) = max_{i=1..n, j=1..n} (sem_rel[i][j]. value)
           //Find the most similar pair of news items say r and s over all items
17.        Ik = Merge_Item (Ir, Is, user-merging-rule)   //Merge r and s to form a new item Ik.
18.        AddElement (Ik , Doc.DocumentRoot)
19.        I = Update(sem_rel) // by deleting one of the merged elts and returns the position
           the other
20.        SemRel(EikE(s,r)j)= Avg (SemRel(EikEsj), SemRel(EikErj))
21.        Relation(EikE(s,r)j)=Relation(Semrel(EikE(s,r)j),Tdisjointness, TEqual)
22.        sem_rel[I][(r,s)] = IR(I, (r,s))
23.     Until all items are merged
24.     Return RSS
End
```

4.7 Output generation

According to Hunter and Summerton (HUNTER, A. and Summerton, R., 2003), an action determines the order and pattern in which an expression would be executed. In this work, we used action to build the resulting output documents in addition to generate merged version. It includes each of the following expressions.

1. `Document` *`Initialize`*(`String` *OutType*): it creates and returns an empty document of *OutType* which could be RSS, XML, XHTML, etc/.
2. `Void` *`AddElement`*(`Element` *nw*, `Element` *Parent*): it adds the element *nw* as child of *Parent*.

Notice that, in building a valid document, *`Initialize`* action should be executed before *`AddElement`*. In addition, there should be only one *`Initialize`* action.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<RSS version="2.0">
<Channel>
  <item>
    <title>U.N. chief launches $600M Gaza aid appeal</title>
    <description> United Nations Secretary-General Ban Ki-moon launch appeal aid people Gaza Israel military offensive | $613m affected offensive, body's top official says provide emergency humanitarian aftermath </description>
    <m>
      <link>http://edition.cnn.com/2008/WORLD/americas/05/01/gitmo.journalist/index.html?eref=edition
      </link><link>http://edition.cnn.com/2008/WORLD/americas/05/01/gitmo.journalist/index.html?eref=
      edition</link>
    </m>
    <m>
      <guid>http://edition.cnn.com/2008/WORLD/americas/05/01/gitmo.journalist/index.html?eref=edition
      </guid><guid isPermaLink="false">http://news.bbc.co.uk/go/rss/-/2/hi/me/723378828.stm</guid>
    </m>
    <category>Middle-east</category>
  </item>
</Channel>
</RSS>
```

Figure 4.5: Result of $merge_{item}$(CNN4 and BBC3) as RSS feed

Example 4.9: Considering Example 4.7 above, the following action list generates an RSS document having the merged items.

`Document` Doc = `Initalize`(“RSS”) //Create a document of given type OutType – default RSS

AddElement(*Merge*$_{Item}$(*CNN4*, *BBC3*), Doc.DocumentRoot)

// DocumentRoot identify the root of the document

4.8 Summary

In this chapter, we have presented a context-aware RSS feeds merging approach. The approach is rich and novel as it combines Knowledge Base, user-context and user-based rules. This approach benefits a user to provide a set of personalized rules on when and how to merge the content of her favorite providers.

The main challenges associated to it are summarized into three:

1) identifying and modeling user context information
2) allowing personalization and preference at different levels such as feed sources, recommendations, contexts or situations
3) providing flexibility in writing and rewriting rules that guides the merging of the news items.

One of the popular approaches to extract a user context is using a specialized widget that read the device header profile. The widget extracts location (e.g., GPS location), user status (e.g., busy, idle, online.), and timing. We model the user context as Knowledge Base, a collection of concepts (i.e. entities and instance of entities) related with relationship (such as *isa*, *ismemeberof*, *uses*, *in*, *at*). We used a rule-based approach to represent *merging rules* which are categorized into *simple elements* and *items* merging rule based on the type of elements to be merged. A *rule* is represented as FOL Horn clause having antecedents and a consequent.

The contribution of this chapter can be summarized as follows:

1. We adopt the agglomerative group average link clustering algorithm to be relationship-aware – *RaGALL*. The existing clustering algorithms (categorized into hierarchal and non-hierarchal) group together mainly highly similar and highly

overlapping documents/news and clusters. Disregarding relationship, for instance, those related with inclusion relationship and having lowest similarity value, would lead to the existence of false negative clusters as related items would be placed in different clusters. This affects both the clustering quality and consequently the merging result. In fact, our approach and idea can be applied to any of the existing clustering algorithms.

2. We proposed a context-aware and rule-based merging framework targeting mainly RSS news items. The proposed framework uses interacting components:
    a. to extract contextual information such as location, time, user profile, and device information, modeled as Knowledge Base, and stored permanently, and
    b. to extract the set of merging rule that fits to context of the user using the rule engine component
3. We proposed a flexible rule-based approach that empowers any user in providing specific notion of merging news items.

We published extract of this chapter in an international conference (GETAHUN, F. and Chbeir, R., 2010) and WWW journal (GETAHUN, F. et al., 2009).

CHAPTER 5
SEMANTIC-AWARE RSS ALGEBRA

Abstract

In this chapter, we study and provide RSS query algebra based on the notion of semantic similarity over dynamic content. The operators are supported with a set of similarity-based helper functions. We categorized the RSS operators into extraction, set membership and merge operators. We showed that the merge operator generalizes the join and the set membership operators. We also provided a set of query rewriting and equivalence rules that would be used during query simplification and optimization.

5.1 Introduction

The issue of algebra is crucial in different disciplines. In mathematics, algebra is a central point, it is defined as a pair (S, θ), where S is a set of values defined either in finite or possibly infinite domain space, and θ is the set of operators closure with the S (i.e., applying an operator over members of S provides another value in S). This definition works also in the context of database; S refers to the data model, and θ is the set of algebraic operators. The operators in the database model satisfy a minimal set of mathematical rules that allow the query optimizer component of the DBMS to rewrite a query into its equivalent forms.

A data model in database is fundamentally important as it describes how data are represented and accessed. Hence, it determines the structure of the data. In traditional database three data models -relational, object oriented and object relational models- are known. The relational model is very popular and highly researched. It represents data as a collection of related relations; each relation stores collection of related tuples. A tuple stores a set of scalar values drawn from the corresponding domain of each column defining the relation. The OO model represents directly objects, classes and hierarchal relationships. The object relational model combines the scalar nature of the relational with the object behavior of the OO model.

These three traditional database models have been used to represent and query data-centric XML documents. Here, query processing involves three steps:

1) mapping the XML documents into the basic components of the model – relations or objects
2) translate a user query into a query in the underlined data model, and execute the query (SQL, OQL)
3) reconstruct the result of the query to get a XML document.

However, the query predicates in such systems is restricted to either exact equality or inequality mainly on numerical values and *deep-equality* on hierarchically structured data (i.e., equality of the entire hierarchical structure inferred from the equality of each level) in case of object-oriented approach. These predicates types are also supported in the native XML query models and algebras.

Based on the numerous researches conducted on XML documents representation and retrieval (ZHANG, Z. et al., 2003; NIERMAN, A. and Jagadish, H. V., 2002; JAGADISH, H. V. et al., 2001; BEECH, D. et al., 1999), we used tree based data model to represent feed documents (cf. Section 3.2.1). In feed documents database, exact text equality is not enough. The examples in Section 1.2 (i.e., Example 1.5 to Example 1.8) demonstrated the need to have query algebra that takes into consideration the two specific behaviors of a feed – dynamism and semantic heterogeneity. In this chapter, we study and provide query algebra that base on the notion of semantic similarity over dynamic content.

The rest of this chapter is organized as follows. In Section 5.2, we provide the basic notions related to our algebra such as data-type and data stream. Then, in Section 5.2, we provide the minimal basic operators needed in semantic-based retrieval of RSS feeds that would be used in the future to extend XQuery. In Section 5.4, we study the properties of these operators and highlight the query optimization strategies. Finally, in Section 5.4.3, we conclude the chapter with summary information.

5.2 Preliminaries

In this section, we provide several notions used in the remaining part of the chapter. To begin, let us define data types which determine the input and output of the provided operators. The data-types are based on the feed data model (i.e. unordered collection of elements cf. Definition 3.1.). In this chapter, we adopt the Extended Backus-Naur Form (EBNF) with the help of symbols shown in Table 5.1.

Table 5.1: Notations used in the paper

Symbol	meaning
{ }	set of values
?	zero or 1 occurrence
+	One or more occurrence of
*	Zero or more occurrence of
[ ]	Array
|	separator-symbol

Definition 5.1. [Data Types]

In addition to the basic simple types such as Integer, Boolean, String, Char, Double, etc. (referred to here as *SimpleType*), our algebra uses the data types depicted in Figure 5.1 and discussed as follows:

- `Object`: it is the most generic data type and all other types inherit its behavior. The *`GetType`* operation returns the data type of an object.
- `Element`: it generalizes both the simple and complex element types. This type contains the basic information about an element such as `name`, `attributes` (collection of type `Attribute`), `value` (refers to the concatenation of the content of all children). The content of an element should be only simple, item or another element type. The `childElements` property contains the children of the element if the type of the element is item or complex element.
- `SimpleElement`: it is a specialized form of `Element` in which the content is a value of `SimpleType` (i.e., text, date, etc.) type. The static method `ER` returns the relatedness between two simple elements.
- `Item`: it refers to a complex element having set of objects of `SimpleElement` type as children. The static method `IR` returns the relatedness between two items. This type represents `item` or `entry` element of RSS and Atom feed respectively.

- `Element-linkage`: it stores the association/linkage between two simple elements together with the identified relationship.

- `Window`: it is a collection of elements defined on a given feed, having a type (i.e., count (BABCOCK, B. et al., 2002), sliding (BAI, Y. et al., 2006), or tumbling (KORN, F. et al., 2001)), and satisfies a boundary condition (i.e., the number of items – count, the start and end condition – sliding and tumbling types). The method *`ExtractWindow`* generates a window of the given type and having array/collection of elements satisfying the start and end conditions as content. A window might contain fixed/count number of recent elements (counting window type) or all data items within a given time- condition (in the case of sliding or tumbling widow type). In sliding window type, an element might belong to one or more windows depending on the starting condition of the window. However, in the tumbling window type, a new window starts only after the previous one is terminated. Hence, an element wouldn't belong to more than one window. Given an integer index i, the method *`GetElement`* returns the i^{th} member element of the window. ■

The class diagram in Figure 5.1 shows the hierarchal/inheritance and dependency relationships existing between different data-types (represented as classes). For example, the complex class `Item` is a kind of the general class `element` having zero or more objects of simple elements as content. Each element has a `name`, a `content/value` and zero or more `attributes`.

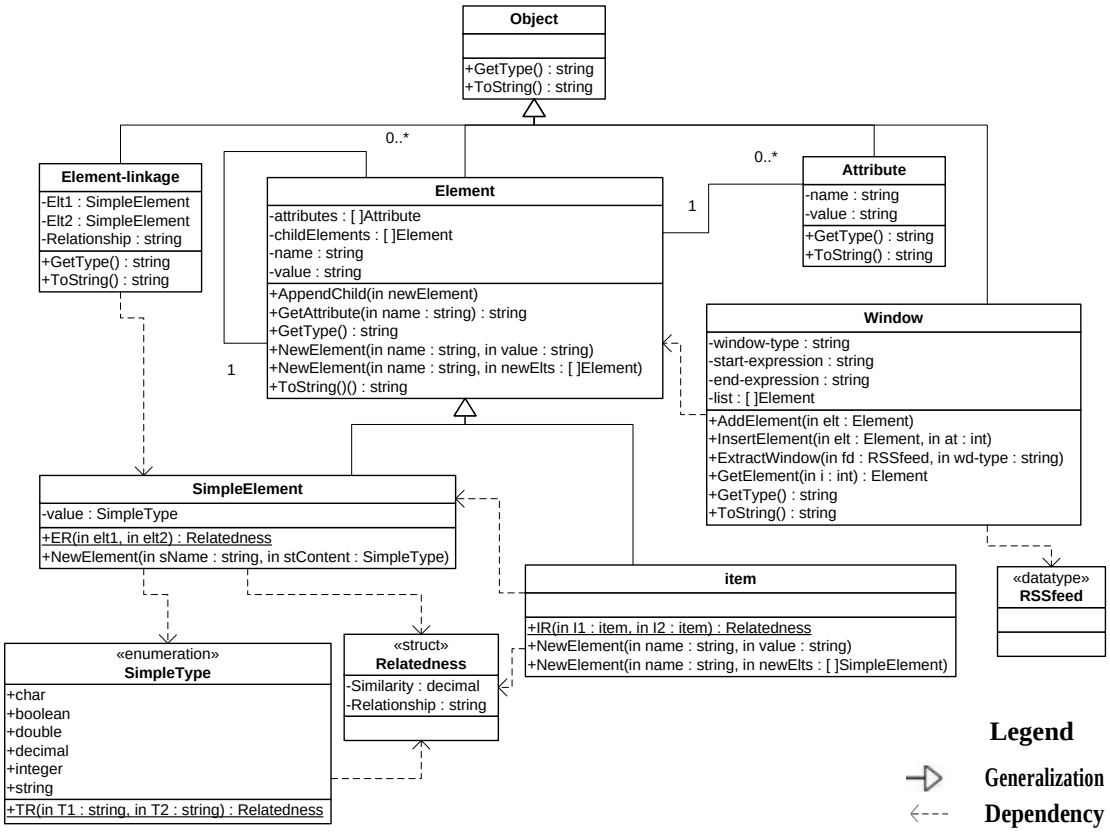


Figure 5.1: Data types used in our algebra

Definition 5.2. [Element Constructor]

The Element type supports the construction of new element using the constructor `NewElement` which is overloaded by both the `SimpleElement` and `Item` classes. The constructors create a new object having the behavior of the class. ■

In the class `Element`, `NewElement` accepts the element name, and/or a set of elements as content (i.e., children of the object to be created). The content of the element is restricted to simple element, item or another element. The derived classes `SimpleElement` and `Item` override it by accepting both the name of the node/element to be created together with its content.

In the class `SimpleElement`, the constructor `NewElement` is denoted as:

`NewElement`(sName: `String`, stContent: `SimpleType`)

where:

- sName is the tag name of the element to be created
- stContent is the content/value of the simple element.

In the class `Item`, the constructor `NewElement` accepts the name of the element to be created and its content which is a collection of `SimpleElement` type. It is denoted as:

`NewElement`(sName: `String`, newElts: `[]SimpleElement`)

where:

- sName is the tag name of the item.
- newElts is an array of simple element.

The constructor creates an item named sName having each member of newElts as child.

To illustrate these, we provide two examples:

Example 5.1: Creating simple element: create an element named title having textual content "Ministers among Somalia blast dead":

$NewElement$("title", "Ministers among Somalia blast dead")

Example 5.2: Creating complex element: create an item having a title "Senior US diplomat resigns over war in Afghanistan" and published on "Thu, 03 Dec 2009 07:27:47 EST". Notice that, { , } is used to show comma separated list of values.

$$NewElement\left("item",\ \begin{Bmatrix} NewElement("title", "Ministers among Somalia blast dead"), \\ NewElement("pubDate",\ "Thu, 03 Dec 2009 07:27:47 EST") \end{Bmatrix}\right)$$

The embedded `NewElement` constructor returns simple element; it is equivalent to:

$$NewElement\left("item",\ \begin{Bmatrix} <title>Ministers\ among\ Somalia\ blast\ dead</title>, \\ <pubDate>\ Thu, 03\ Dec\ 2009\ 07:27:47\ EST</pubDate> \end{Bmatrix}\right)$$

and its result is shown in Figure 5.2.

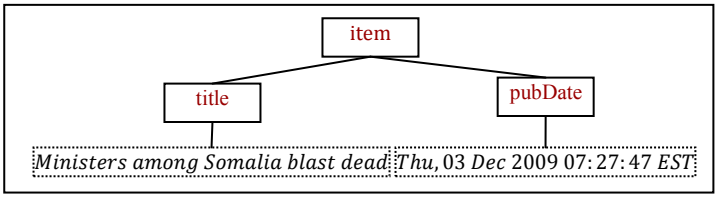


Figure 5.2: Tree representation of NewElement constructor

Definition 5.3. [Date Stream]

A data stream is a sequence of data transmitted in a time dependent manner. The source of the stream sends the data in either asynchronous or synchronous manner using a push or pull strategy. ■

RSS feed is streamed in an asynchronous and pull strategy. For instance, the RSS readers request for the list of changed news items since last download; and the provider transmits them. As RSS news items are time oriented, and its management can be handled using a window having time based boundary conditions (cf. window type in Definition 5.1).

5.3 RSS Algebra

We have categorized the operators into three categories: *extraction*, *set members* and high level *merge*. The extraction operators are dedicated to retrieve data from the database and include selection and its extension TopK, and join. Each of these operators accepts a set of windows and a supportive parameter set. Let us provide the 6 types of functions followed by the definition of the notion of selection predicate and the associated parameters.

We have categorized the functions into 6 types and presented as follows:

1. **String functions:** accept string parameters and/or return a string or a collection of strings as result
   - `String` *`Concat`*(T_1: `String`, T_2: `String`, delimiter: `String`): returns the concatenation of two texts T_1 and T_2 separated by a delimiter

- `String[]` *`getConcepts`*(T_1: `String`) :returns the concept set of the text T_1 that supports Definition 3.13
- `String[]` *`getCommon`*(T_1: `String`, T_2: `String`): returns the shared concepts of the two texts (i.e., intersection of *`getConcepts`*(T_1) and *`getConcepts`*(T_2))
- `String[]` *`getDifferent`*(T_1: `String`, T_2: `String`): returns the set of concepts existing as the member of concept set of only one text (i.e., exclusive or of *`getConcepts`*(T_1) and *`getConcepts`*(T_2))
- `String` *`BuildText`*(CS: `[]String`): returns a phrase/sentence that contains all the concepts in concept set CS. For instance, using the Google API, *`BuildText`* returns the first www document containing all concepts in CS.
- `String` *`LCA`*(T_1: `String`, T_2: `String`): returns the least common ancestor of two concepts T_1 and T_2 w.r.t. a reference Knowledge Base

2. **Similarity functions:** accept as input mainly two objects (`texts`, `simple elements` or `items`) and a semantic flag (to consider semantics or not) and returns a value between 0 and 1
   - `Decimal` *`TSIM`*(T_1: `String`, T_2: `String`, ConsiderSemanticFlag: `Boolean`): returns a similarity value between the two strings T_1 and T_2 considering semantics if ConsiderSemanticFlag is `True`. One way to do this is to identify the concept sets of T_1 and T_2, build their corresponding vectors, and compute their similarity value. (cf. text relatedness algorithm in Section 0)
   - `Decimal` *`ESIM`*(e_1: `SimpleElement`, e_2: `SimpleElement`, ConsiderSemanticFlag: `Boolean`): returns a similarity value between two simple elements while combining the similarity between tag names and contents.
   - `Decimal` *`ISIM`*(i_1: `Item`, i_2: `Item`, ConsiderSemanticFlag: `Boolean`, ic: `[]String`): returns a similarity value between items. Item similarity value is

computed by combining the similarity between those elements whose tag names are in the item connector -ic (which is an additional parameter).

3. **Relationship function:** accepts two objects (texts, simple elements or items), a semantic flag ConsiderSemanticFlag, and three optional parameters (equality threshold T_{Equal}, disjointness threshold $T_{Disjointness}$, and item connector ic) and returns the name of the relationship existing between the objects.
   - `String` *`getRelation`*(o_1: `Object`, o_2: `Object`, ConsiderSemanticFlag: `Boolean`, T_{Equal}: `Double?`, $T_{Disjointness}$: `Decimal?`, ic: `[]String?`): returns a string value that represents the relationship (i.e., either Equal, Include, Overlap, or Disjoint) between input objects
4. **Boolean/Logical functions:** accept two objects (texts, simple elements or items), a semantic flag ConsiderSemanticFlag, and three optional parameters (equality threshold T_{Equal}, disjointness threshold $T_{Disjointness}$, and item connector ic) and return a Boolean value
   - `Boolean` *`IsX`*(o_1: `Object`, o_2: `Object`, ConsiderSemanticFlag: `Boolean`, T_{Equal}: `Decimal?`, $T_{Disjointness}$: `Decimal?`, ic: [ ]`String?`): returns `True` if the relationship between o_1 and o_2 is X
     where, $X \in \{Equal, Include, Overlap, Disjoint\}$
   - `Boolean` *`IsSimilar`*(o_1: `Object`, o_2: `Object`, ConsiderSemanticFlag: `Boolean`, T_{Equal}: `Decimal`, ic: [ ] `String?`): returns `True` if o_1 and o_2 are similar, i.e., the similarity value (computed taking into consideration semantics if ConsiderSemanticFlag is `True` and/or the item connector ic) is greater than or equal to the T_{Equal}
5. **Complex function:** returns a value of non simple type such as `Element`, `Element-Linkage`, or `Object`
   - `Element-linkage[]` *`getCorrespondence`*(`i`$_1$`:Item,i`$_2$`:Item)`: returns a collection/list containing correspondences between sub-elements of each item.

Each member *e* of the array/result has three components $\langle e_1, e_2, r_{12} \rangle$ where e_1 is a sub-element of i_1, e_2 is a sub-element of i_2, and r_{12} is the relationship between e_1 and e_2. Notice that correspondence between items is identified on the basis of maximum similarity between elements.

6. **Merging functions:** accept two elements (simple or items) and return the result of merging (i.e. putting them together in a given pattern) w.r.t. the merging rule (if any). These functions are detailed in Section 4.5 and are provided in Table 4.3 and formalized here as follows:
   - `Item` *`KeepLatest`*(i_1: `Item`, i_2: `Item`) returns the latest/recent news item.
   - `Element` *`KeepFirst`*(e_1: `Element`, e_2: `Element`) returns the first element e_1.
   - `Element` *`KeepSecond`*(e_1: `Element`, e_2: `Element`) returns the second element e_2
   - `SimpleElement` *`Concat`*(e_1: `SimpleElement`, e_2: `SimpleElement`, delimiter: `string`): returns a new element e_k. The name of e_k is the least common ancestor of e_1 and e_2 tag names, and the content is build by concatenating the content of e_1 and e_2 separated by a delimiter (i.e., e_k ::= *`newElement`*(*`LCA`*(*e_1.name,e_2.name*),*`Concat`*(*e_1.content,e_2.content*, delimiter))
   - `Element` *`KeepBoth`*(i_1: `Item`, i_2: `Item`, deepconcat: `Boolean`, delimiter: `String`, vB: `String?`): returns an item that contains the result of concatenating the corresponding sub-elements of i_1 and i_2 if deepconcat is `True`; otherwise, it returns an element named vB having i_1 and i_2 as children
   - `Element` *`Merge`* (e_1: `Element`, e_2: `Element`, Merging_Rule: `[]String`) returns the result of merging two elements e_1 and e_2 using the associated merging rules. Notice that the Merging_Rule is dependent on the type of the inputs.

Definition 5.4. [Selection Predicate]

A selection predicate is a logical expression used to restrict the result of RSS query expressed with XQuery 1.1 (ROBIE, J. et al., 2009). A selection predicate p is used in the `where` clause of FLWOR expression. It is denoted as:

$$\mathrm{p} ::= \mathrm{A}\ \theta^{\mathrm{PR}}\ \mathrm{V} \tag{5.1}$$

where:

- *A* is an operand and it might be an element tag name or value
- $\theta \in \{SIM\} \cup \{IsX, IsSimilar\} \cup \{=, <, \leq, >, \geq, \neq, like, contains, \ldots\}$, *SIM* is an operator implemented as one of the similarity -`TSIM`, `ESIM`, `ISIM`
- PR is a parameter set associated to the operator θ. It contains similarity threshold, semantic flag, etc. as detailed in Definition 5.5
- *V* is an operand, it might be a value in the domain of *A*

A selection condition is defined as:

1. a simple selection predicate
2. a combination of simple selection predicates with conjunction, disjunction or negation. i.e., if $p1$ and $p2$ are simple selection predicates then following are selection predicates
   i) $p1 \wedge p2$, ii) $p1 \vee p2$ iii) $-p1$ ■

Notice that, in this report, we use selection predicate and selection condition interchangeably.

Table 5.2: Notations used for semantic aware RSS algebraic operators

Symbol	Meaning	Symbol	Meaning
δ_p	Similarity selection	δ_p^k	TopK
$\cup$	Union	$\Cup$	Additive Union
$\cap$	Intersection	$\Cap$	Additive Intersection
$\setminus$	Difference	$\oplus$	Merge
$\bowtie$	Similarity Join	$\overrightarrow{\theta}$	Symmetric operator θ

Definition 5.5. [Parameter of Operator]

Given a similarity operator θ, its parameter set PR contains a threshold value T_{Equal}, semantic flag ConsiderSemanticFlag, an item connector ic, and/or a set of merging rules Merging_Rule. We categorized the parameter set $PR \in \{PT, PI, PM\}$ into three:

1. PT ::= {T_{Equal}, ConsiderSemanticFlag}: represents the equality threshold (decimal value) and Boolean value that determine the use of semantics or not. It is used in selection and TopK operators.
2. PI ::= PT $\cup$ {ic}: represents the content of PT along with an item connector. It is used in selection, join, intersection, and difference operators.
3. PM ::= PI $\cup$ {Merging_Rule}: represents the content of PI along with the set of merging rules that would be used in the merge operator. ■

In order to facilitate the readability of a query expression in this report, we use θ and $\overset{PR}{\theta}$ interchangeably.

Definition 5.6. [Semantically Enhanced Predicate]

Given a selection predicate $p := A\ \theta\ V$ in which the operator $\theta \in \{=, <, \leq, >, \geq, \neq, like, contains, \ldots\}$, p is semantically enhanced if the operands and operator are rewritten with concepts extracted from a Knowledge Base. It is defined as follows.

Given three Knowledge Bases -label (LKB), operator (OKB) and value (VKB), p' is the semantically enhanced form of p if each term t in the attribute A, the operator θ and the

value v are rewritten with related concepts extracted from their corresponding Knowledge Base within a given threshold ε. It is formalized as:

$$p' = \overline{N_{LKB,\varepsilon}}(A)\ \overline{N_{OKB,\varepsilon}}(\theta)\ \overline{N_{VKB,\epsilon}}\big(getConcepts(V)\big) \quad (5.2)$$

where:

- `getConcepts` is a function that returns the concept set of the value V
- $\overline{N_{LKB,\varepsilon}}(A)$ is the global semantic neighborhood of attribute A using label Knowledge Base LKB
- $\overline{N_{VKB,\epsilon}}\big(getConcepts(V)\big)$ is the global semantic neighborhood of the concept set of the value to be searched using the value Knowledge Base VKB
- $\overline{N_{OKB,\varepsilon}}(\theta)$ is the global semantic neighborhood of the operator θ using the operator Knowledge Base OKB shown in Figure 5.3.

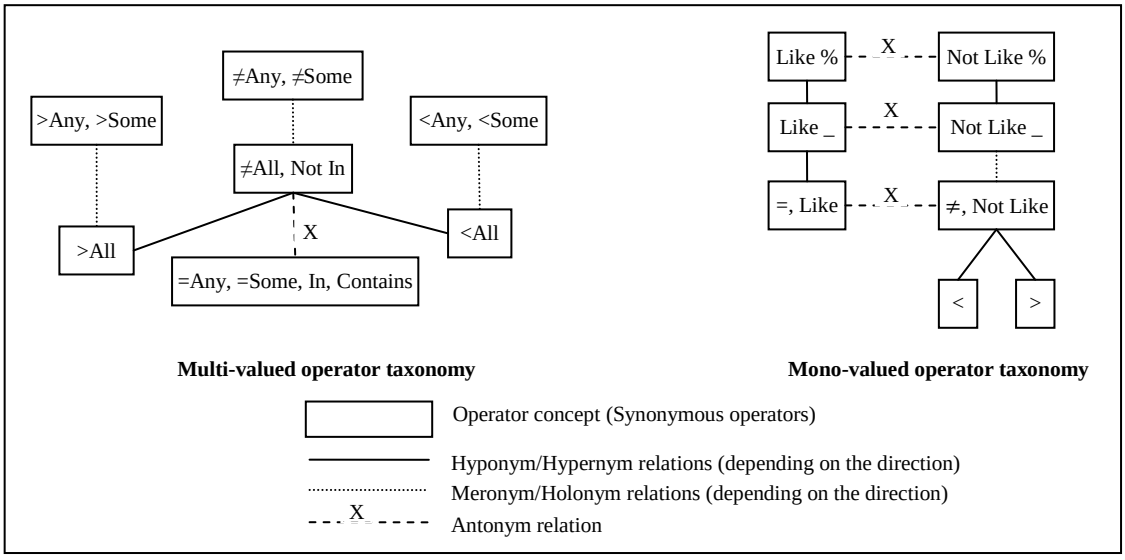


Figure 5.3: Sample operator Knowledge Base extracted from (Getahun et al., 2007)

For instance, referring to the operator Knowledge Base in Figure 5.3, and the label and value Knowledge Bases in Figure 3.3, the predicate $p \coloneqq '\mathit{description}'\ \mathit{contains}\ '\mathit{car}'$, is semantically enhanced within a threshold of 0 to:

$p' = \overline{N_{LBKB,0}}(\mathit{description})\ \overline{N_{OKB,0}}(\mathit{contains})\ \overline{N_{VKB,0}}(\mathit{car})$. i.e.,

$p' \coloneqq$ {$\mathit{description}$, $\mathit{summary}$, $\mathit{content}$} {contains, =any, =some, in} {car, auto, $\mathit{automobile}$}

Hence, the query processor would look for those news items having *car*, *auto* or *automobile* values in *description*, *summary* or *content* elements of the feed, rather than being restricted only to *description*.

5.3.1 Similarity Selection

Similarity selection is a unary operator on a window *w*. It is defined as follows.

Definition 5.7. [Similarity Selection (δ_p)]

Given a window w and a query predicate p, the similarity selection denoted as $\delta_p\ (w)$ and returns all elements in w satisfying the predicate p. ■

The basic building block of our similarity computation is the texts similarity function (i.e., `TSIM`). The simple element and item similarity functions call this function directly or indirectly. The result of our relatedness approach detailed in Section 3.6 is used here. For each element *e* in the input window, the text similarity selection works in three steps:

Step 1. Extract the concept sets of both operands and builds the associated vectors.

Step 2. If semantic flag is True, rewrite each unreferenced concept with the most similar concept from the other text.

Step 3. Compute the similarity between the texts using vector based similarity method.

The Pseudo Code 7 details the similarity selection operator. It accepts a window *w*, an attribute *A* (element tag name), a value *V* to be searched, an operator θ with its associated

parameter set PR. The value *flag* in PR determines the use of external Knowledge Base or not, and the threshold value controls the degree of similarity.

For each element *e* in the input w, the operator computes the similarity value after identifying the type of the operator θ. In lines 7-20 (case of similarity operator), the type of the value *V* to be searched is checked with `GetType` which is `String`, `SimpleElement` or `Item`. Notice that, the XPath expression *e//A* selects a descendent node of *e* named *A*. Then, a similarity value between *e//A* and *V* is computed using the functions `TSIM` (line 8), `ESIM` (line 10) or `ISIM` (line 12). The function `TSIM` extracts the concept sets of content of *e//A* and *V*, builds their corresponding vectors, and finally computes the similarity value (this can be done using for instance the vector similarity method such as cosine as detailed in (GETAHUN, F. et al., 2009)). In line 14, if the computed similarity *simv* is greater than the equality threshold, a Boolean *Found* is set.

Lines 18-20 check if the element *e* satisfies the predicate that uses a similarity based Boolean operator - `IsX` or `IsSimilar`. If *e* satisfies the predicate, a dummy similarity value of 1 is assigned to *simv*. In lines 21- 24, a new element named *sim* with content *simv* is added as child of the element e. The modified element *e* is finally added to the result set using `AddElement`. The content of *sim* would be used later to rank the result in some pattern (for instance, used to extend selection to TopK selection operation).

Pseudo Code 7: Similarity Selection operator

```
1.  Input:
        W: Window, A: String, V: Object, θ: Operator, Pr: Object
        // list of elements, attribute name, value to be searched, operator and its parameter
    Variable:
        Simv: Double                                  // similarity value
2.      Found: Boolean                                // flag to indicate query is found
3.  Output:
        Result: Window                                //list of elements satisfying predicate
4.  Begin
5.    Foreach e in w
6.      IF (Typeof(θ) = "Similarity" Then
7.        IF (V.GetType() = "String" Then             // text similarity selection
8.          Simv =TSIM(e//A.Content,V,PR.ConsiderSemanticFlag)
9.        Else IF(V.GetType() = "SimpleElement" Then
10.         Simv = ESIM(e//A, V, PR.ConsiderSemanticFlag)
11.       Else
12.         Simv = ISIM(e//A, V, PR.ConsiderSemanticFlag, PR.ic)
13.       End IF
14.       IF(Simv ≥ PR.T_Equal) Then
15.         Found = True
16.       End IF
17.     Else IF(Typeof(θ) = "Boolean" And θ(e//A, V, PR) = True Then
18.       Simv = 1
19.       Found = True
20.     End IF
21.     IF Found Then     //add element sim containing similarity score
22.       e.AppendChild(newElement("sim", Simv)
23.       Result.AddElement(e)
24.     End IF
25.     Found = False
26.   Next
27.   return Result
28. End
```

To illustrate this, we provide two examples showing similarity selection and Query by Example. Let w_1 be a window defined on CNN news published between 5 and 7 o'clock on December 3, 2009 and PT = {0.6, True}.

Example 5.3: Similarity selection: Identify all news in w_1 having title element describing "Bus explosion in Damascus" (with a similarity value of 0.6).

The selection query is represented as: $\delta_{title\ \underset{sim}{\{0.6,\text{True}\}}\ \text{"Bus explosion in Damascus"}}(w_1)$. The operator identifies the concept sets of "Bus explosion in Damascus" (i.e., 'bus, 'explosion' and 'Damascus'). Text based selection operator rewrites each concept in query and the title of each item in w_1 with its semantically related concepts. W.r.t. WordNet taxonomy (WORDNET 2.1, 2005), 'Bus' is related to 'autobus, coach, public transport, fleet', and 'explosion' is related to 'detonation, blow, blowup, etc' and 'Damascus' is related to 'capital of Syria, Syria, etc'. Hence, the retrieval is not restricted only to "Bus explosion in Damascus" and rather returns all news items with semantic similarity value greater than or equal to 0.6.

XQuery representation of the RSS selection operator is:

```
<svRoot> {
  for $e in w where $e/title {0.6,True}
                             sim
                                 "Bus explosion in Damascus"
  return $e
} </svRoot>
```

```
<item>
  <title>Bus blast kills, hurts dozens in Syria</title>
  <guid>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</guid>
  <link>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</link>
  <description>An explosion killed dozens of passengers in a bus in the Syrian capital Thursday morning, officials said.</description>
  <pubDate>Thu, 03 Dec 2009 06:03:40 EST</pubDate>
  <sim>0.63 </sim>
</item>
```

Figure 5.4: Sample result of the RSS selection operator

The operator *sim* (which is text similarity) returns True value if the content of title element is similar to "Bus explosion in Damascus" or related concepts with a similarity value of at least 0.6. Figure 5.4 shows sample result of the RSS query.

This query can also be formulated using a predicate that uses simple element similarity and represented as: $\delta_{title\ \underset{IsSimilar}{\{0.6,\text{True}\}}\ \text{<title>Bus explosion in Damascus </title>}}(w_1)$. The equivalent XQuery expression:

```
<svRoot>{
  for $e in w
  where $e/title {0.6,True} IsSimilar < title > Bus explosion in Damascus  </title >
  return $e
}</svRoot>
```

The result of this operator `IsSimilar` (simple element similarity) is the same as the text selection result shown in Figure 5.4 as the two elements have the same tag name.

Example 5.4: Query by Example: Given a sample news item I_1 (extracted from the BBC news service and shown in the `where` clause of XQuery expression given below), retrieve all news items in w_1 similar to it.

The user query is represented as: $\delta_{item\ \underset{sim}{\{0.6,True,\{title,description\}\}}\ I_1}(w_1)$ in which {title, description} is used as item connector. The equivalent XQuery expression is:

```
<svRoot>{ for $e in w1
  where e {0.6, True,{title, description}} sim <item><title>Deadly bus explosion in Damascus
  </title><description>Three people die as an explosion hits a bus in the Syrian capital
  Damascus, but officials say it was not a terrorist act</description></item>
  return $e
} </svRoot>
```

The operator *sim* is item selection function – `ISIM` and the result of the query is also similar to the one shown in Figure 5.4.

5.3.2 TopK similarity selection

TopK is a unary and specialized form of similarity selection operator defined on a window *w*. It is defined as follows.

Definition 5.8. [TopK Similarity Selection (δ_p^K)]

Give a window *w*, a query predicate *p*, and an integer *K*, the TopK operator denoted as $\delta_p^K(w)$ selects the *K* most similar elements in *w* satisfying the predicate p. ■

The predicate *p* is written in the form of Equation (5.1) and hence contains an attribute *A*, a value *V* to be searched (which can be text, or element), a parameter set *PR*. It is formalized as:

$$\delta_p^K(w) \equiv (e_1, ..., e_k)_{\succ} = \{e_i \in \delta_p(w) \mid sim(e_i/A, V, PR) \geq sim(e_j/A, V, PR)\} \quad (5.3)$$

where:

- $1 \leq i \leq j \leq K$
- sim is a similarity function that returns the similarity score between attribute *A* defined in element e_i and *V*, and attribute *A* defined in e_j and *V*.

The TopK similarity selection implemented in Pseudo Code 8 works in three steps.

Step 1. Identify the candidates of the TopK operator. This is performed using the similarity selection of *p* on *w* (where *p* a selection predicate defined over the attribute *A* and value *V*) and returns all elements in *w* satisfying the predicate as shown in Line 3.

Step 2. Sort the candidate list in descending order using the similarity value. It is to be recalled that the result of similarity selection operator has *sim* child element denoting the degree of similarity. Sorting can be done using the *Order By* clause of XQuery as done in line 4.

Step 3. Extract the first *K* elements of the sorted list. Lines 5-7 extract such list using a simple iteration statement (for loop).

Pseudo Code 8: TopK Similarity Selection operator

```
Input:
1.    w: Window, a: String, v: Object, pi: Object, k: Integer
      // array of elements, attribute, value to be searched, search operator, #items searched
Output:
2.    result : Window                    //list of elements satisfying predicate
Begin
3.    temp = δ_p (w)                     // select list of elements in w satisfying p
4.    Sort (temp, using sim in descending)    // sort the list in descending order
      //for $s in temp order by $s/sim/text( ) descending
5.    for i = 1 to k
6.      result.addElement (temp[i])
7.    Next
8.    return result
9.  End
```

Notice that, the equality threshold value in *PI* can be 0 and the TopK operators returns the first K elements most similar to the query which might include the dissimilar elements.

To illustrate this, let's consider the Example 5.5.

Example 5.5: TopK: Show the first 2 most similar news items published by CNN (between 5 and 7 o'clock on December 3, 2009) and similar to the sample news item I_1 used in Example 5.4 while considering semantics.

Let w_1 be the window defined on feed and let PI be {0.0, True, {title, description}}. The TopK query here is represented as: $\delta^{2}_{item\ \underset{ISIM}{\{0.0,True,\{title,description\}\}}\ I_1}(w_1)$. The item connector is defined on *title* and *description* elements with a similarity value of 0.0. Figure 5.5 shows the result sorted in descending order on the *sim* element (i.e., the similarity score

between the input I_1 and result). Here, the second news item is less similar to the input query I_1 (<item><title>Deadly bus explosion in Damascus ></title> … </item>) as similarity score is 0.23.

```
<item>
   <title>Bus blast kills, hurts dozens in Syria</title>
   <guid>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</guid>
   <link>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</link>
   <description>An explosion killed dozens of passengers in a bus in the Syrian capital Thursday morning, officials
    said.</description>
   <pubDate>Thu, 03 Dec 2009 06:03:40 EST</pubDate>
   <sim>0.63</sim>
</item>
<item>
   <title>Ministers among Somalia blast dead</title>
   <guid>http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition</guid>
   <link>http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition</link>
   <description>An explosion at a graduation ceremony in the Somali capital Thursday killed at least 15 people,
    including three government ministers and nine students, local journalists told CNN.</description>
   <pubDate>Thu, 03 Dec 2009 07:27:47 EST</pubDate>
   <sim>0.23</sim>
</item>
```

Figure 5.5: The result of selecting the top 2 news similar to I_1

5.3.3 Set membership operators

Our set membership operators are binary and accepts two windows defined on streams. The operators are different from XPath sequence based operators *op*:*union*, *op*:*intersect* and *op*:*except* (difference) (CLARK, J. and DeRose, S., 1999) mainly for two reasons: (1) our operators don't intentionally remove duplicate values. This is due to the fact that each news item is published in a given time and it is unique. But, the same/different publishers might publish the same news at different time; and (2) the similarity between nodes/elements is semantic- and syntactic- aware. In XPath, two nodes are identified to be equal using the operator *fn:deep-equal* which checks deep-nested strict equality without considering semantic information.

Notice that, the main challenge in doing set membership operations in database system is identifying identical elements or members. In RSS context, we let a user to suggest the set of tag names that might be used to identify and connect items (cf. Definition 3.21).

Thus, the set membership operators intersection and difference accept item connector in the parameter set *PI*.

Definition 5.9. [Intersection (∩)]

Given two windows w_1 and w_2, the intersection operator, denoted as $w_1 \overset{\text{PI}}{\cap} w_2$, returns all elements of w_1 which have at least one similar element in w_2. It is formally defined as:

$$w_1 \overset{\text{PI}}{\cap} w_2 = \{e | e \in w_1 \wedge \exists e_i \in \delta_p(w_2)\} \quad (5.4)$$

where, p is a selection predicate i.e., $p ::= A \overset{PI}{\theta} e$ and $\theta \in \{SIM\}$ ■

For each element e in the window w_1, the similarity selection operator $\delta_p(w_2)$ returns all elements e_i in w_2 similar to it. Hence, the intersection operator returns all elements existing in both windows.

From the definition above, the following property is identified.

Property 5.1. The intersection operator is not commutative.

Proof:

The proof of this property is trivial as the intersection operator keeps only members of the first operand and it is likely that the news could be published by different publishers at different time. Hence, $w_1 \overset{\text{PI}}{\cap} w_2 \neq w_2 \overset{\text{PI}}{\cap} w_1$

Example 5.6: Intersection of two windows: Retrieve all news items of CNN having 80% similar title content as the title content of news items in BBC and published between 5 and 7 o'clock on December 3, 2009 while considering semantic.

Let w_1 and w_2 be windows defined on the CNN and BBC feeds and $\text{PI} = \{0.8, True, \{Title\}\}$. The query is represented as $w_1 \overset{\text{PI}}{\cap} w_2$. Figure 5.6 shows the partial result, this item is shown as the news item *BBC2* is identical to the query (i.e., <title>Bin Laden not in Pakistan, says PM</title>). Notice that, the link, guid elements of CNN2 and BBC2 are different.

```
…
<item>
   <title>Bin Laden not in Pakistan, PM says</title>
   <guid>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</guid>
   <link>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</link>
   <description>Pakistan's prime minister Thursday rejected claims Al Qaeda leader Osama bin Laden is hiding within his
   country as global pressure mounted on Islamabad to tackle terrorists linked to escalating conflict in neighboring
   Afghanistan.</description>
   <pubDate>Thu, 03 Dec 2009 06:23:16 EST</pubDate>
</item>
…
```

Figure 5.6: Partial result of CNN ∩ BBC

Definition 5.10. [Difference (\)]

Given two windows w_1 and w_2, the difference operator, denoted as $w_1 \backslash w_2$, returns all elements existing only in w_1 (i.e., an element e in w_1 is added to result if and only if there is no element e_i in w_2 similar to it). The operator is similar to the relational counterpart *Except*. It is formalized as:

$$w_1 \overset{PI}{\backslash} w_2 = \{e \in w_1 \mid \nexists e_i \in \delta_p(w_2)\} \tag{5.5}$$

where:

- p is a selection predicate i.e., $p ::= A_\theta^{PI}\ e$ and $\theta \in \{SIM\}$
- $\delta_p(w_2)$ selects all elements similar to e. ■

Notice that, the difference of two unbounded windows (each defined on separate source) extract all news items published only by the first source. Example 5.7 illustrates this.

From the definition above, the following property is identified.

Property 5.2. The difference operator is not commutative

Proof:

The proof of this property is trivial as the operator keeps only elements of the first operand.

Example 5.7: Difference of two windows: Retrieve all news items of CNN without 80% semantically similar title elements to those in BBC and published between 5 and 7 o'clock on December 3, 2009.

Let w_1 and w_2 be windows defined 5 and 7 o'clock news of CNN and BBC feeds and PI is {0.8, True, title}. Figure 5.7 shows the partial result of the query $w_1 \overset{PI}{\setminus} w_2$.

```
<item>
  <title>Bus blast kills, hurts dozens in Syria</title>
  <guid>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</guid>
  <link>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</link>
  <description>An explosion killed dozens of passengers in a bus in the Syrian capital Thursday morning, officials
    said.</description>
  <pubDate>Thu, 03 Dec 2009 06:03:40 EST</pubDate>
</item>
```

Figure 5.7: Result of CNN \ BBC

Definition 5.11. [Union (∪)]

Given two windows w_1 and w_2, the union operator, denoted as $w_1 \cup w_2$, returns all elements of w_1 or elements of w_2 without removing duplicates. It is formalized as:

$$w_1 \cup w_2 = \{e \mid e \in w_1 \vee e \in w_2\} \quad (5.6)$$

The result of union operator is similar to the UNION ALL relational operator and the Union operator in SPARQL Query Language for RDF (PÉREZ, J. et al., 2009).

From the definition above, the following property is identified.

Property 5.3. The union operator is commutative.

Proof:

As RSS is unordered collection of elements and the union operator keeps members of each window, the $w_1 \cup w_2$ and $w_2 \cup w_1$ returns the same result. Hence, it is commutative.

Example 5.8: Union of two windows: Show all CNN and BBC news published between 5 and 7 o'clock on December 3, 2009.

Let w_1 and w_2 be windows defined on the feeds. The query is represented as: $w_1 \cup w_2$. The XQuery equivalent expression is:

```
for $u in w1 return $u
for  $v in w2 return $v
```

The result of this query is list of items from each window first from w_1 followed by items in w_2 as shown in Figure 5.8. Even though, CNN2 and BBC2 are identical news items (using title) the duplicate is not removed.

```
…
<item>
  <title>Bin Laden not in Pakistan, PM says</title>
  <guid>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</guid>
  <link>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</link>          CNN2
  <description>Pakistan's prime minister Thursday rejected claims Al Qaeda leader Osama bin Laden is hiding within
  his country as global pressure mounted on Islamabad to tackle terrorists linked to escalating conflict in neighboring
  Afghanistan.</description>
  <pubDate>Thu, 03 Dec 2009 06:23:16 EST</pubDate>
</item>
<item>
  <title>Bus blast kills, hurts dozens in Syria</title>
  <guid>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</guid>          CNN3
  <link>http://edition.cnn.com/2009/WORLD/meast/12/03/syria.bus.blast/index.html?eref=edition</ link >
  <description>An explosion killed dozens of passengers in a bus in the Syrian capital Thursday morning, officials
  said.</ description>
  <pubDate>Thu, 03 Dec 2009 06:03:40 EST</pubDate> </item>
<item>
  <title>Bin Laden not in Pakistan, says PM</title>
  <description>Pakistan's prime minister tells UK counterpart Gordon Brown he does not think Osama Bin Laden is          BBC2
    in his country.</description>
  <link>http://news.bbc.co.uk/go/rss/-/2/hi/uk_news/politics/8392211.stm</link>
  <guid isPermaLink="false">http://news.bbc.co.uk/2/hi/uk_news/politics/8392211.stm</guid>
  <pubDate>Thu, 03 Dec 2009 11:05:52 GMT</pubDate>
</item>
```

Figure 5.8: Partial result of CNN ∪ BBC

5.3.4 Similarity join

The similarity join is a binary operator defined on two windows. We assume that the parameters associated to the operator (i.e., semantic flag, threshold value and/or ic) are provided by the user.

Definition 5.12. [Similarity join (⋈)]

Given two windows w_1 and w_2, w_1 similarity join w_2 using joining condition defined as a predicate p, denoted as $w_1 \overset{\bowtie}{p} w_2$, associates each element e in w_1 with the set of similar elements e_i in w_2 (i.e., e_i is the result of selecting e in w_2). It is formalized as:

$$w_1 \overset{\bowtie}{p} w_2 \;= \left\{ f(e, \{e_i\}, tn) \middle| e \in w_1 \;\wedge \exists e_i \in \delta_{p'}(w_2) \right\} \tag{5.7}$$

where:

- p is join condition denoted as $p = \; w_1.\{ic_1\}\, \theta\, w_2.\{ic_2\}$
- $\{ic_i\}$ is an item connector in window w_i (i = 1 or 2)
- $\theta \in \{SIM\}$ is a similarity function
- $\delta_{p'}(w_2)$ selects the list of elements in w_2 similar to the element e i.e., satisfying the predicate p' $(i.e., p' ::= w_2.\{ic_2\}\, \theta\, e.\{ic_1\})$
- f is a function provided by the user and determines the structure of the join result
- tn is a string – the tag name of the root. ■

The function f creates a new element tn that contains the e and e_i as contents. The default join function f is denoted as: $f(e, e_i, tn) ::= newElement\,(tn, \{e, e_i\})$. This function creates an element named *tn* having elements e and e_i as children which is the same as the result of calling the function that keeps both elements i.e., KeepBoth(e, e_i, False, , tn).

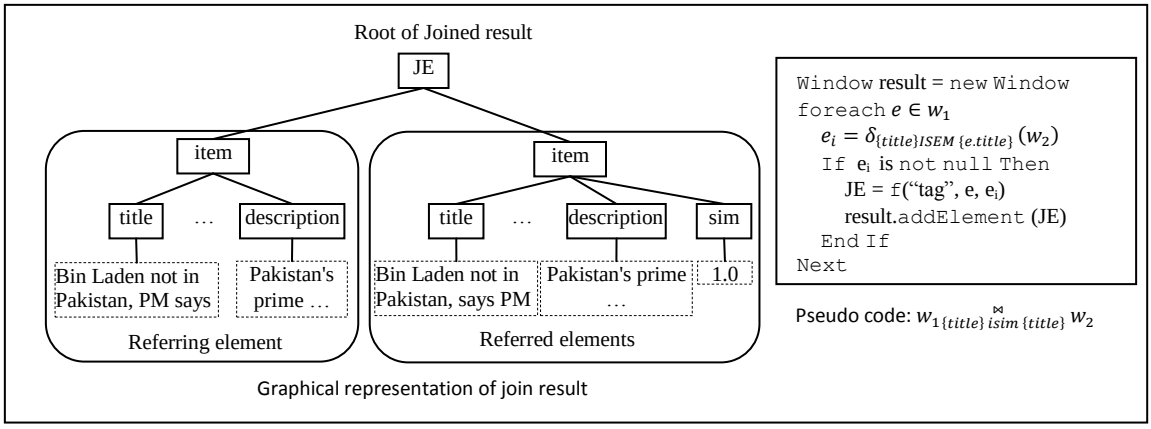


Figure 5.9: Tree representation of join result

In (FISHER, D. et al., April 2004), the authors noted the need to access the first *e* (named hereafter as referring element) and the second member e_i (called referred element) of the join result and defined two operators *fst* and *snd* respectively accomplishing these tasks. Here, we adopt the same idea so as to get referring and referred component of the join result. Figure 5.9 shows pseudo code and tree based join result of Example 5.9. Notice that each referred element has *sim* element showing the similarity between the referring and referred elements.

From the definition above, the following property is identified.

Property 5.4. The similarity-join operator is not commutative (i.e., $w_1 \bowtie w_2 \neq w_2 \bowtie w_1$)

Proof:

The proof of this property is trivial, as the elements in the join result are categorized as referring and referred (has a child element *sim*), the operator is not commutative.

Example 5.9: Joining windows: Consider Example 1.5 in Scenario 3, the request of the journalist can be represented as a join of the two windows defined on CNN and BBC within the start and ending timestamp (i.e. between 5 and 7 o'clock on December 3, 2009).

Assume that two news items are similar if there corresponding titles are 90% similar and place the result as children of a new element named 'JE'.

Let w_1 and w_2 be two windows defined on the CNN and BBC feeds. Elements from each window are linked only if their title elements are semantically similar with 90%. Figure 5.10 shows the partial result of executing the query expression: $w_{1\{title\}} \overset{\bowtie}{isim}_{\{title\}} w_2$.

The first child of JE is the referring item and all the remaining items are referred items. Each referred item has a sub-element *sim* with a score representing the degree of similarity it has with the referring item. JE.*fst* returns the referring element, whereas JE.*snd* returns the referred elements.

```
…<JE><item><title>Bin Laden not in Pakistan, PM says</title>
        <guid>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</guid>
        <link>http://edition.cnn.com/2009/WORLD/asiapcf/12/03/pakistan.bin.laden/index.html?eref=edition</link>
        <description>Pakistan's prime minister Thursday rejected claims Al Qaeda leader Osama bin Laden is hiding within
          his country as global pressure mounted on Islamabad to tackle terrorists linked to escalating conflict in neighboring
          Afghanistan.</description>
        <pubDate>Thu, 03 Dec 2009 06:23:16 EST</pubDate>
      </item>
      <item><title>Bin Laden not in Pakistan, says PM</title>
        <description>Pakistan's prime minister tells UK counterpart Gordon Brown he does not think Osama Bin Laden is in
          his country.</description>
        <link>http://news.bbc.co.uk/go/rss/-/2/hi/uk_news/politics/8392211.stm</link>
        <guid isPermaLink="false">http://news.bbc.co.uk/2/hi/uk_news/politics/8392211.stm</guid>
        <pubDate>Thu, 03 Dec 2009 11:05:52 GMT</pubDate>
        <sim>1.0</sim>
    </item>
  </JE> …
```

Figure 5.10: Partial results of CNN join BBC

Next, we provide the definition of Additive Union and Additive Intersection operators that would be used to extend similarity-based join and perform multi-windows join.

Definition 5.13. [Additive Union (⋓)]

Given two windows w_1 and w_2, each window contains a collection of elements generated as a result of a similarity-based join operation; w_1 additive union w_2 denoted as $w_1 \Cup w_2$ returns all elements either in w_1 or w_2 like union operator. Except if two elements

have the same referring elements, the operator combines the referred elements. It is formalized as:

$$w_1 \Cup w_2 = \{f_{\Cup}(u) | u \in w_1 \vee u \in w_2\} \quad (5.8)$$

where: $f_{\Cup}$ is a function that combines the referred component of those elements having similar referring elements. ■

The Pseudo Code 9 represents the operator ⋓. In lines 4-6, each member of w_1 is added directly to the final result. However, member e_1 of w_2 is added directly to the result set only if it doesn't exist in *Result* (line 17); otherwise, there is an element *e* already added in *Result* having identical referring element as e_1 (line 10); and these referred elements are combined (i.e., add referred element of e_1 as children of *e*) in lines 11-13.

Pseudo Code 9: Additive Union

```
    Input:
1.    W1 :Window,  W2 : Window
    Variable:
2.    Added : Boolean , e, e1 : Element
    Output:
3.    Result : Window
    Begin
4.    Foreach E In W1
5.      Result.Addelement(E)            // add member of w1 to result as it is
6.    Next
7.    Foreach e1 in W2
8.      For I = 1 To Result.Count
9.        e = Result[I]
10.       IF e1.fst = e.fst Then        //compare the referring elements
11.         e.AppendChild(e1.snd)       //add the refereed elements
12.         Added = True
13.         Result[I] = e
14.         Break                       // move out of the loop
15.       END IF
16.     Next
17.     IF Not Added Then Result.Add(e1)
```

Pseudo Code 9: Additive Union

```
18.     Next
19.     Return Result
        End
```

An example demonstrating additive union is shown in Example 5.11.

Definition 5.14. [Additive Intersection (⋒)]

Given two windows w_1 and w_2, each containing the result of similarity join operator, the additive intersection operator, denoted as $w_1 \Cap w_2$, returns all elements of w_1 having similar elements in w_2. It is formally defined as:

$$w_1 \overset{PI}{\Cap} w_2 = \{f_{\Cap}(e_i) | e_i \in w_1\} \quad (5.9)$$

where: $f_{\Cap}$ is a function that searches for elements of w_2 having similar referring and referred as e_i. ■

Pseudo Code 10 presents the ⋒ operator. This operator is the extended form of the intersection operator and it is implemented with Nested For Loop. For each element e_i in w_1, it searches for all elements e_j in w_2 having similar referring elements (line 8) followed by identifying the intersection of their corresponding referred element list (line 9). If their corresponding referred list intersects (i.e., there is at least one element in *temp*), an element named after tag name of e_i, containing the referring of e_i and the intersection of the referred list - *temp* is added to the final result (line 11).

From the definition above, the following property is identified.

Property 5.5. The additive intersection operator is not commutative (i.e., $w_1 \Cap w_2 \neq w_2 \Cap w_1$).

Proof:

The proof of this property is trivial as the result of the operator is always from the first window.

Pseudo Code 10: Additive Intersection

```
Input:
1.    w1 : Window, w2 : Window
2.    pi: Parameter-Set
Variable:
3.    added : Boolean, w : Element
Output:
4.    result : Window
Begin
5.   Foreach ei in w1
6.     root = ei.name                          // return the root element of joined result
7.     Foreach ej in w2
8.        If ei.fst = ej.fst Then              // compare the referring elements
9.           temp = ei.snd PI∩ej.snd
10.          If temp.Count > 0 Then
11.             result.Add (newelement(root, { ei.fst, Temp})   // new element
12.          End If
13.       End If
14.    Next
15.  Next
16.  Return result
End
```

This operator guarantees in getting common elements existing in different sources as demonstrated with the following example.

Example 5.10: Additive intersection: A journalist wants to get common news published in CNN, BBC and NYT between 5 and 7 o'clock on December 3, 2009. Let w_1, w_2 and w_3 be the windows defined on each source.

The query of the journalist can be interpreted as joining these three windows i.e., $w_1 \bowtie w_2 \bowtie w_3$. This comes down to joining w_1 with w_2 and w_1 with w_3 followed by additive intersection to get all news published by the three sources i.e., $w_1 \bowtie w_2 \bowtie w_3 = w_1 \bowtie w_2 \Cap w_1 \bowtie w_3$.

5.3.5 Symmetric similarity join

The similarity join defined in Section 5.3.4 doesn't possess the symmetric property needed by the query optimizer to facilitate query rewriting and simplification. We extend the similarity-based join operator to be suitable for query optimization using additive union operator as defined below.

Definition 5.15. [Symmetric Similarity Join ($\overleftrightarrow{\bowtie}$)]

Given two windows w_1 and w_2, w_1 symmetric similarity join w_2 using a joining condition defined as predicate p, denoted as $w_1 \overleftrightarrow{\bowtie}_p w_2$ and returns all news items in w_1 together with similar elements in w_2 and vice-versa. It is formalized as:

$$w_1 \overleftrightarrow{\bowtie}_p w_2 = w_1 \overset{\bowtie}{}_p w_2 \cup w_2 \overset{\bowtie}{}_p w_1 \quad (5.10)$$

where: $\cup$ is the *Additive Union* operator that combines the result of the semantic similarity join. ■

Example 5.11: Symmetric join of windows: A journalist wants to get linked news published in CNN and BBC having 90% similar title elements independent of the source and published between 5 and 7 o'clock on December 3, 2009; and put the result as children of a new element named 'JE'.

Let w_1 and w_2 be the two windows defined on the feeds. The expression $w_{1\{\text{title}\}} \overleftrightarrow{\bowtie}_{isim\ \{\text{title}\}} w_2$ represents the user query. Figure 5.11 is the partial result of the symmetric join expression in XML format. The additive union operator combines the result of CNN join BBC and BBC join CNN and keeps both as the redundancy elements because of the additive union.

Figure 5.11: Partial results of CNN symmetric join BBC

The symmetric similarity join is generalized to join multiple windows generated from the same or different sources. The multi-window similarity has important behavior that could be used by query optimizers which is cascading/parallelizing individual operations.

Definition 5.16. [Multi-Window Symmetric Similarity Join]

Given n ($n > 2$) windows $w_1, \ldots, w_n$, the multi-window symmetric similarity join operator returns all common elements of the n windows. ■

Theorem: The multi-window symmetric similarity join of n ($n > 2$) windows $w_1, \ldots, w_n$, denoted as $w_1 \overleftrightarrow{\bowtie} w_2 \ldots \overleftrightarrow{\bowtie} w_n$ is the result of combining the pair-wise symmetric similarity join of windows using additive intersection operator as formalized in Equation (5.11). i.e.,

$$w_1 \overleftrightarrow{\bowtie} w_2 \ldots \overleftrightarrow{\bowtie} w_n \;=\; \overset{\Cap}{_{1\le i,j\le n, i<j}} w_i \overleftrightarrow{\bowtie} w_j \tag{5.11}$$

Proof:

We prove the validity of this equation using proof by mathematical induction.

Prove for $n = 3$, i.e., show that $w_1 \overleftrightarrow{\bowtie} w_2 \overleftrightarrow{\bowtie} w_3 \;=\; \overset{\Cap}{_{1\le i,j\le 3, i<j}} w_i \overleftrightarrow{\bowtie} w_j$

Using the RHS of the equation, i.e.,

$\overset{\Cap}{_{1\le i,j\le 3, i<j}} w_i \overleftrightarrow{\bowtie} w_j = w_1 \overleftrightarrow{\bowtie} w_2 \Cap w_1 \overleftrightarrow{\bowtie} w_3 \Cap w_2 \overleftrightarrow{\bowtie} w_3$

By Definition 5.15, $w_1 \overleftrightarrow{\bowtie} w_2$, returns all elements existing in both windows

By Definition 5.14, $w_1 \Cap w_2$, returns all elements in w_1 having same referring and referred elements.

Hence, $\overset{\Cap}{_{1\le i,j\le 3, i<j}} w_i \overleftrightarrow{\bowtie} w_j$, returns all elements existing in all of the three windows.

Therefore, $\overset{\Cap}{_{1\le i,j\le 3, i<j}} w_i \overleftrightarrow{\bowtie} w_j = w_1 \overleftrightarrow{\bowtie} w_2 \overleftrightarrow{\bowtie} w_3$

Assume that, the equation is true for n > 3, W.N.T. the equation is true for n+1:

The RHS of the equation is simplified to be:

$\overset{\Cap}{_{1\le i,j\le n, i<j}} w_i \overleftrightarrow{\bowtie} w_j = \overset{\Cap}{_{1\le i,j\le n-1}} \left(w_i \overleftrightarrow{\bowtie} w_j \right) \Cap \left(\Cap_{i=1}^{n-1} w_i \overleftrightarrow{\bowtie} w_n \right)$, for i<j

$\overset{\Cap}{_{1\le i,j\le n-1}} \left(w_i \overleftrightarrow{\bowtie} w_j \right) = w_1 \overleftrightarrow{\bowtie} w_2 \ldots \overleftrightarrow{\bowtie} w_{n-1}$, using the assumption

$\Cap_{i=1}^{n-1} (w_i \overleftrightarrow{\bowtie} w_n) = \Cap_{i=1}^{n-1} (w_n \overleftrightarrow{\bowtie} w_i)$, using Definition 5.15

$\overset{\Cap}{_{1\le i,j\le n, i<j}} w_i \overleftrightarrow{\bowtie} w_j = (w_1 \overleftrightarrow{\bowtie} w_2 \ldots \overleftrightarrow{\bowtie} w_{n-1}) \Cap (\Cap_{i=1}^{n-1} (w_n \overleftrightarrow{\bowtie} w_i))$ returns all elements existing in all windows

Therefore, $w_1 \overleftrightarrow{\bowtie} w_2 \ldots \overleftrightarrow{\bowtie} w_{n-1} \overleftrightarrow{\bowtie} w_n = \overset{\Cap}{_{1\le i,j\le n, i<j}} w_i \overleftrightarrow{\bowtie} w_j$ ■

5.3.6 Merge

Given two windows w_1 and w_2, w_1 merge w_2, denoted as $w_1 \oplus w_2$, returns the result of combining their content based on a set of merging rules in *PM* (parameter associated to merge). It is formalized as

$$w_1 \overset{PM}{\oplus} w_2 \quad = \sum \text{Merge-action}\left(e, \{e_j\}\right) / e \in w_1, \; e_j \in w_2 \; \wedge Condition\left(e, e_j\right) = \text{True} \tag{5.12}$$

where:

- *Merge-action* is the action component of the users' merging rule associated particularly to a condition/predicated defined over the element *e* and e_j.
- $\sum$ represents any aggregation function

For each element *e* in w_1, the operator identifies all elements e_j in w_2 that make the merge condition component of the merging rule True. This is done following a pre-defined partial order defined for instance on relationship (*equal*, *include* and *overlap*). Then, it applies the specific *merge-action* specified in the merging rule set. As a result, the merge operator might return results that include the result of *Join* and *Intersection* operators depending on the used merge-action. It is to be noted that neither the *Join* nor *Intersection* operator identifies the news related with *overlap* relationship. To illustrate this, let's consider the following example.

Example 5.12: Merging two windows: Merge all news items published between 5 and 7 o'clock on December 3, 2009 in both CNN and BBC considering title element only, using the following merging rules: Keep the latest of identical news, keep the detailed news in case of inclusion; otherwise, keep both news items as children of VR.

Let w_1 and w_2 be windows defined on the feeds. The XQuery FLWOR expression shown in Figure 5.12 merges items based on the identified relationship and the associated merging rules.

```
let $s ::= w1 ⊕{0.8, true, {title}, {⟨equal,getLatest⟩, ⟨include,keepfirst⟩, ⟨true,keepboth/false,|,'VR'⟩}} w2
for $I in $s
return {$I}
```

Figure 5.12: Merging news items using a particular user's merging rule

The partial result of this query is shown in Figure 5.13. The CNN1 overlap with BBC1 and hence the operator puts both of them as the children of an element VR, whereas CNN2 and BBC3 news are identical so the latest news BBC3 is kept. This example clearly shows that the merge operator provides more result than *Intersection* and *Join* operators w.r.t. these merging rules.

```
<VR>
  <item>
    <title>Ministers among Somalia blast dead</title>
    <guid>http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition</guid>
    <link>http://edition.cnn.com/2009/WORLD/africa/12/03/somalia.attacks/index.html?eref=edition</link>
    <description>An explosion at a graduation ceremony in the Somali capital Thursday killed at least 15 people,
      including three government ministers and nine students, local journalists told CNN.</description>
    <pubDate>Thu, 03 Dec 2009 07:27:47 EST</pubDate> </item>
  <item>
    <title>Somali ministers killed by bomb</title>
    <description>A suicide bomber disguised as a woman kills at least 19 people, including government
      ministers, at a hotel in the Somali capital.</description>
    <link>http://news.bbc.co.uk/go/rss/-/2/hi/africa/8392468.stm</link>
    <guid isPermaLink="false">http://news.bbc.co.uk/2/hi/africa/8392468.stm</guid>
    <pubDate>Thu, 03 Dec 2009 13:24:49 GMT</pubDate> </item>
</VR>
```

CNN1 Overlaps with BBC1

```
<item>
  <title>Bin Laden not in Pakistan, says PM</title>
  <description>Pakistan's prime minister tells UK counterpart Gordon Brown he does not think Osama Bin Laden
    is in his country.</description>
  <link>http://news.bbc.co.uk/go/rss/-/2/hi/uk_news/politics/8392211.stm</link>
  <guid isPermaLink="false">http://news.bbc.co.uk/2/hi/uk_news/politics/8392211.stm</guid>
  <pubDate>Thu, 03 Dec 2009 11:05:52 GMT</pubDate>
</item>
```

CNN2 Equals BBC3

Figure 5.13: Partial result of merging CNN and BBC feeds

Property 5.6. The *Merge* operator generalizes the *set membership* and the *Join* operators.

Proof:

a) By definition, the *Union* of two windows is the set of elements existing in either of them. The Merge operator acts as union using the merging rule *KeepBoth* for any relationship.

i.e., $\cup \equiv \bigoplus^{\text{Merging\_rule ::=}\{\langle ALL-relations,\ KeepBoth\rangle\}}$

b) By definition, the *Intersection* of two windows is the set of elements existing in both and using the member of the first window as representative. Hence, using the merging rules *KeepFirst* for equality and *IgnoreAll* for the remaining relationships the merge operator acts intersection

i.e., $\cap \equiv \bigoplus^{\text{Merging\_rule ::=}\{\langle equal,Keepfirst\rangle,\ \langle else,IgnoreBoth\rangle\}}$

c) By definition, the *Difference* of two windows is the set of elements existing only in the first window. Using the merging rule *IgnoreBoth* for equality and *KeepFirst* for the remaining relations in the merge operator.

i.e., $\setminus \equiv \bigoplus^{\text{Merging\_rule ::=}\{\langle equal,IgnoreBoth\rangle,\ \langle Else,KeepFirst\rangle\}}$

d) By definition, the binary *Join* operator returns only similar elements of each window identified with a given similarity threshold. Consequently, the elements in join result set are related with equality relationship. Hence, the merge operator provides the same result using *KeepBoth* for equality relation and *IgnoreBoth* for all the remaining relationships.

i.e., $\bowtie \equiv \bigoplus^{Merging\_rule ::=\{\langle equal,KeepBoth\rangle,\ \langle Else,IgnoreBoth\rangle\}}$ ■

Property 5.7. The *Merge* operator is not symmetric.

Proof:

The result of the *Merge* operator is dependent on a set of merging rules provided by the user (c.f. Section 5.3.6).

Using Property 5.6, *Merge* generalizes the *set membership* operators among which *Intersection* and *Difference* are not symmetric.

Therefore, we can conclude that *Merge* is not symmetric. ■

Property 5.8. The *Selection* and *Merge* operators are the minimal set of operators need in RSS context.

Proof:

Selection operator retrieves those elements that satisfy a selection condition; and it is the base for the *TopK* operator.

As proved in Property 5.6, *Merge* generalizes *Join* and the *set membership* operators.

Therefore, *Selection* and *Merge* are the two basic operators in RSS context. ■

Table 5.3: The summary of commutativity property of each binary operator.

Binary operators	Commutativity property
Union	✓
Intersection	×
Difference	×
Similarity join	×
Symmetric similarity join	✓
Additive intersection	×
Additive union	✓
Merge	×

5.4 Query optimization

In database query processing system, there are a number of query plans that the DBMS can follow to execute a query. These query plans are equivalent in the final result. However, the time and cost needed in executing these plans might vary. Thus, the DBMS has query optimizer component that examines all possible alternatives and chooses the plan with lesser cost. As one of the optimization strategies, a query optimizer uses algebraic rewriting approach to transform a user query into algebraic expression. Then, the algebraic expression is transformed into equivalent and less costly expression. In this section, we provide a set of equivalent rules followed by a set of heuristic based optimization strategies.

5.4.1 Equivalent rules

In this section, we provide and prove some of the equivalent rules that would be used by RSS query optimizer. In particular, we study the behavior of the distribution of selection operator over extraction and set membership operators. To ease the readability of the equivalent rules, we prefer to use *Join*, and *set membership* operators directly rather than using *Merge*.

Given two windows w_1 and w_2, and selection conditions/predicates p_1 and p_2 (which aren't linked to the source of window) the following equivalence rules hold:

Rule 5.1. Cascading of similarity selection

Similarity selection defined on conjunction of predicates is equivalent to cascading of similarity selection over each predicates.

$$\delta_{p1\ AND\ p2}(\mathrm{w}_1) \equiv \delta_{p1}\left(\delta_{p2}(\mathrm{w}_1)\right)$$

Proof:

To prove the expression, consider the right hand side - RHS i.e.,

$\delta_{p1}\left(\delta_{p2}(\mathrm{w}_1)\right) = \{u \in \delta_{p1}(\mathrm{w}) | u$ satsifies the predicate $p2\}$ (by Definition 5.7)

$w \subseteq \mathrm{w}_1$ contains all elements satisfying predicate *p2* i.e., w is the result of $\delta_{p2}(\mathrm{w}_1)$

$= \{u \in \mathrm{w} | u$ satisfies the predicate $p1$ and predicat $p2\}$ (by Definition 5.7).

$= \left\{u \in \mathrm{w}_1 \middle| u \in \delta_{p1\ AND\ p2}(\mathrm{w}_1)\right\}$

$= \delta_{p1\ AND\ p2}(\mathrm{w}_1)$ ■

Rule 5.2. Commutativity of similarity selection

Similarity selection operator is commutative.

$$\delta_{p2}\left(\delta_{p1}(\mathrm{w}_1)\right) \equiv \delta_{p1}\left(\delta_{p2}(\mathrm{w}_1)\right)$$

Proof:

To prove the expression, consider the left hand side - LHS, i.e.,

$\delta_{p1}\left(\delta_{p2}(\mathrm{w}_1)\right) = \delta_{p1\ AND\ p2}(\mathrm{w}_1)$, (Using Rule 5.1)

$= \delta_{p2\ AND\ p1}(\mathrm{w}_1)$ (Logical AND is commutative)

$= \delta_{p2}\left(\delta_{p1}(\mathrm{w}_1)\right)$ (Using Rule 5.1) ■

Rule 5.3. Disjunction of similarity selection

Selection defined on disjunction of predicates is equivalent to the union of selection over individual predicates.

$$\delta_{p1\ OR\ p2}(\mathrm{w}_1) \equiv \delta_{p1}(\mathrm{w}_1) \cup \delta_{p2}(\mathrm{w}_1)$$

Proof:

To prove the expression, consider the right hand side - RHS, i.e.,

$\delta_{p1}(\mathrm{w}_1) \cup \delta_{p2}(\mathrm{w}_1) = \{u \in \delta_{p1}(\mathrm{w}_1)\ or\ u \in \delta_{p2}(\mathrm{w}_1)\}$ (using Definition 5.11)

Hence, *u* satisfies the predicates defined on either p_1 or p_2

$\therefore \delta_{p1\ OR\ p2}(\mathrm{w}_1)$ ■

Rule 5.4. Distribution of similarity selection over union

Similarity selection over union of two windows is equivalent to the union of similarity selection defined over individual windows (i.e., similarity selection operator is distributive over the union of two windows w_1 and w_2).

$$\delta_{p1}(\mathrm{w}_1 \cup \mathrm{w}_2) \equiv \delta_{p1}(\mathrm{w}_1) \cup \delta_{p1}(\mathrm{w}_2)$$

Proof:

To prove the expression, consider the left hand side - LHS i.e.,

$\delta_{p1}(\mathrm{w}_1 \cup \mathrm{w}_2) = \{u \in (\mathrm{w}_1 \cup \mathrm{w}_2) | u \text{ satisfies the predicate } p1\}$ (Definition 5.7)

$= \{u \in \mathrm{w}_1 \text{or } u \in \mathrm{w}_2 | u \text{ satisfies the predicate } p1\}$ (Definition 5.11)

$= \{u \in \delta_{p1}(\mathrm{w}_1)\ or\ u \in \delta_{p1}(\mathrm{w}_2)\}$

$= \delta_{p1}(\mathrm{w}_1) \cup \delta_{p1}(\mathrm{w}_2)$ ■

Rule 5.5. Distribution of similarity selection over intersection

Similarity selection over intersection of two windows is equivalent to the selection defined on the first window intersecting with the second window.

$$\delta_{p1}(\mathrm{w}_1 \cap \mathrm{w}_2) \equiv \delta_{p1}(\mathrm{w}_1) \cap \mathrm{w}_2$$

Proof:

To prove the expression, consider the left hand side - LHS i.e.,

$\delta_{p1}(\mathrm{w}_1 \cap \mathrm{w}_2) = \{u \in (\mathrm{w}_1 \cap \mathrm{w}_2) | u \text{ satisfies the predicate } p1\}$ (using Definition 5.7)

$= \{u | u \in \mathrm{w}_1, \exists x_i \in \delta_{A\ \theta\ u}(\mathrm{w}_2) \text{ and } u \text{ satisfies the predicate } p1\}$ (using Definition 5.9)

Notice, by the definition of intersection operator there is at-least one element in w_2 similar to u.

Hence, $\delta_{p1}(\mathrm{w}_2)$ returns all elements similar to u.

$= u \in \delta_{p1}(\mathrm{w}_1)\ and\ u \in \mathrm{w}_2$ (as u is similar to x_i)

$= \delta_{p1}(\mathrm{w}_1) \cap \mathrm{w}_2$ ■

Rule 5.6. Distribution of selection over difference

Selection operation is distributive over the difference of two windows w_1 and w_2.

$$\delta_{p1}(\mathrm{w}_1 \backslash \mathrm{w}_2) \equiv \delta_{p1}(\mathrm{w}_1) \backslash \mathrm{w}_2$$

Proof:

To prove the expression, consider the left hand side - LHS, i.e.,

$\delta_{p1}(\mathrm{w}_1 \backslash w2) = \{u \in (\mathrm{w}_1 \backslash \mathrm{w}_2) | u$ satisfies the predicate $p1\}$ (Using Definition 5.7)

$= \{u \in \mathrm{w}_1 | \nexists xi \in \delta_{p1}(\mathrm{w}_2)\ \ u$ satisfies the predicate $p1\}$, similarity selection operator returns all elements similar to *u*.

$= u \in \delta_{p1}(\mathrm{w}_1)\ and\ \nexists x_i\ \in \mathrm{w}_2$ / x_i is similar to u

$= u \in (\delta_{p1}(\mathrm{w}_1) \backslash \mathrm{w}_2)$

$= \delta_{p1}(\mathrm{w}_1) \backslash \mathrm{w}_2$ ■

Rule 5.7. Distribution of similarity selection over join

Similarity selection defined over similarity join is equivalent to joining the result of similarity selection over individual windows (i.e. similarity selection is distributive over similarity join of two windows w_1 and w_2).

Notice that the predicate *p1* is defined without explicit distinction between referring and referred elements (i.e., the predicate is not defined to refers to different sources).

$$\delta_{p1}(\mathrm{w}_1 \overset{\bowtie}{p'} \mathrm{w}_2) \equiv \delta_{p1}(\mathrm{w}_1) \overset{\bowtie}{p'} \delta_{p1}(\mathrm{w}_2)$$

Proof:

To prove the expression, consider the left hand side - LHS i.e.,

$\delta_{p1}\left(\mathrm{w}_1 \overset{\bowtie}{p'} \mathrm{w}_2\right) = \{u \in (\mathrm{w}_1 \overset{\bowtie}{p'} \mathrm{w}_2) | u \text{ satisfies the predicate } p1\}$ (Definition 5.7)

$u \in (\mathrm{w}_1 \overset{\bowtie}{p'} \mathrm{w}_2) \equiv \{u = (u', x_i) | u' \in w_1, \exists x_i \in \delta_{p'}(w_2)\}$ u satisfies the predicate *p1* (using Definition 5.12) and

where $u' and\ x_i$ are the referring and referred component of the element u.

Using Definition 5.7, the similarity selection returns all elements x_i similar to u'.

u contains x_i as children and satisfies the predicate *p1* which has to be defined on an element that exists in u' and x_i.

$= u' \in \delta_{p1}(\mathrm{w}_1)$ AND $x_i \in \delta_{p1}(\mathrm{w}_2)$

$= \delta_{p1}(\mathrm{w}_1) \bowtie \delta_{p1}(\mathrm{w}_2)$ ■

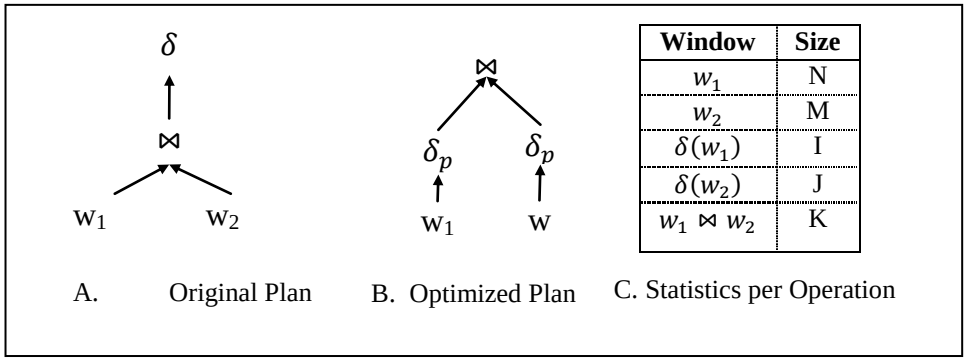
5.4.2 Heuristic base query optimization

Now, let us list the steps in the heuristic-based RSS query optimization strategies that make use of the equivalence rules identified above to transform a given algebraic query to less costly equivalent one:

1. Decompose selection into a cascade of selection using Rule 5.1 to Rule 5.3. This helps to push down the selection operation in the query plan.
2. Push down selection operation as much as possible using Rule 5.4 to Rule 5.7
3. Apply the most restrictive selections first using Rule 5.1 to Rule 5.3. One of the naïve selection criteria is the size/number of elements in a window.

Recalling the Example 1.8 in our motivating Scenario 3, the query optimizer may generate two query plans as shown in Figure 5.14. Executing these two plans provides similar result but with different costs.

Given the statistical information shown in Figure 5.14.C and assuming that an operation is done in a period; the original plan requires $N \times M$ periods for join and K periods for selection. However, the optimized plan requires time dependent on the size of each window (N and M) and the number of items resulted from selection operation (I and J) which are the input for join. Thus, total cost of the optimized query is $N + M + I \times J$ periods which is lesser than $N \times M + k$.



Window	Size
w_1	N
w_2	M
$\delta(w_1)$	I
$\delta(w_2)$	J
$w_1 \bowtie w_2$	K

Figure 5.14: Query optimization strategy

Table 5.4 shows the timing and number of items returned after executing a query representing Example 1.8 before and after the query optimization. In this experiment, we intentionally vary the number of elements in each window; we record the time and the number of elements after each operation. We observer that query optmization doesn't change the number of element in the result. However, comparing duration part of column 4 (i.e., time needed to complete the original query $\delta_p(w_1 \bowtie w_2)$) and duration part of the last column (i.e., time needed to complete optimized query $\delta_p(w_1) \bowtie \delta_p(w_2)$) of the table, one can clearly see that joining the result of selection is much cheaper than selecting the result of join.

Table 5.4: Query statistics per operation

w_1	w_2	$w_1 \bowtie w_2$		$\delta_p(w_1 \bowtie w_2)$		$\delta_p(w_1)$		$\delta_p(w_2)$		$\delta_p(w_1) \bowtie \delta_p(w_2)$	
#	#	#	Duration	#	Duration	#	Duration	#	Duration	#	Duration
10	10	2	2.17	0	2.26	1	0.20	1	0.26	0	0.48
10	100	10	30.89	1	31.13	1	0.24	3	2.97	1	3.28
10	200	10	51.37	1	51.63	1	0.20	9	5.16	1	5.50
10	300	10	80.54	1	80.75	1	0.20	9	5.16	1	5.50
10	400	10	115.10	1	115.36	1	0.21	10	17.64	1	18.24
10	500	10	176.26	1	176.57	1	0.18	12	17.78	1	18.25
10	600	10	218.15	1	218.63	1	0.22	16	19.48	1	20.06
10	700	10	229.29	1	229.54	1	0.18	17	26.45	1	27.03
35	100	28	98.93	1	99.84	1	0.68	3	2.37	1	3.12
35	200	30	217.42	1	218.33	1	0.72	9	5.54	1	6.39
35	400	35	444.29	1	445.54	1	0.67	10	11.76	1	12.62
35	700	35	722.37	1	723.92	1	0.75	17	24.54	1	25.60

5.4.3 Discussion

In Section 5.4.1, we have presented and proved set of equivalent rules dedicated to the use of selection over other operators. Here, we discuss three issues related to symmetric merging, pushing down selection over merging operator and query optimizer.

Symmetric merging

On one hand, our merging operator is not symmetric (Property 5.7). Nevertheless in the database community, the symmetricity of an operator is basic and crucial to facilitate query rewriting, simplification and query optimization.

The issue is to know how, given two windows w_1 and w_2, we can define a symmetric merging operator, denoted for instance as $w_1 \overleftrightarrow{\oplus} w_2$, and formalized as:

$$w_1 \overleftrightarrow{\oplus} w_2 = w_1 \oplus w_2 \; \theta \; w_2 \oplus w_1 \tag{5.13}$$

where: θ is an operator to be used in defining the symmetric merge (might be ⋓).

However, the validity of Equation (5.13) depends on the actions associated to the user merge rules (for instance, relationships).

To illustrate this, let's consider the difference operator which is not symmetric (c.f. Property 5.2). Using the Property 5.6, the merge operator generalizes the difference operator. However, the symmetric merge as the difference operator in Equation (5.13) (when $\theta = \Cup$) returns the list of elements in W_1 exclusive or W_2 which is in contradiction to the definition of difference as the result contains elements in only in W_1 or only W_2. Hence, we can't conclude the validity of Equation (5.13) and it need further study.

In spite of the doubt on the generic symmetric merging operator, we can define its specific cases (using $\theta = \Cup$), symmetric join and symmetric intersection operators, which support Equation (5.13) using the following merging rules:

$$\bowtie \equiv \bigoplus^{\text{Merging\_rule ::=}\{\langle equal, KeepBoth\rangle,\ \langle Else, \text{IgnoreBoth}\rangle\}}$$

and

$$\cap \equiv \bigoplus^{\text{Merging\_rule ::=}\{\langle equal, Keepfirst\rangle,\ \langle else, IgnoreBoth\rangle\}}$$

It is to be recalled that, we defined the symmetric join in Section 5.3.5.

Pushing down selection over merging operator

On the other hand, pushing down selection operation before any other operation is one of the heuristic-based strategies provided in Section 5.4. Pushing down the similarity selection operator over the merging of two windows is not necessarily equivalent to merging the result of selection over individual windows.

Using the Property 5.6, the merging operator generalizes the *Union*, *Intersection*, *Difference* and *Join* operators. Hence, the *selection* operator distributes over the *Merge* operator using the equivalence Rule 5.4 to Rule 5.7.

However, it is to be noted that the result of merging two windows is dependent on the merging rules provided by the user. So, even if the same merging rule is applied to both side of the expression, there is a chance that an element in window to be removed either with the selection or the merging rule. This is demonstrated in the next example.

Example 5.13: Assume that a journalist wants to select all merged news returned from Example 5.12 having the description element containing "Golden Brown". The Example 5.12, returns all merged news published between 5 and 7 o'clock on December 3, 2009 by CNN and BBC considering similar title element and using the merging rules: Keep the latest of identical news, keep the detailed news in case of inclusion; otherwise, keep both news items as children of VR.). i.e.,

$$\text{Merging\_rule} ::= \{\langle equal, KeepBoth\rangle, \langle Includes, \text{KeepFirst}\rangle, \langle Else, \text{KeepFirst}\rangle\}$$

The query $\delta_{description\ contains\ "Golden\ Brown"}(w_1 \oplus^{Merging\_rule} w_2)$, returns the latest of the equal news items (CNN2 and BBC3), i.e., BBC3.

However, pushing down the selection over the merge operator involves executing selection on each window and merge the result. It is denoted as:

$$\delta_{\text{description contains "Golden Brown"}}(w_1) \oplus^{\text{Merging\_rule}} \delta_{\text{description contains "Golden Brown"}}(w_2)$$

The selection query $\delta_{description\ contains\ "Golden\ Brown"}(w_1)$ returns empty window as none of the elements have a description containing "Golden Brown". Whereas, the selection query $\delta_{description\ contains\ "Golden\ Brown"}(w_2)$ returns one item,BBC3. The merging of empty window and the window having *BBC3* returns empty window.

Therefore, this counter example demonstrates a specific case in which pushing down selection over merging is not always `True` unless the merge operator is acting specifically the *set membership* or *Join*, i.e.,

$$\delta_p(w_1 \oplus w_2) \not\equiv \delta_p(w_1) \oplus \delta_p(w_2)$$

Issues in query optimizer

The design of query optimizer in feed processing has to consider a number of issues in addition to the query rewriting. Among other issues, while evaluating the binary operator (e.g., *Merge*) the query processor and the optimizer have to decide on what to do if one of the windows is empty. There are two options: either to wait until the first element in the window arrives (case of Lazy query processing) or return an empty result – eager query processing (similar to propagating *NULL* value in relational SQL).

The query processing addressed in this thesis did not give attention to the type of window, the expiration of the content of the windows and arrival rate. The query processor evaluates the query eagerly as soon as the window arrives. The issue of continuous query can be handled in two ways: applying the eager query evaluation together with eager expiration and re-evaluation strategy; or lazy query evaluation together with lazy content expiration and revaluation at a given time *t*. It is to be noted that the windows arrival rate determines the load on the query processor and the efficacy of the query result. We believe that the feed stream query processor and optimizer have to take into consideration the following issues:

- window buffer size
- windows arrival rate and query revaluation strategy
- the number of items arriving
- when to use semantic information and identify the maximum neighborhood distance that provide the best value
- and the environment of the machine (e.g., processing speed)

5.5 Summary

In this chapter, we have presented window-based and semantic-ware RSS algebra. Our operators are categorized into three: extraction, set member ships and high level *Merge*

operator. The *extraction* operator is dedicated to extract elements/items satisfying a given condition and includes: the unary *selection* and *TopK*, and the binary *Join*. The *Set Membership* operators are capable to identify *Union*, *Intersection* and *Difference* of two windows. The *Merge* is a high-level and high order operator that generalizes the *Set memberships* and *Join* operators. Each operator accepts a set of windows as inputs together with specific parameter set that contains threshold value, semantic flag, item connector and/or a set of merging rules. A True semantic flag leads to enhancing the selection condition (particularity the tag name, operator and content) using label, operator and value Knowledge Bases (cf. two categorizes of Knowledge Base in Section 3.2.2).

To summarize: in this chapter, we have presented RSS algebra that takes into consideration the two specific properties of RSS feeds, heterogeneity and dynamicity.

The contribution of this chapter can be summarized as follows:

1. We provided a set of semantic-aware and window-based similarity operators.
2. We demonstrated the user query in XQuery format together with our algebraic operator.
3. We defined a novel operator *Merge* that generalize the binary *Join* and *set membership* operators.
4. We investigated the property of the operators and provided a set of equivalent rules. We also proved their validities.
5. We showed that *Select* and *Merge* are the two minimal operator in RSS context
6. We have published partial result of this chapter in an international conference (GETAHUN, F and Chbeir, R, 2010) and also submitted to the International Journal of Information Sciences (GETAHUN, F. and Chbeir, R., 2010)

CHAPTER 6
EASY RSS MANAGER AND EXPERIMENTATION

Abstract:

Easy RSS Manager is a desktop prototype designed using Microsoft C# having a semantic-aware RSS Reader, and semantic-aware and window-based RSS query components. It is designed to validate, demonstrate and test the practicability of the different proposals of this research. In particular, we test the timing complexity and the relevance of our approaches using both real and syntactic datasets.

6.1 Introduction

One objective of this study is to design an easy to use, adaptable and customizable system to validate the different proposals made in this thesis. To do so, we designed a prototype system called Easy RSS Manager (*EasyRSSManager*) and developed with Microsoft C# programming language. We stored a user personal information, a set of associated merging rules, a set of system parameters, and two Knowledge Bases in a light weight relational database – MySQL.

EasyRSSManager allows a user to

1. provide her notion of merging news items using the merging editors
2. identify semantic neighborhood of a word/concept and compute the similarity between a pair of words
3. formulate both syntactic- and semantic- aware queries and visualize the result

In addition, *EasyRSSManager* is used as a test platform in conducting a set of experiments to:

1) evaluate the timing analysis of our RSS relatedness and query rewriting approaches,
2) measure the quality/relevance of
   a) our enclosure similarity measure,
   b) our relatedness measure in identifying topological relationships, grouping related news items and consequently performing RSS merging,
   c) using semantic information in querying RSS news items.

All experiments were carried out on an Intel Core Centrino Duo Processor machine (with 1.73 GHz processing speed and 1GB of RAM).

The rest of this chapter is organized as follows. In Section 6.1, we give a description of the different components and system architecture of *EasyRSSManager*. In Section 6.3, we present a logical database design of *EasyRSSManager*. Section 6.4 presents sample

graphical interfaces used to manager user favorite feeds, merging rule editor, and feed query interface. In Section 6.5, we provide the experimentation that validates the enclosure similarity measure, RSS relatedness algorithm, RSS merger and RSS query approaches and in Section 6.6, we conclude the chapter with summary information.

6.2 Architecture of *EasyRSSManager*

EasyRSSManager is a desktop application designed to perform two tasks:

1) extend the Google-Reader with the semantic-based measures and the merging strategy introduced in Chapter 3 and Chapter 4 respectively; and
2) facilitate the formulation of RSS feeds query using the set of algebraic operators proposed in chapter 5

The general system components of *EasyRSSManager* are shown in Figure 6.1. It encompasses six main interacting components:

1) The *database* component: it manages
   a) two Knowledge Bases containing (1) a value Knowledge Base, WordNet 2.1 (WORDNET 2.1, 2005) lexical taxonomy, exploited in evaluating text content relatedness, and (2) a label knowledge base used in evaluating element label relatedness,
   b) user profiles and merging preferences. When the system starts for the first time, the user would provide an initial profile that includes a list of tag names that would be used as item connector.

   The database used in *EasyRSSManager* is detailed in Section 6.3.
2) The *Google Reader* is an online RSS feed aggregator for managing (add, edit, remove) feeds. In *EasyRSSManager*, we use Google Reader API to access the atom feeds registered by users.

3) The *Relatedness Engine* component is responsible to identify the similarity and relationship between a pair of texts, simple elements or items. It measures similarity and relation automatically after

   a) stemming text values using Porters' algorithm(PORTER, M. F., 1980),

   b) generating a vector for each text,

   c) computing the similarity between words/concepts using the *Semantic Measure* component (which implements our enclosure similarity measure), and

   d) computing the relatedness and relationships at different level of granularity, i.e., text, label, simple element, and item (complex element).

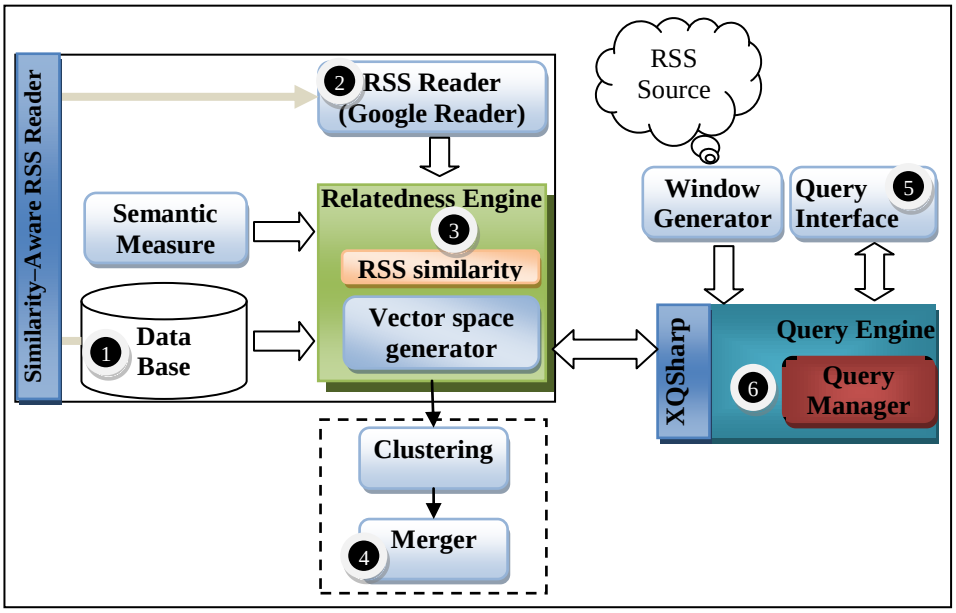


Figure 6.1: EasyRSSManager Architecture

4) The *Merging module* put together a set of news items in the same cluster according to the current user's merging rules and preferences. Notice that the put cluster is generated using the *Clustering module*.

5) The *Query Interface (Query Input and Output)* component allows a user to formulate

queries. It is a Graphical User Interface (GUI) that facilitates the formulation of RSS query by selecting an operator with its associated parameters and specifying the value to be searched. It also allows visualizing the result of the query. A sample query interface is discussed in Section 6.4.

6) The *Query Manager* component accepts the user query, formulated with the help of the query input and output component. The inputs (i.e., windows) are generated with the help of *Window Generator* component which returns list of elements within timestamp based boundary conditions. It parses, validates, and executes the query using XQSharp[43](CBCL, 2009) after communicating with the *similarity engine* (when similarity functions or operators are involved in the query expression).

6.3 Database schema

The logical database in Figure 6.2 represents entities referring to user's personal information, merging rules, semantic knowledge and relationship between the entities. The following are the list of tables extracted from the logical model:

- $User(\underline{UserId}, FirstName, LastName, Email, PassWord)$: contains the basic information about a user uniquely identified with UserID.
- *Feed-Sources*$(\underline{SourceID}, URL, Title, Description)$: represents RSS feed providers information such as feed address (URL), title and description. The content of this table is updated using a dedicated web crawler.
- *User-Sources*(UserId, SourceID): represents the association between a user and the registered feed sources. It contains the primary key of the participating tables as foreign keys.

[43] XQSharp is a fast, schema-aware XML Processor for the Microsoft .NET Framework versions 2.0 or later. It builds upon the classes in the System.Xml namespace to provide up-to-date standards compliant implementations of XQuery 1.0 and XPath 2.0 native .NET query processor.

- *Rule-Type*(Rule-Type-ID, Description): stores the merging rule categories. The description column stores the value simple or item.

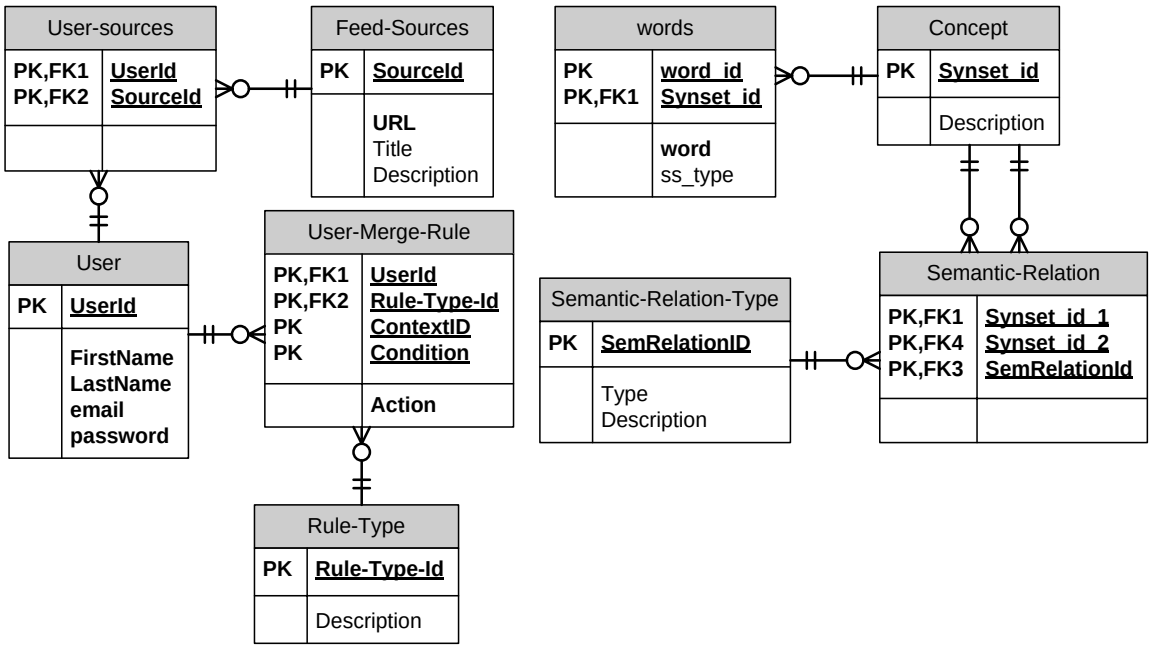


Figure 6.2: Sample logical database model of EasyRSSManager

- *User-Merging-Rule*(UserId, ContextId, Condition, Action, Rule-Type-ID): represents the user context merging rule. The table associates a user identified by UserID and having the context id (ContextID) with a set of merging rules. Each merging rule has a condition and action and belongs to a particular rule type (Rule_Type_ID).
- *Concept*(Synset_ID, description): represents concepts in Knowledge Base (i.e., collection of words having the same meanings). Synset_ID is a unique identifier that identifies a concept
- *Words*(word_id, Synset_ID, word, ss_type): represents a word together with the concept or synset it belongs to. SS_type indicates the type of word (such as verb, noun).

- *Semantic-Relation-Type*(SemRelationID, Type, Description): represents the different semantic relationship discussed in Chapter 2. Each relationship is uniquely identified by SemRelationID.
- *Semantic-Relation*(Synset_Id_1, Synset_Id-2, SemRelationID): represents the semantic-relationship existing between two concepts identified with corresponding synset id's.

6.4 User interfaces

We develop a set of Graphical User Interfaces to facilitate the interaction between a user and the *EasyRSSManager*. These interfaces include login, word similarity computation, user profile, merging rule editor and query interface. Here, we present the key interfaces.

6.4.1 Word similarity computation

The interface in Figure 6.3 allows a user to identify the global semantic neighborhood of a concept and to compute the enclosure similarity between two words/terms or concepts.

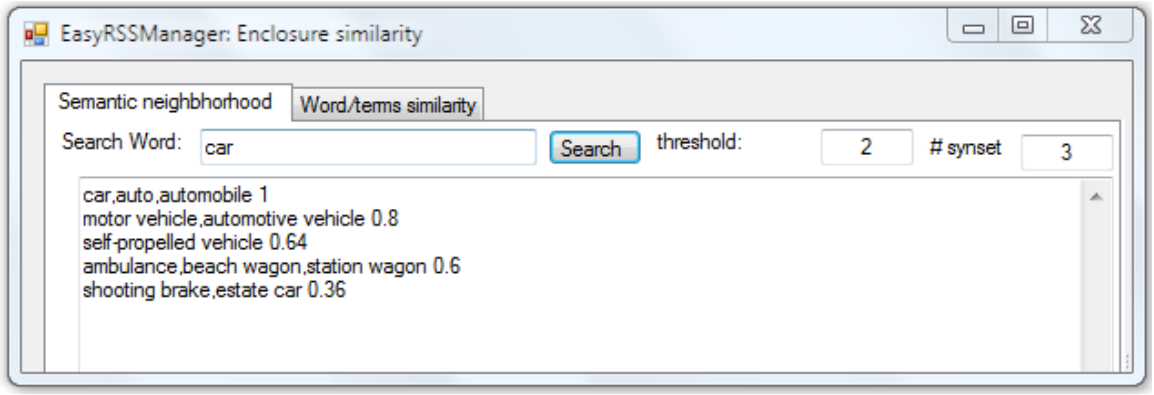


Figure 6.3: Screenshot of global semantic neighborhood generation page

Semantic Neighborhood page:

This page allows a user to identify the global semantic neighborhood of a concept within a given threshold and having a given number of words per concept (i.e., #synet). For example, Figure 6.3 shows the global semantic neighborhood of *car* within a threshold of 2 and having a maximum of 3 words per a concept.

Words/terms similarity page:

This page is used to compute the enclosure similarity between two words/terms/concepts related within a given threshold and having a given number of words per a concept (i.e., #synet). For example, Figure 6.4 shows the enclosure similarity between *car* and *auto* within a threshold of 2 (i.e., the maximum path length between a pair of concepts) and having a maximum of 3 words per a concept.

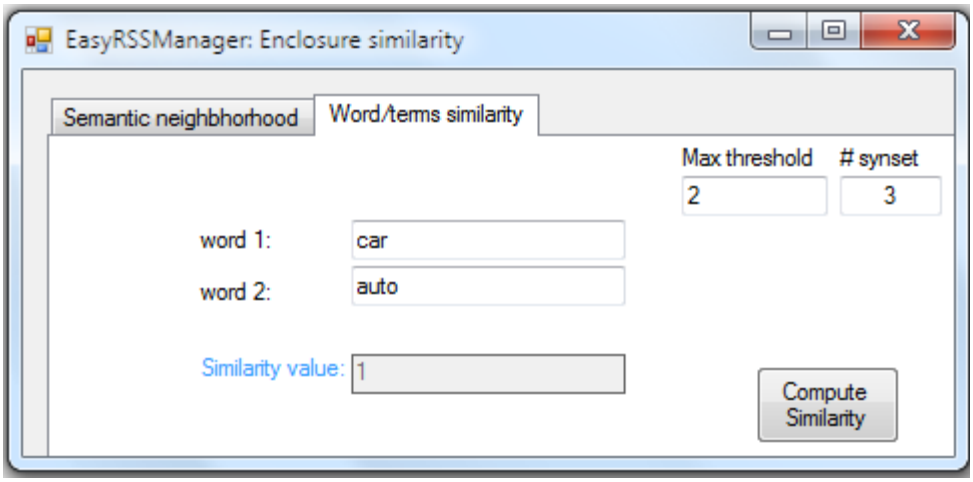


Figure 6.4: Screenshot of enclosure similarity computation page

6.4.2 User profile editor

The user profile editor interface shown in Figure 6.5 is composed of two pages. It enables to register and edit personal information together with feed preferences.

The user information page allows a user to register and edit the basic personal information. The use of gmail email address/login-name as username allows the user to imports feeds registered in Google Reader.

The feeds preference page allows to register the URL (feed address) and title (any text) of the RSS feed provider.

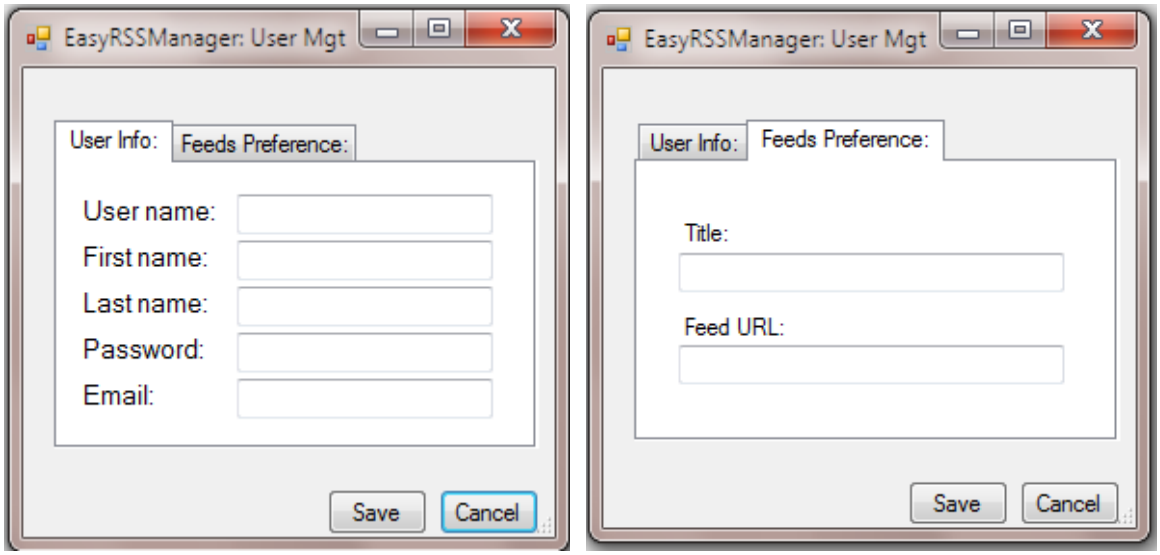


Figure 6.5: Screenshot of user profile editor

6.4.3 Merge-rule editor

The rule editor shown in Figure 6.6 allows a user to register and edit out-put type (Step 1), items merging rule (Step 2) and simple elements merging rules (Step 3).
The condition part of the rule is restricted to a relationship that might exist between items and/or simple elements. The grid shows the list of rules registered by the current use.

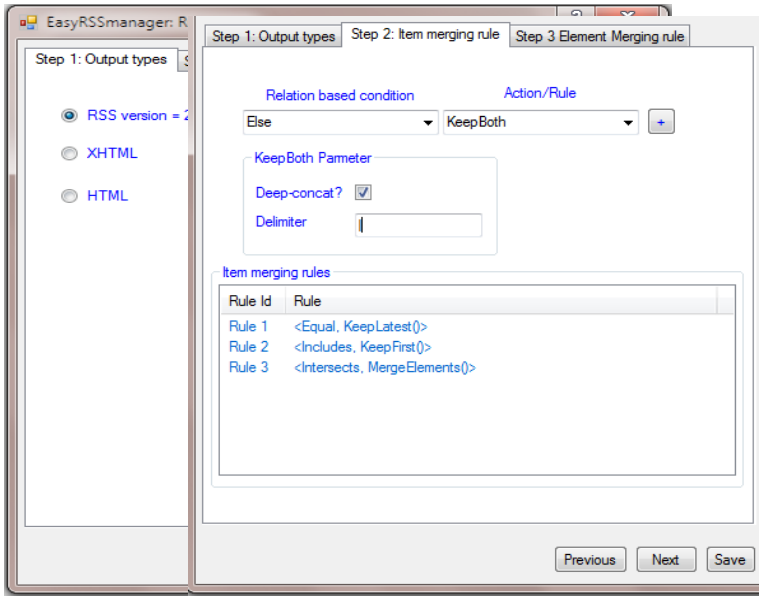


Figure 6.6: Snapshot of merging rules editor

6.4.4 RSS Query interface

The RSS query interface allows a user to search for RSS news items in time-aware windows. Querying RSS streams is done in two steps:

1) defining the boundaries of the window (step 1) and,
2) specifying the query predicate (step 2).

These two steps are handled with window definition and parameter page of the interface.

Window definition

The interface shown in Figure 6.7 allows a user to associate RSS source with window boundary information. The boundaries are two Date and Time value that specify the start and ending condition of the window to be generated. Here, a user chooses an RSS feed source and defines the boundary associated to it. The `getWindow` in the window generator component generates a window containing the list of elements extracted from the chosen RSS feed. The window is used later by the query processor. It is to be recalled

that some of our operators are order dependent hence the order in which the user selects the RSS sources determines the result.

Specify predicate:

The interface shown in Figure 6.8 allows a user to specify query predicate information. The attribute and operator component of a predicate can be easily selected with a mouse clicks. In addition, this interface eases associating the appropriate parameters of the selected operator, presenting the XQuery expression equivalent to the user action, and displaying the result of the RSS query. It also allows saving the already formulated query as a feed source for later queries (similar to a materialized view in relational database).

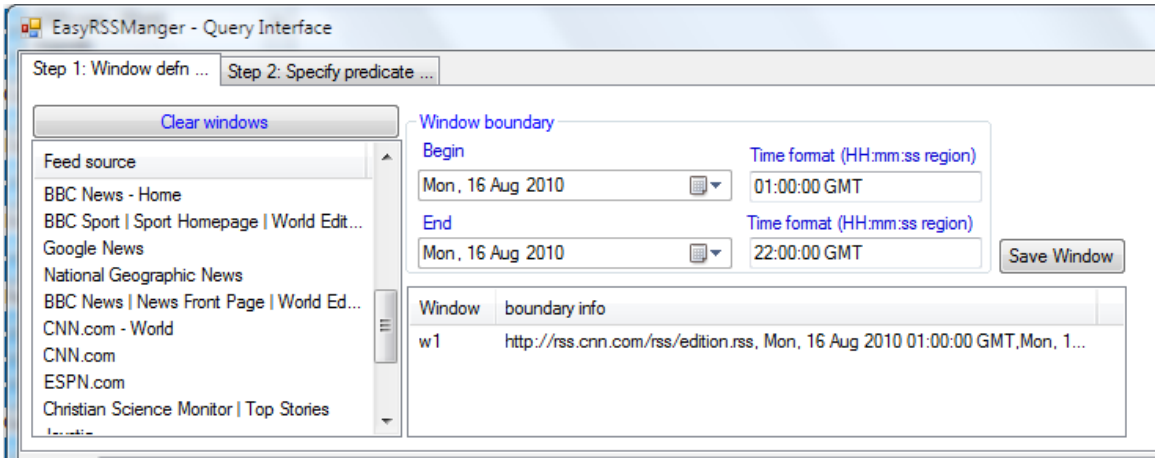


Figure 6.7: Snapshot of a window definition interface

Example 6.1: topk query Bob is interested to get the first 2 news items from CNN published between 1 o'clock and 22 o'clock on Monday 16 August 2010 having semantically similar title content as "flood Pakistan".

This query is formulated using the interface shown in Figure 6.8. The selected operator is TopK and the attributed is title. Notice that item connector text box is disabled as the query is not based on an item. Figure 6.8 shows both the query and the final result.

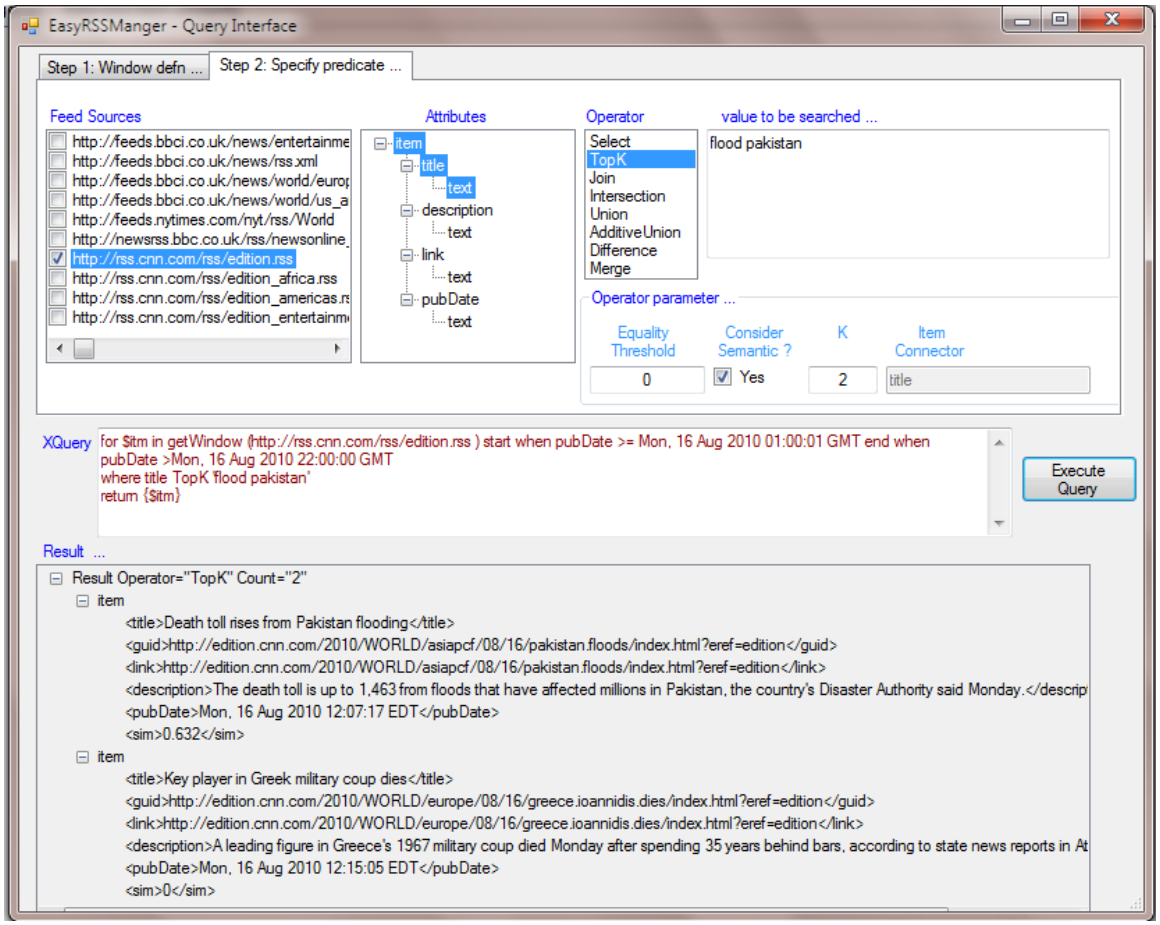


Figure 6.8: Snapshot of query input and output

In the next section, we use the *EasyRSSManager* as a test platform in conducting a set of experiments.

6.5 EXPERIMENTATION

In this section, we present the set of experiments conducted to validate our approach. The section is organized as follows: in Section 6.5.1, we present the dataset used in these experiments. In Section 6.5.2, we present timing analysis. Section 6.5.3 presents the relevance measure as applied to enclosure similarity measure, RSS relatedness, RAGALL, merger and semantic query.

6.5.1 Dataset

In conducting the set of experiments, we used both syntactic[44] and real dataset.

- Syntactic dataset: we developed a C# prototype that generates RSS document that conforms to RSS 2.0 specification. The prototype accepts the following parameters:
  - the number of news items to be generated
  - the maximum number of concepts per news item
  - the number of disjoint clusters
  - the number news items per cluster related with Equality, Inclusion and Overlapping relationship
- Real dataset: we have used two groups of real datasets
  1) Group 1: It contains 158 RSS news items extracted from 5 well known news providers (CNN, BBC, USAToday, L.A. Times and Reuters). We manually grouped the news into 6 predefined clusters: US Presidential elections 08, Middle-east, Mumbai-attacks, space-technology, oil, and football. However, we did not identify the relationships that could exist between news items.

[44] syntactic dataset refers to a dataset generated using a specialized program

2) Group 2: It contains Antinio Gulli's corpus of news articles gathered from more than 2000 news sources using academic news search engine, ComeToMyHead, developed by Antonio Gulli (GULLI, A., 2004). We have extracted 567 news articles published as Top News of CNN, BBC, Newsweek, The Washington Post, Reuters, Guardian, and Time. We group the news into 6 clusters. Table 6.1 shows the clusters and number of news related with equality, inclusion and overlap relations.

Table 6.1: Manual Clusters and distribution of relationships

Cluster Name	# Equal	# Including	# Overlapping	Total
Mortgage	100	69	0	169
Afghan	17	5	10	32
Bin-Laden	13	4	1	18
Arafat	30	6	19	55
Terrorism	27	19	78	124
USA-Election	60	9	100	169

6.5.2 Timing analysis

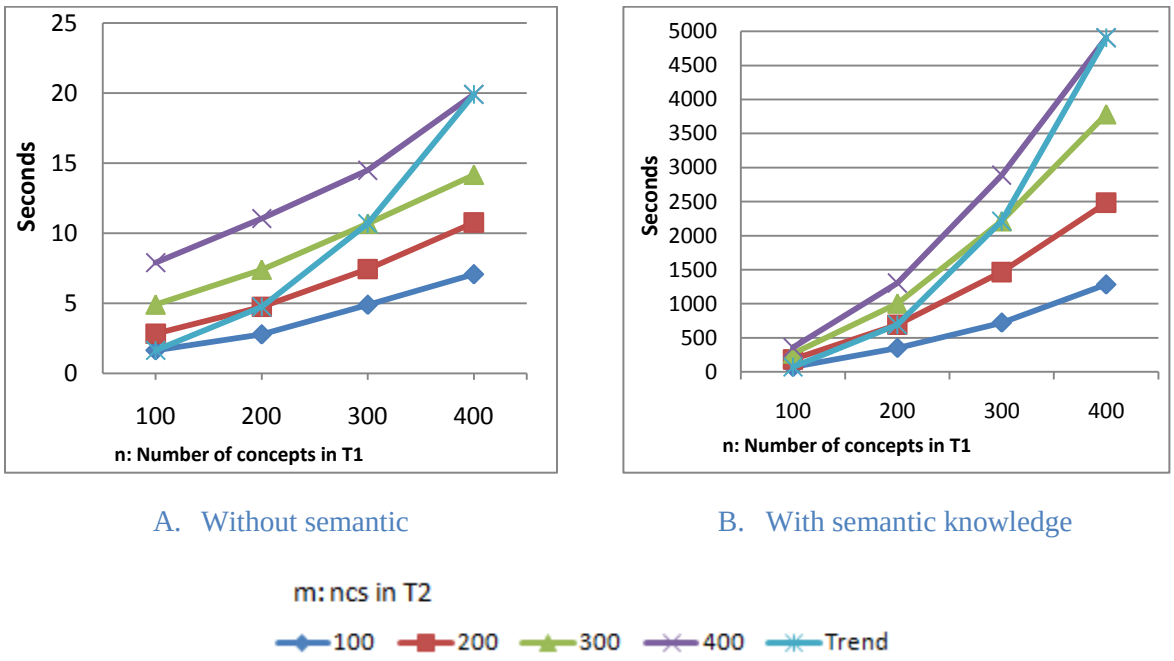
In this section, we present the timing analysis of our RSS relatedness measure and query rewriting approach. We implemented the three relatedness algorithms – texts, simple elements and items- and verified the theoretical computational complexity of the relatedness algorithm, and compared the efficiency against two existing similarity approaches.

6.5.2.1 Timing analysis and efficiency of RSS relatedness measure

We experimentally tested the time complexity of our RSS relatedness algorithm, w.r.t. the sizes of input texts t_1 and t_2 i.e., number of concept sets (n and m) and value Knowledge Base information (number of concepts - n_c and depth - d). Note that we used

two disjoint synthetic news items having various numbers of concepts (between 100 and 400) and relationship computation is not included here as its impact is negligible.

On one hand, we can quickly observe the polynomial nature of the timing result shown in Figure 6.9, demonstrating the polynomial dependency on input text size (Figure 6.9.A) and Knowledge Base information (Figure 6.9.B). The *x* axis represents the number of concepts in a concept set and the *y* axis shows the number of seconds consumed to compute the relatedness value.



A. Without semantic

B. With semantic knowledge

Figure 6.9: Timing analysis on various concept set sizes in t_1, t_2 (n, m)

In Figure 6.9, we also show the effect of varying number of concepts in synsets. Figure 6.9.A shows the timing result without considering Knowledge Base information while varying the size of the input texts. Increasing the number of concept sets increases the timing in a quadratic fashion (i.e., the dot line shows the growth rate trend of the algorithm). Figure 6.9.B represents the timing result considering a fixed size Knowledge Base (having 100 concepts with a maximum depth of 8). The time needed to compute the

relatedness between items increases drastically (compared to the result shown in Figure 6.9.A) and in a polynomial fashion. The Figure 6.9.B also shows the cost of using semantic information in system efficiency.

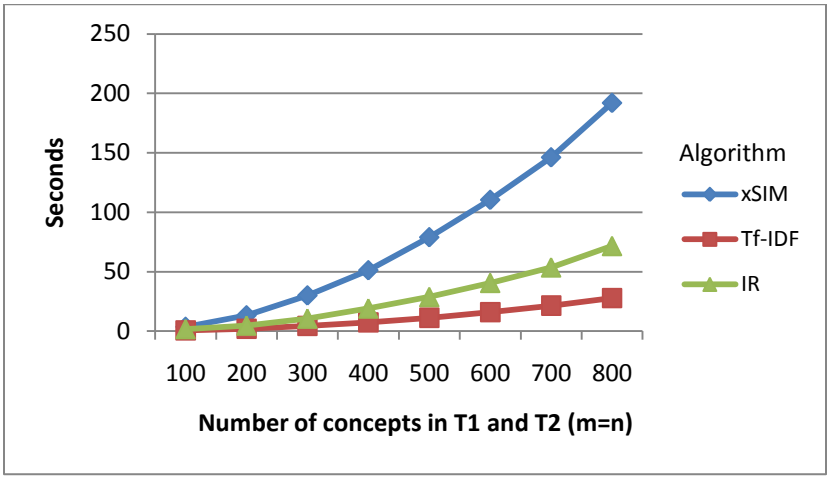


Figure 6.10: Timing result obtained using three algorithms: xSim, TF-IDF and IR

On the other hand, we wanted to compare the efficiency of our algorithm with similar existing ones. As alternative algorithms, we chose *xSim* (KADE, A. M. and Heuser, C. A., 2008) and *TF-IDF*(MCGILL, M. J., 1983), the former being one of the most recent XML-dedicated similarity approaches in the literature, the latter underlining a more generic method for computing similarity and which could be utilized to compare RSS items. In all three algorithms (including ours- IR), computing relatedness between randomly generated synthetic news is performed without semantic relatedness assessment (as both *xSim* and *TF-IDF* do not consider semantic information) using cosine similarity. Figure 6.10 shows that our approach yields better timing results in comparison with *xSim*, but performs worse than *TF-IDF* due to the fact that *TD-IDF* does not consider the structure of RSS news items but only their concatenated contents. We believe that our measure IR performs better than xSim as xSim works in three steps (i.e., (i) identify document list (ii) compute content, tag name and path similarity values (iii) combine these similarity values), which add considerable time.

6.5.2.2 *Timing Analysis query rewriting*

One of the issues in query optimization is transforming a query into a less costly query plan. Figure 6.11 shows the timing when the user query in Example 1.8 is executed in windows having various elements. The X-axis represents the number of elements in window W_2. For each window, we vary the number of elements in w_1 and monitor the required time to complete both the original and optimized query. The graph show that the optimized query finishes instantaneously – i.e., pushing down the selection reduces the timing.

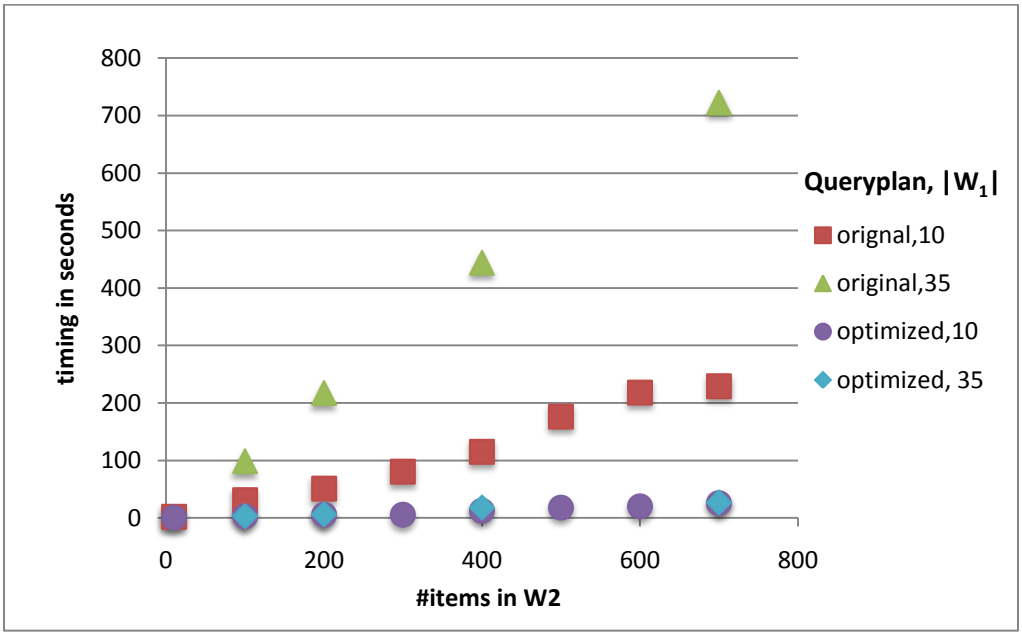


Figure 6.11: Timing analysis of pushing down selection over join

6.5.3 Relevance of our approaches

The relevance of an approach refers to the degree to which it is applicable or practical to handle the problem at hand. This can be done by comparing the approach against existing solutions or assessing the rating of humans or system users. One of the known methods to

compare different approaches on the same problem is to use the popular information retrieval metrics *precision* (PR) and *recall* (R) (MCGILL, M. J., 1983).

In this set of tests, we used hierarchical clustering method to measure the relevance of our approach by grouping together related/similar news. Checking clustering quality involves:

(i) the use of a set of original clusters that contains which RSS news item belongs to which clusters

(ii) mapping the discovered clusters to the original clusters.

Here, we exploit the *precision* (PR) and *recall* (R) (MCGILL, M. J., 1983) to check the relevance of the discovered clusters.

The precision PR measures the degree to which the identified cluster is exact to the original cluster. The recall R measures the percentage of relevant results identified i.e., the degree to which the identified clusters are relevant.

In addition, an *f-score* value is used to compare the accuracy of different clustering results based on the combined values of PR and R as these values are not discussed in isolation while measuring the relevance:

$$f - score = \frac{2 \times PR \times R}{(PR + R)} \tag{6.1}$$

In the following sub-sections, we measure the relevance of our approaches using both statistical and human rating methods.

6.5.3.1 Enclosure similarity measure

To evaluate the quality of our concept similarity measure, we used a human rated dataset organized by Miller and Charles (MILLER, G. and Charles, W., 1998). In their study, 38 undergraduate subjects are given 30 pairs of nouns and were asked to rate the similarity of meaning of each pair on scale from 0 (unrelated) to 4 (highly related). The average

rating of each pair represents a good estimate of how similar/related the two words/concepts are.

Recall that comparing two similarity measures is not easy and so far there do not exist a known standard that assist this comparison. In this experiment, we compare our measure against four measures that use WordNet as external Knowledge Base on Miller and Charles dataset:

1) simple edge counting (sim_{edge})
2) (WU, Z. and Palmer, M. S., 1994) Wu and Palmer edge counting method ($Sim_{Wu\ \&\ Palmer}$)
3) (LIN, D., 1998) information content based measure of Lin (sim_{Lin}) and
4) integrated similarity measure of Hong-Ming and Smith – Sim_{Hong} - (HONG-MINH, T. and Smith, D, 2008)

Hong-Ming & Smith combine the simple edge counting approach with information content base approach to measure word similarity. In this test, we used only 28 word pairs (c.f. Table 6.2) out of 30 as we found that the WordNet 2.1 taxonomy didn't properly classify some of the words such as (lad, wizard and monk, slave).

In Table 6.2, all similarity values are between 0 and 1. It is to be noted that, a similarity measure provides a higher value (e.g., the colored cells in Table 6.2 represent a maximum value among the other measures) is not a justification to conclude the relevance of the measure in identifying similarity. Rather, we believe that it is necessary to correlate the result of a measure against the rating of humans. The higher the correlation value it is most likely that the measure is capable in identifying the similarity at least within the provided dataset. For instance, the $Sim_{Wu\ \&\ Palmer}$ provides higher value for a pair of concept having smaller human rated similarity value. Noting the last row in the Table 6.2, the correlation values on the Miller-Charles data set show that our measure outperforms all the other method with a correlation value of 90%.

Table 6.2: Human and computer ratings of the Miller–Charles set of word pairs

#	*word1*	*word2*	Human Rating	Sim_{edge}	Sim_{Lin}	Sim_{Hong}	$Sim_{Wu\ \&\ Palmer}$	Enclosure Sim
1	noon	string	0.02	0.08	0.00	0.00	0.35	0.00
2	rooster	voyage	0.02	0.05	0.00	0.00	0.15	0.07
3	glass	magician	0.03	0.13	0.13	0.07	0.53	0.07
4	chord	smile	0.03	0.09	0.27	0.07	0.44	0.10
5	coast	forest	0.11	0.14	0.13	0.38	0.62	0.27
6	shore	woodland	0.16	0.17	0.14	0.27	0.67	0.27
7	forest	graveyard	0.21	0.10	0.08	0.13	0.50	0.18
8	coast	hill	0.22	0.20	0.71	0.71	0.71	0.27
9	food	rooster	0.22	0.07	0.10	0.26	0.29	0.12
10	cemetery	woodland	0.24	0.10	0.08	0.07	0.50	0.27
11	monk	oracle	0.28	0.13	0.23	0.26	0.59	0.65
12	journey	car	0.29	0.07	0.00	0.00	0.19	0.05
13	lad	brother	0.42	0.20	0.29	0.27	0.71	0.72
14	crane	implement	0.42	0.20	0.00	0.80	0.78	0.27
15	brother	monk	0.71	0.50	0.25	0.54	0.96	0.76
16	tool	implement	0.74	0.50	0.92	0.73	0.94	1.00
17	bird	crane	0.74	0.25	0.00	0.85	0.88	0.81
18	bird	cock	0.76	0.50	0.80	0.85	0.96	0.94
19	food	fruit	0.77	0.13	0.13	0.73	0.47	0.33
20	furnace	stove	0.78	0.13	0.22	0.32	0.57	0.57
21	midday	noon	0.86	1.00	1.00	1.00	1.00	1.00
22	magician	wizard	0.88	1.00	1.00	1.00	1.00	1.00
23	asylum	madhouse	0.90	0.50	0.98	0.90	0.96	0.96
24	coast	shore	0.93	0.50	0.97	1.00	0.92	1.00
25	boy	lad	0.94	0.50	0.82	0.87	0.93	0.94
26	gem	jewel	0.96	1.00	1.00	1.00	1.00	1.00
27	journey	voyage	0.96	0.50	0.69	0.92	0.96	0.93
28	car	automobile	0.98	1.00	1.00	1.00	1.00	1.00
	Correlation		**1.00**	**0.77**	**0.74**	**0.88**	**0.80**	**0.90**

In Figure 6.12, we present the correlation between the human rating and the three measures that have more than 80% correlation value. The graph shows the human rating values and the value of the other three measures over the Miller and Charles datasets. The human rating values are sorted in increasing value. The graph also shows a close correlation between our measure (*Enclosure-Sim*) and the human rating in comparison with the Wu & Palmer (*Sim-Wu & Palmer*) and Hong-Ming and Smith (*Sim-Hong*).

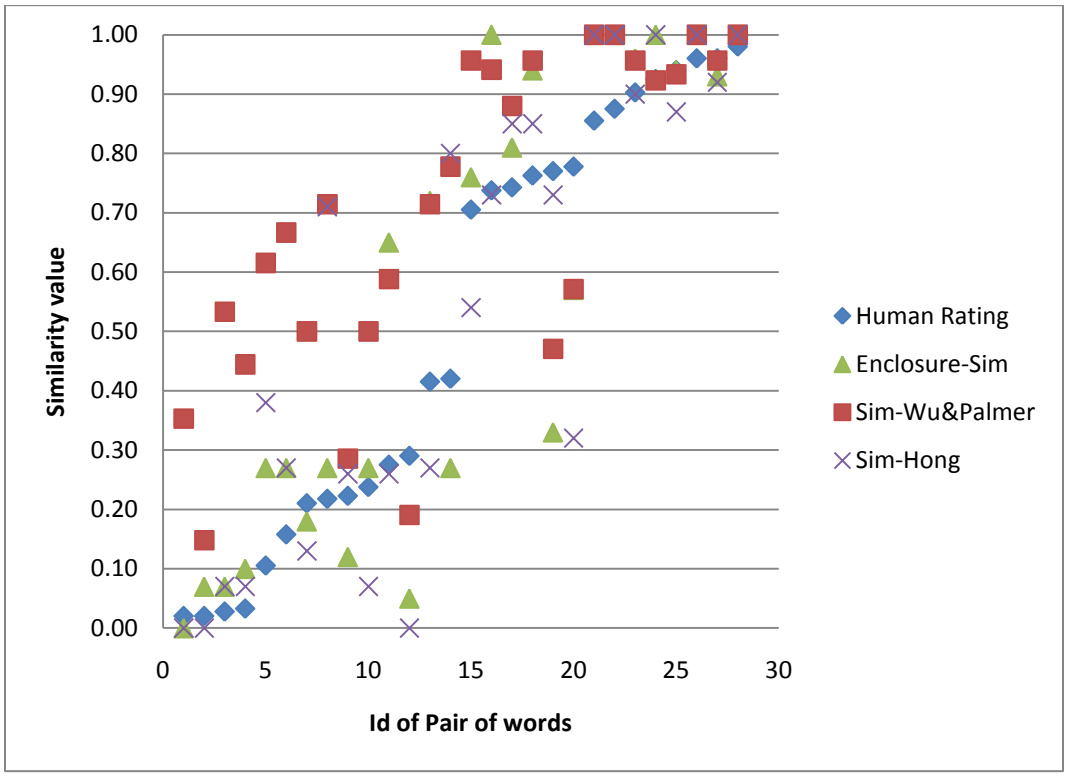


Figure 6.12: Correlation between human ratings and measures in the Miller–Charles datasets

6.5.3.2 Our relatedness measure

Using our clustering strategy (cf. Section 4.3), we compared (*i*) our semantic relatedness algorithm, (*ii*) the *TF-IDF* measure and (*iii*) *xSim* on real datasets, with and/or without semantic information. For each measure, we calculate PR, R, and *f*-score values.

In this test, we used 158 RSS news items of Group 1 real dataset (c.f. 6.5.1) and grouped into 6 predefined clusters.

Recalling the definition of *precision* and *recall*, and our clustering algorithm *RaGALL* (cf. Section 4.3), the precision and recall graphs exhibit two basic properties independent of the similarity measure used:

(i) precision around clustering level 1 (which contains news related with related value of 1 and/or with equality/inclusion relationship of) is maximum (i.e., PR = 1 and the clusters are smaller and disjoint), whereas recall value is very low (it means that there are many mis-matching clusters)

(ii) precision around clustering level 0 (results in all news items with relatedness value greater than or equal to 0 together) is very low (resulting in bigger clusters), whereas recall value is higher as mis-clustering is lower.

Hence, the actual clustering of datasets should end before attaining clustering level zero.

Figure 6.13 shows the *f-score* graph corresponding to each of the three similarity measure. Even though the relationship between news items was not identified in this dataset, our relationship-aware clustering algorithm groups all items related with inclusion and equality in the appropriate cluster (between clustering levels 1 and 0.7). The average *f*-score *value* computed over the entire clustering level shows that our semantic relatedness measure provides relevant clustering results (clusters closer to the predefined ones, particularly between 1 and 0.37) compared to *xSim* and *TF-IDF*.

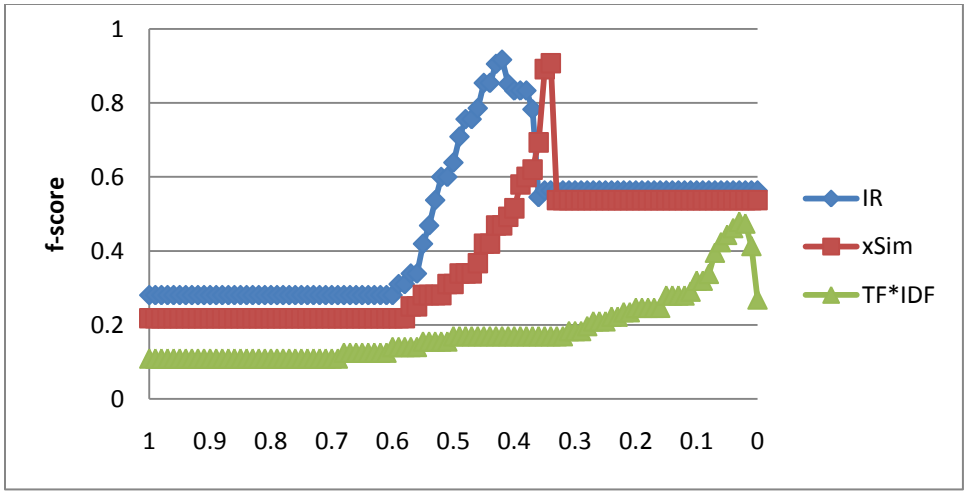


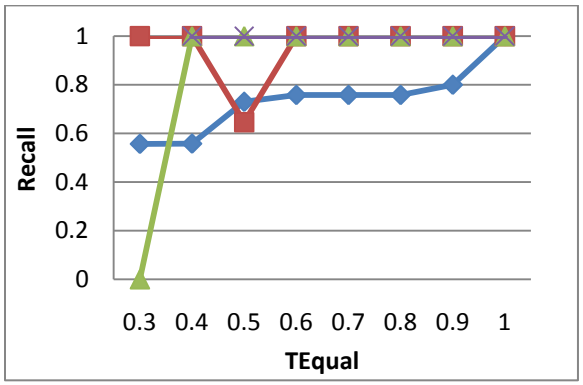
Figure 6.13: f-score on Group 1 real data set

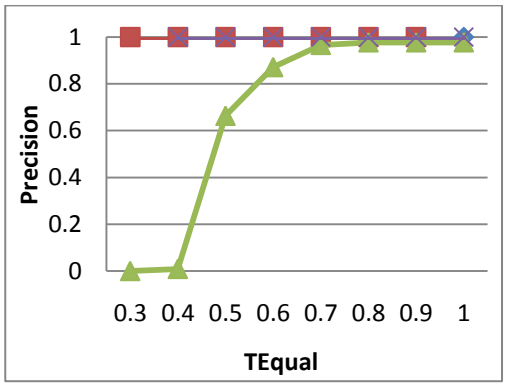
6.5.3.3 Item/Element Relations

In this set of tests, we show to which extent our relatedness measure correctly identifies the *equality*, *inclusion*, *intersection/overlapping* and/or *disjointness* relations between elements. We generated 600 synthetic news items with various distributions and used equal weight to label and content similarity.

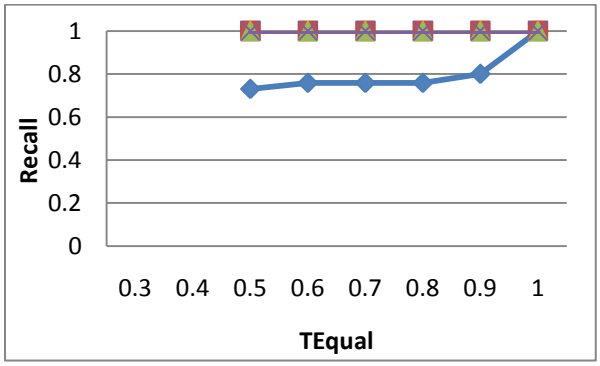
Figure 6.14 shows the Recall and Precision graphs generated on a distribution having: 100 news items related with *equal*, 50 related with *included*, 350 related with *overlapping*, and 100 related with *disjoint* by varying the two similarity thresholds ($T_{Disjointness}$ and T_{Equal}) between 0.3 and 1. In Figure 6.14, we present only some of the relevance graph computed with $T_{Disjointness}$ = 0.3, 0.5 and 0.8 while varying the T_{Equal} between 0.3 and 1 (representing X-axis). Notice that the $T_{Disjointness}$ value is logically less than or equal to T_{Equal}; and the graphs only show the valid combinations. The precision graphs show that our measure accurately identifies *inclusion* relationship independent of the value of $T_{Disjointness}$. However, the recall value w.r.t. the *overlap/intersection* relationship becomes lower as the news items might be considered as *equal*. For instance, at T_{Equal} of 0.5 (c.f. the recall graph in Figure 6.14.A, the sudden drop in recall value is

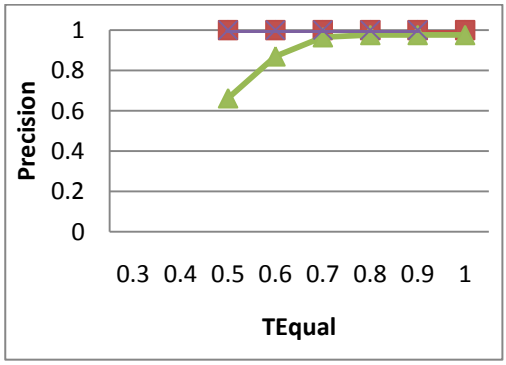
the result of considering news items related with include as equal) and also when T_{Equal} less than 0.4, the overlapping news are considered as *equal* (c.f. Figure 6.14.A)



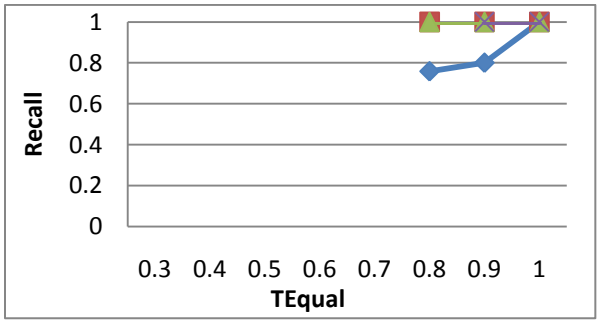


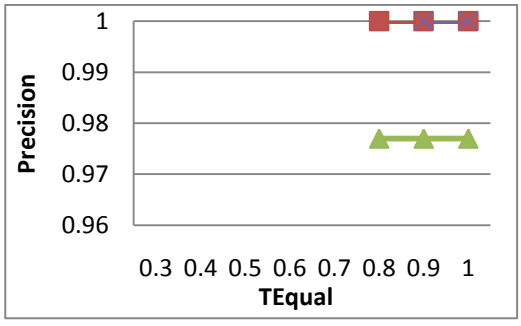
A) Recall and Precision graph at $T_{Disjointness}$ = 0.3





B) Recall and Precision graph at $T_{Disjointness}$ = 0.5





C) Recall and Precision graph at $T_{Disjointness}$ = 0.8

Figure 6.14: Relevance of relationships identification using synthetic RSS data

We have noticed that our measure misclassifies disjoint news items and considers them as overlapped due to element label relatedness when $T_{Disjointness}$ is less than 0.30. The use of higher $T_{Disjointness}$ value misclassifies overlapping news items as disjoint (c.f. Figure 6.14.C). The precision value decreases with the *overlap* relationship around a threshold of 0.5, as the news items are considered equal using T_{Equal} between 0.3 and 0.6 and $T_{Disjointness}$ less than T_{Equal}.

From this experiment, we can conclude here that a correlation can be identified between the threshold values and the distribution of news relationships. Observing the nature of the relevance graphs, we recommend to use $T_{Disjointness}$ = 0.3 and T_{Equal} = 0.7 as default threshold values in computing the relatedness between textual values. But, we believe that accurate values can be inferred using learning and mining techniques. This issue needs to be studied further in the future.

6.5.3.4 Relation aware clustering

As stated in Section 4.3, our *RaGALL* algorithm places news items related with *include* and *equal* relationship, in addition to those having maximum relatedness, in the same cluster. We evaluate this fact experimentally using Group 2 real dataset (c.f. 6.5.1) having 567 news items. In Figure 6.15, we present *f-score* results when clustering real data using our *RaGALL* algorithm and the original group average link level (*GALL*) algorithm. Our clustering algorithm group together the news items related with include and equal relationship at level 1, whereas the group average link clustering algorithm contains only equal news (which would have maximum relatedness values). Our *RaGALL* makes sure that news items related with include relationship are in the same cluster independent of the similarity value and the clustering can be terminated without waiting till the end of the clustering.

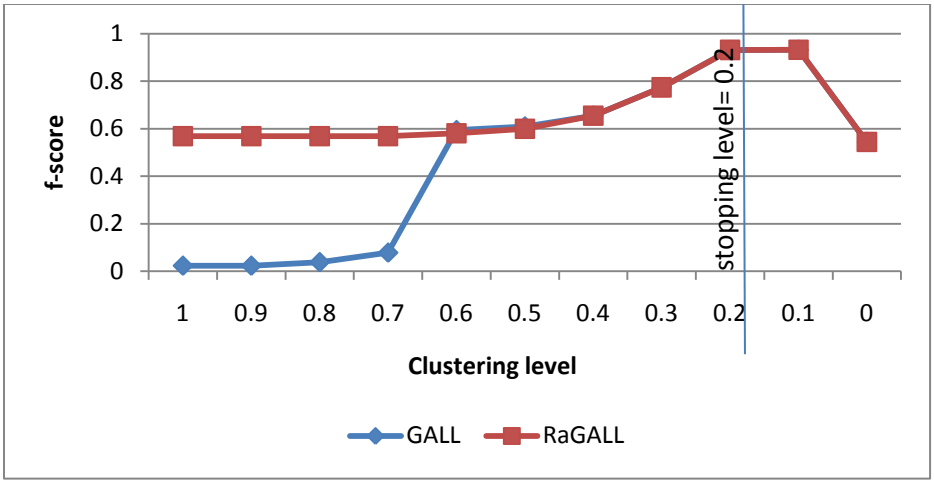


Figure 6.15: Group 2 real RSS items clustered with GALL and our RaGALL algorithms

6.5.3.5 Feed merging

To test the relevance of our feed merging approach in providing qualitative, redundant free and customizable option, we have extended the non official Google Reader API, and we let 5 university users to rate their experience of using the two systems -Google Reader API and *EasyRSSManager*.

We let the students to use both desktop prototype *EasyRSSManager* and the extended version of Google Reader API in order to merge news collected from users' favorite sources. The selected students have experience surfing the Web, using aggregators and have Gmail account. The relevance checking has been done in three steps and incrementally. Initially, the students were asked to use each system without the merging option. Then, they used each system with the default merging rules option; and finally they are allowed to provide their personalized merging rules by combining the template provided in Figure 6.6.

At the end, the subjects answered rated (1 (easy or strongly disagree) – 5(best or strongly agree)) questions focusing on three basic requirements (i.e., R1 – R3). A sample questionnaire is attached as annex 1.

R1. Completeness of the merging approach in merging RSS documents.

- The approach provides content existing in all sources
- Easiness of using merging rule editor
- Need to have customizable merging rules

R2. Quality of the merging option – redundancy free RSS news result

- Quality of the merged result
- The extent to which the merger provides the expectation of the user: for instance, getting set of similar news published by more than one publisher.

R3. Flexibility of the merging option in allowing users to have personalized result

- Flexibility in providing new merging rule
- Extensibility of the merging approach
- Overall rating of *EasyRSSManager* in comparison to non merged Google Reader

Table 6.3 shows the rating of each student to each of the three requirements. The average ratings over each requirement confirm the relevance of the approach. In the future we have a plan to release large scale public version of our prototype and collect users' relevance feedback.

Table 6.3: Students response to three requirements

Requirement / Student	R1	R2	R3
S1	5	5	4
S2	5	5	5
S3	4	5	4
S4	4	5	3
S5	5	5	5
Average	**4.6**	**5**	**4.2**

Currently, we are working on a large scale web-based public version of the prototype and will collect users' relevance feedback.

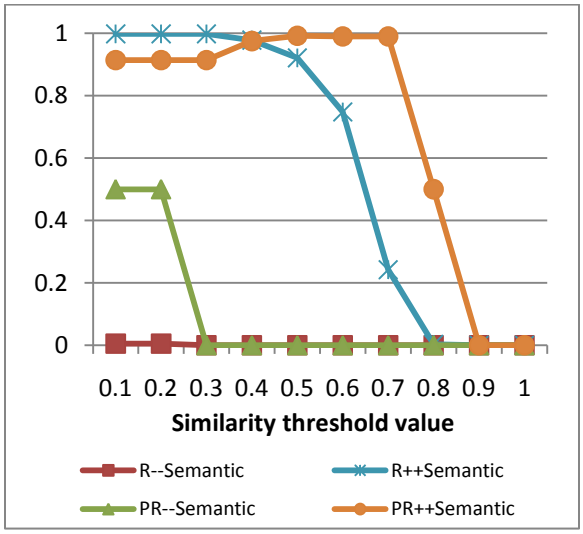
6.5.3.6 Semantic-aware RSS algebra

To test the relevance of our semantic-aware RSS algebra, we have conducted an experiment on a dataset having 902 real news items extracted from the AG's dataset(GULLI, A., 2004).

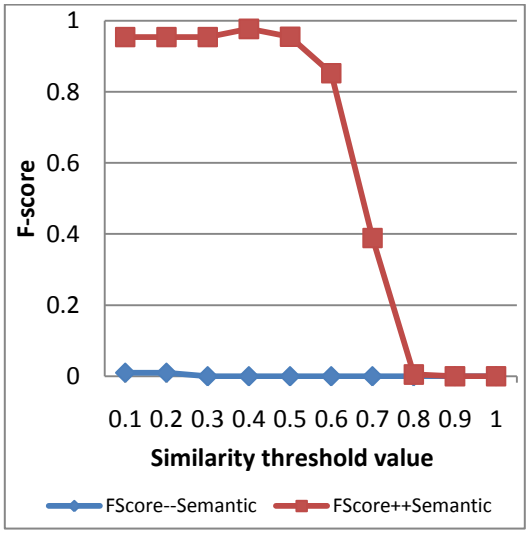
Recalling the definition of recall and precision, querying without semantic information returns values existing exactly in the database and hence the precision is supposed to be higher. On the contrary, semantic query processing improves the recall value.

In conducting the experiment, we filter manually 400 news items as relevant result to the user query to get all news items with title value "*car bombing in Baghdad*". Then, we issued an equivalent query "*auto explosion in Bagdad*" and computed the *recall R*, *precision PR* and *f-score* value with and without considering semantic information while varying the equality threshold value between 0 and 1. The semantic information is restrained only within a maximum threshold of 3. Figure 6.16 shows the relevance of the text-based selection operation computed using semantic (denoted as ++ *semantic*) and without (referred to as --*semantic*). Even if the dataset used in the experiment contains a set of news about the query few is retrieved without using semantic (c.f. PR and R at threshold value of 0.1 and 0.2). As a result, the recall value at any threshold value is less than 0.1. Hence, the use of the semantic information provides a better similarity value and identifies a number of relevant news items that couldn't be retrieved otherwise. The Figure 6.16 shows that the semantic-aware query processing provides more relevant result with threshold value between 0.1 and 0.6. The use of higher threshold value necessitates exactness and higher precision value and hence the recall value decreases. The *f-score* (c.f. Figure 6.16.B) shows the relative quality of having semantic-aware query processing in comparison to the query without semantics.

For this particular query, the semantic query processing provides the most relevant result at similarity threshold of 0.4 and the recall value is much higher for any threshold value less than 0.4.



A) Recall and Precision graphs



B) f-score graph

Figure 6.16: PR, R and f-score graph for relevance of semantic based selection operation

6.6 Summary

In this chapter, we have presented our prototype –*EasyRSSManager*- designed to validate and demonstrate the practicability of the different proposals of this research. *EasyRSSManager* is both semantic-aware RSS Reader, semantic-aware and window-based RSS query processor.

The semantic RSS reader component proposes an extension to Google Reader using WordNet based enclosure similarity. The data component of the *EasyRSSManager* is stored in MySQL. The graphical user interfaces are designed to facilitate the interaction between end-user and *EasyRSSManager*

In addition, we have presented the result of the set of experiments conducted to verify the different proposals discussed in this thesis. The experiments are conducted using the prototype *EasyRSSManager* on both real and syntactic datasets. The validity of our approaches have been confirmed by comparing the result of our approaches to known approaches using Information Retrieval measures (i.e., Recall, Precision and f-score), and human based relevance rating.

The contribution of this chapter can be summarized as follows:

- we verified experimentally the polynomial nature of our relatedness measures.
- we evaluated the relevance of our *enclosure semantic similarity* measure in identifying the similarity between words using Miller-Charles dataset and compared it against some of the WordNet based semantic similarity measures. We found that our measure correlates better to human similarity rating.
- we evaluated the relevance of our RSS semantic relatedness measure in computing similarity and identifying relationship existing between elements using *recall*, *precision* and *f-score*
- we evaluated the relevance of our RSS merger using human relevance rating
- we also evaluated the relevance of semantic-aware RSS algebra using real dataset and semantic-based query.

CHAPTER 7
CONCLUSION

The recent popularity of RSS and Atom formats fuel the web communities, publishing industries, web services, etc. to publish and exchange XML documents from distributed sources. In addition, it allows a user to consume data/information easily without roaming from site to site using software applications (e.g., mashup tools, feed readers/aggregators). At the heart of this juncture point, there is a need to have an integrated framework that allows a user and/or an application to:

- integrate the heterogeneous and distributed feed data, and
- query a news stream using an easy to use interface.

Providing such integrated feed information involves identifying the user context, handling heterogeneity caused by the different feed encoding formats and versions, and feed contents. Until now, the main approaches in fusing/integrating such information involve the use of exact matching operations. However, defining such key values on text-rich and author dependent semi-structured information is close to impossible.

The main theme of this thesis is the study of semantic-aware feed management framework. In this work, we provide a framework that:

1) integrates semantic information in news feed management,
2) measures the semantic relatedness between entities to be compared

3) integrates distributed and heterogeneous feeds using context-aware and rule-based approach
4) facilitates querying dynamic news items using semantic-aware operators without defining identical key values
5) facilitates the news feed management using easy to user interface.

The remainder of this chapter is organized as follows. In Section 7.1, we summarize the main contribution of our works. In Section 7.2, we conclude the report with some of our future research directions.

7.1 Contributions

The major contributions of this thesis can be summarized as follows:

1. we proposed a generic approach to identify the semantic neighborhood of a concept as a set of related concepts extracted from a given Knowledge Base. We also proposed a generic asymmetric semantic similarity measure able to identify the semantic similarity value and relationship existing between a pair of concepts. The measure is based on the ratio of the number of shared concepts in the global semantic neighborhood of each concept and the cardinality of the global semantic neighborhood of the second concept.
2. we introduced three algorithms (*TR*, *ER*, and *IR*) that work at different level of granularity following the bottom-up design principle. The algorithms return a pair notifying the degree of semantic similarity and relationship values between entities to be compared.
3. we extended the link clustering algorithm to make it relationship-aware as the existing clustering algorithm group together mainly highly similar and highly overlapping documents/news. We demonstrated that disregarding relationship would lead to the existence of false negative clusters as related items could be assigned to different clusters.

4. we proposed a context-aware and rule-based merging framework targeting news items. We also demonstrated that the merger can be plugged into existing RSS aggregators (such as Google Reader) and provide collections of news items satisfying a personalized merging conditions.
5. we introduced window-based and semantic-aware querying operators. We defined a novel operator –*Merge*- that generalizes the binary *Join* and *Set membership* operators. We also showed that *Select* and *Merge* operators are the minimal required in feed context.
6. we developed a prototype –*EasyRSSManager*- to validate and demonstrate the practicability of the different proposals made in this thesis. We also tested the relevance of our approaches using both real and synthetic datasets.
7. We published several issues in both international conference and journal.

Publications

International Journal (2):

1. GETAHUN, F and R. CHBEIR. 2010. RSS Query Algebra: Towards Better News Management. Journal of *Information sciences* (submitted)
2. GETAHUN, F., J. TEKLI, R. CHBEIR et al. 2009. Semantic-based Merging of RSS items. *World Wide Web: Internet and Web Information Systems Journal Special Issue: Human-Centered Web Science*. 13, pp.169-207.

International Conference (4)

1. GETAHUN, F and R CHBEIR. 2010. SEMANTIC AWARE RSS QUERY ALGEBRA. *In*: *12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010)*. Paris , France.
2. GETAHUN, F. and R. CHBEIR. 2010. RSS Merger. *In*: *Extraction et gestion des connaissances EGC 2010*. Hammamet, Tunisie, pp.637-638.

3. GETAHUN, F., J. TEKLI, R. CHBEIR et al. 2009. Relating RSS News/Items. *In*: *9th International Conference on Web Engineering ICWE 2009*. San Sebastian, Spain: Springer Verlag LNCS, pp.442-45.

4. GETAHUN, F., J. TEKLI, M. VIVIANI et al. 2009. Towards Semantic-based RSS Merging. *In*: *International Symposium on Intelligent Interactive Multimedia Systems and Services*., pp.53-64.

7.2 Future research direction

As mentioned above, our research work resulted in various contributions. At the same time it allows us to identify various improvement and future research related to efficiency, parameter tuning, and also extending the result of the work into other application areas. In the remaining part of this section, we present few possible improvements and extensions related to the realm of this research followed by its potential applications.

7.2.1 Possible improvement and extensions

7.2.1.1 Enclosure similarity measure

The enclosure similarity measure proposed in this thesis is capable to identify both the similarity and the relationship between concepts/words using the number of shared and difference words in their set of words extracted from their corresponding global semantic neighborhood. It is to be noted that a word might have different senses and hence have different associated meanings. The enclosure similarity measure developed in this thesis considers only the most popular sense of a word. Consequently, the similarity between two words having different senses and also their corresponding concepts might be less. We believe that word/concept sense disambiguation could be one of the promising areas to improve similarity value and would be used as a pre-step to our enclosure similarity

measure. One of the viable solutions for word sense disambiguation problem is to look for the closer senses representing the two words/concepts. This can be done using Dijksra's shortest path algorithm (CORMEN, Thomas H. et al., 2001). In addition, the measure considers the use of neighborhood threshold value provided by the user. In the future we study the behavior and identify the maximum threshold value that should be utilized in getting optimal value.

7.2.1.2 Texts relatedness measure

The relatedness between two textual values is computed by combining the relatedness/similarity between concepts extracted from each text. Hence, the relatedness value is directly proportional to the number of common concepts and inversely proportional to the number of different concepts of both texts.

In our text relatedness approach, we disregard the frequency of a concept (i.e., the number of times a concept appears in the text) to give equal chance to each concept and disregard the bias on using redundant concept. However, using all concepts of the texts has impact on the similarity value. To demonstrate this, let us consider the following three texts:

T_1: Iran powers up nuclear plant. (AJE - Al Jazeera English)

T_2: Nuclear fuel set to arrive in Iran. (CNN world news)

T_3: Despite sanctions, Iran fuels first nuclear reactor (NYT world news)

The three texts describe a similar event but the utilized concepts are different. It is straight forward to infer that $SIM(T_1, T_2) > SIM(T_1, T_3)$ as the number of common concepts of T_1 and T_2 is more than the number of common concepts of T_1 and T_3 and the number of different concepts of T_1 and T_2 is less than that of the number of different concepts of T_1 and T_3. i.e., $\frac{|CS(T_1) \cap CS(T_2)|}{|CS(T_1) \setminus CS(T_2)|} > \frac{|CS(T_1) \cap CS(T_3)|}{|CS(T_1) \setminus CS(T_2)|}$. However, the actual information in T_3, is almost the same as that of T_1 and T_2.

The effect of this problem is very visible in text similarity selection operation. Consider the following selection query over the texts (T_1, T_2 and T_3): "search for all news items with title '*Iran nuclear'* having a similarity value of 0.7". The result of this query might be empty or may not include T_3.

We believe that there are two viable solutions to address this problem. The first is to apply query rewriting and relaxation approaches whenever the result of the query is empty or insufficient. However, we feel that this solution might add some noise to the result. The second solution is to summarize the text with key-phrase extraction algorithm (WITTEN, I. H. et al., 1999) and identify the keywords (concepts) automatically using machine learning techniques. As a result, our relatedness algorithm would be applied on shorter texts and hence both the performance and relatedness value might be improved.

7.2.1.3 Query optimization

In this thesis, we proposed a window-based and semantic-aware feeds querying operators to retrieve the set of news items satisfying a given condition. Normally, the use of semantic information improves the relevance of query result at the cost of degrading the system efficiency. One option to improve the system efficiency is to adaptively optimize the query before it is executed. We believe that the query processor has to adaptively decide when to use semantic information, the window size, and also the re-querying strategy.

7.2.1.4 Adaptive threshold values

In this thesis, we proposed text relatedness algorithm which is used later in the text based selection operator. The algorithm accepts two threshold values (i.e., disjoint and equality thresholds) as an input. Especially, the equality threshold value determines whether two texts are related/similarity or not and consequently affects the news items to be returned. The use of appropriate threshold value is crucial for an efficient utilization of the query operator and getting relevant results. For instance, the lower the equality threshold value,

the higher the noise in the result set. Whereas, the higher the equality threshold value, the lesser is the number of relevant results. In the future, we plan to investigate the different options to tune the equality threshold automatically and adaptively in consideration of the length of the texts to be compared using artificial intelligent and mining approaches.

7.2.1.5 Multimedia feed aggregation

Recently various extensions of RSS and Atom have emerged and applied to multimedia files (audio, video, and image)- MRSS[45], RSS-TV[46]-, geographical encoded objects – GeoRSS, etc. There is a high demand to integrate the different media syndicate formats and contents so that users and applications can transparently and easily access the information. Doing this involves:

i) defining a generic interoperability multimedia data model that represents all participant feeds
ii) identifying the relatedness between the multimedia feeds
iii) aggregating related items in proper manner
iv) allowing multi-criteria query (using both low-level features such as color, texture, shape, and high-level features such keywords and contextual information) and
v) presenting the result to the user adapting to the current context.

We believe that the result of this thesis can be extended into multimedia feed streams retrieval for two main reasons:

1) our relatedness algorithms can be easily tuned to work with a generic multimedia integration item model
2) our framework is relatively generic and can work with any external Knowledge Base and similarity measures

[45] MRSS is an RSS module that supplements the <enclosure> element capabilities of RSS 2.0 to allow for more robust media syndication: http://video.search.yahoo.com/mrss
[46] http://www.rss-tv.org/

7.2.2 Potential application

In this sub-section, we provide some of the application areas where the result of this thesis can be applied.

7.2.2.1 Thai RSS News Search Engine - Ongoing research

Currently, we are working in collaboration with the National Electronics and Computer Technology Center (NECTEC) of Thailand to extend our similarity measure to retrieve RSS news items. The main objective of this research is to design a similarity-based RSS News Search and Merge Engine applicable to Thai language. The result will allow Thai users to formulate a multi-lingual query (English and Thai) to retrieve mainly Thai resources such as online news and blogs and provide more relevant, ranked, and aggregated results. In addition, it will provide personalized news information taking into consideration user's preferences with the help of user-based ontology (e.g. Agrovoc[47]). In this research, we witnessed that retrieving Thai documents introduces its own challenge such as segmentation and named entity identification rather than stemming and stop-word removal. Subsequently, we need to adapt our measure to fit to the identified entities, expand user query terms with semantic information.

7.2.2.2 Extending XQuery

Even though XQuery 1.1 is extremely powerful, its functionality is limited to the retrieval of querying a set of documents using a predicate that uses equality or inequality of simple values or deep-equality of elements. Until now, the standard XQuery is not semantic-aware. In (RYS, M., 2003), Rys documented three different approaches to extend XQuery:

1. sub-language approach: adding a language interface on the top of XQuery engine
2. function approach: adding a set of functions to XQuery engine

[47] http://naist.cpe.ku.ac.th/agrovoc

3. syntactic approach: adding a new statement/operator/clause that provide the functionality

We believe that the set of similarity function proposed in this thesis work can be easily incorporated in XQuery engine using either function or syntactic approach.

7.2.2.3 Extending Prototype

We believe that the result of this research would benefit the news readers at large. Thus, we plan to release a public and online version of the system that would help a user to:

1) compute the similarity between a pair of words/concepts and texts.
2) search for news items using a query string composed of
    a) set of keywords
    b) set of keywords together with context information (i.e., tag label that contains the text)
    c) a well defined structured news item (query by example).
3) merge news items

In addition, the public version would help us to collect large scale assessment of our proposals.

Bibliography

BIBLIOGRAPHY

ADALI, S., P. BONATTI, M. L. SAPINO, and V. S. SUBRAHMANIAN. 1998. A Multi-Similarity Algebra. *In*: *SIGMOD*., pp.402-413.

AGRAWAL, S., S. CHAUDHURI, and V. R. NARASAYYA. 2000. Automated selection of materialized views and indexes in SQL databases. *In*: *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*. San Francisco, CA, USA, pp.496-505.

ALANEN, M. and I. PORRES. 2003. *Difference and Union of Models*. Turku, Finland: Turku Centre for Computer Science.

ALDENDEFER, M. S. and R. K. BLASHFIELD. 1984. *Cluster Analysis*. CA, Sage: Beverly Hills.

ALTINEL, M., P. BROWN, S. CLINE et al. 2007. Damia: a data mashup fabric for intranet applications. *In*: *VLDB '07*., p.1370–1373.

ANDERSON, N. H. 1981. Foundations of information integration theory. *In*: *Academic Press*. San Diego, CA, USA.

BABCOCK, B., M. DATAR, and R. MOTWANI. 2002. Sampling from a moving window over streaming data. *In*: *SODA*., pp.633-634.

BAEZA-YATES, R. and B. RIBEIRO-NETO. 1999. *Modern Information Retrieval*. Addison Wesley.

BAI, Y., H. THAKKAR, H. WANG et al. 2006. A data stream language and system designed for power and extensibility. *In*: *CIKM*. CIKM, pp.337-346.

BARIONI, M. C. N., H. L. RAZENTE, A. J. M. TRAINA, and C. TRAINA. 2006. SIREN: A similarity retrieval engine for complex data. *In*: *32nd international conference on Very large data bases*., pp.1155 - 1158.

BATINI, C., Lenzerini, M., and Navathe, S. B. 1986. A comparative analysis of methodologies for database schema integration. *In*: *ACM Computing Surveys*., pp.323-364.

BEECH, D., A. MALHOTRA, and M. RYS. 1999. A formal data model and algebra for xml.

BEERI, C. and Y. TZABAN. 1999. SAL: An Algebra for Semistructured Data and XML. *In*: *WebDB (Informal Proceedings)*., pp.37-42.

BERGAMASCHI, S., S. CASTANO, M. VINCINI, and D. BENEVENTANO. 2001. Semantic integration of heterogeneous information sources. *In*: *Data and Knowledge Engineering*., p.215–249.

BERGAMASCHI, S., F. GUERRA, M. ORSINI et al. 2007. RELEVANT News: a semantic news feed aggregator. *In*: *4th Workshop on Semantic Web Applications and Perspectives (SWAP 2007)*. Bari, Italy, pp.150-159.

BERLINER, B. 1990. CVS II: Parallelizing Software Development. *In*: *Proceedings of the Winter 1990 USENIX Conference*. Washington DC, USA: USENIX Assoc, pp.341-352.

BERNSTEIN, P. A. 2003. Applying Model Management to Classical Meta Data Problems. *In*: *CIDR*., pp.209-220.

BERNSTEIN, P. A. and L. M. HAAS. 2008. Information integration in the enterprise. *In*: *Communications of the ACM*., pp.72-79.

BILLE, P. 2005. A survey on tree edit distance and related problems. *In*: *Theoretical Computer Science*., pp.217-239.

BRACHMAN, R. J. and J. G. SCHMOKE. 1985. An Overview of the KL- ONE Knowledge Representation System. *In*: *Cognitive Science*., pp.171-216.

BRAY, T. 2005. *RSS 2.0 and Atom 1.0 compared*. [online]. [Accessed 16 May 2010]. Available from World Wide Web: <http://www.tbray.org/atom/RSS-and-Atom>

Bibliography

BRAY, T., J. PAOLI, C.M. SPERBERG-MCQUEEN et al. 2006. *Extensible Markup Language (XML) 1.1 (Second Edition)*. [online]. [Accessed 07 April 2010]. Available from World Wide Web: <http://www.w3.org/TR/xml11/>

BRUNET, G., M. CHECHIK, S. EASTERBROOK et al. 2006. A manifesto for model merging. *In*: *GaMMa '06*. Shanghai, China, pp.5 - 12.

BUNEMAN, P., A. DEUTSCH, and W.C. TAN. 1999. A deterministic model for semi-structured data. *In*: *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*. Jerusalem, Israel.

BUTTLER, D. 2004. A Short Survey of Document Structure Similarity Algorithms. *In*: *International Conference on Internet Computing*., pp.3-9.

CATANIA, B., E. FERRARI, A. Y. LEVY, and A. O. MENDELZON. 2000. XML and Object Technology. *In*: *ECOOP Workshops 2000*., pp.191-202.

CBCL. 2009. *XQSharp: XQuery 1.0 for the Microsoft.NET Framework*. [online]. [Accessed 01 April 2010]. Available from World Wide Web: <http://xqsharp.com/xqsharp/>

CHAMBERLIN, D., J. ROBIE, and D. FLORESCU. 2000. Quilt: An XML Query Language for Heterogeneous Data Sources. *In*: *Springer-Verlag*., pp.53-62.

CHAUDHURI, S. and U. DAYAL. 1997. An overview of data warehousing and OLAP technology. *In*: *ACM SIGMOD Record*., pp.65-74.

CHAWATHE, S. S. 1999. Comparing hierarchical data in external memory. *In*: *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, pp.90-101.

CHAWATHE, S. S., H. GARCIA-MOLINA, J. HAMMER et al. 1994. The TSIMMIS project: Integration of heterogeneous information sources. *In*: *IPSJ*., pp.7-18.

CHEN, W. and T. BøEN. 2008. A Personalized RSS News Filtering Agent. *In*: *AI-2007, the Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of AI*., pp.321-326.

CHEN, Z., H. V. JAGADISH, L. V. S. LAKSHMANAN, and S. PAPARIZOS. 2003. From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery. *In*: *VLDB*., pp.237-248.

CIONCA, L. 2008. *Keeping good news stories together: just part of what it means to work on Google News*. [online]. [Accessed 09 June 2010]. Available from World Wide Web: <http://googlenewsblog.blogspot.com/2008/05/keeping-good-news-stories-together-just.html>

CLARK, J. and S. DEROSE. 1999. *XML Path Language (XPath) Version 1.0. W3C Recommendation*. [online]. [Accessed 23 Mar 2010]. Available from World Wide Web: <http://www.w3.org/TR/xpath/>

CLUET, S., C. DELOBEL, J. SIMÉON, and K. SMAGA. 1998. Your Mediators Need Data Conversion. *In*: *ACM SIGMOD International Conference on Management of Data*. Seattle, Washington, USA, pp.177-188.

CODD, E. F. 1970. A relational model of data for large shared data banks. *In*: *Communications of the ACM*., p.377–387.

COHEN, W. 1998. A web-based information system that reasons with structured collections of text. *In*: *Proceedings of Autonomous Agents'98*., pp.400--407.

COHEN, S., J. MAMOU, Y. KANZA, and Y. SAGIV. 2003. XSEarch: A Semantic Search Engine for XML. *In*: *VLDB 2003*., pp.45-56.

COLAZZO, D., P. MANGHI, and C. SARTIANI. 2001. *Xtasy: A typed xml database management system*.

COLLINS-SUSSMAN, B., W. F. BRIAN, and C. M. PILATO. 2004. *Version Control with Subversion*. O'Reilly Media.

Bibliography

CORMEN, Thomas H., Charles E. LEISERSON, Ronald L. RIVEST, and Clifford STEIN. 2001. Section 24.3: Dijkstra's algorithm. *In*: *Introduction to Algorithms*, MIT Press and McGraw-Hill, p.595–601.

COTTER, P. and B. SMYTH. 2000. Personalization Technologies for the Digital TV World. *In*: *Proceedings of Prestigious Applications of Intelligent Systems, (PAIS'00)*. Berlin, Germany: IOS, p.701–705.

CURBERA, F. 1998. *alphaworks*. [online]. [Accessed 19 May 2O1O]. Available from World Wide Web: <http://www.alphaworks.ibm.com/tech/xmldiffmerge>

DAHLGREN, K. 1995. A Linguistic Ontology. *In*: *International Journal on Human-Computer Studies*., p.809–818.

DALAMAGAS, T., T. CHENG, K.-J. WINKEL, and T.K. SELLIS. 2006. A methodology for clustering XML documents by structure. **31**(3), p.187–228.

DE VRIES, A.P., N. MAMOULIS, N. NES, and M.L. KERSTEN. 2002. Efficient k-nn search on vertically decomposed data. *In*: *SIGMOD*. New York: ACM, p.322–333.

DEROSE, S., E. MALER, and D. ORCHARD. 2001. *XML Linking Language (XLink) Version 1.0*. [online]. [Accessed 07 April 2010]. Available from World Wide Web: <http://www.w3.org/TR/xlink/>

DEUTSCH, A. 1998. *XML-QL: A Query Language for XML*. [online]. [Accessed 25 November 2009]. Available from World Wide Web: <http://www.w3.org/TR/NOTE-xml-ql/>

DEY, A. K. 2001. Understanding and using context. *Personal and Ubiquitous Computing: Special issue on Situated Interaction and Ubiquitous Computing*. **5**(1), pp.4-7.

DEY, A. K., G. D. ABOWD, and D. SALBER. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*. **16**(2), pp.97-166.

DI LORENZO, G., H. HACID, H. PAIK, and B. BENATALLAH. 2009. Data integration in mashups. *In*: *SIGMOD Rec.*, pp.59-66.

DITTRICH, K. R. and D. JONSCHER. 1999. All together now: Towards integrating the world's information systems. *In*: Y. Masunaga and S. SPACCAPIETRA, (ed). *Advances in Multimedia and Databases for the New Century*. Kyoto, Japan: World Scientific Press, p.109–123.

DOAN, A. and A. Y. HALEVY. 2005. Semantic Integration Research in the Database Community: A Brief Survey. *In*: *AI Magazine.*, pp.83-94.

DUSCHKA, O. M. and M. R. GENESERETH. 1997. Query planning in infomaster. *In*: *SAC '97: Proceedings of the 1997 ACM symposium on Applied computing*. New York, NY, USA: ACM, pp.109-111.

EHRIG, M. and Y. SURE. 2004. Ontology mapping - an integrated approach. *In*: *First European Semantic Web Symposium*. Greece: LNCS, p.76–91.

EJIGU, D., M. SCUTURICI, and L. BRUNIE. 2008. Hybrid Approach to Collaborative Context-Aware Service Platform for Pervasive Computing. *JOURNAL OF COMPUTERS*. **3**(1), pp.40-50.

ENNALS, R. J. and M. N. GAROFALAKIS. 2007. Mashmaker: mashups for the masses.. *In*: *SIGMOD '07.*, p. 1116–1118.

FERNÁNDEZ, M. F., J. SIMÉON, and P. WADLER. 2000. An Algebra for XML Query. *In*: *FSTTCS.*, pp.11-45.

FISHER, D., F. LAM, and R. K. WONG. April 2004. Algebraic Transformation and Optimization for XQuery. *In*: *Advanced Web Technologies and Applications (Proc. 6th Asia-Pacific Web Conference)*. LNCS , pp.201-210.

Bibliography

FLESCA, S., G. MANCO, E. MASCIARI, and L. PONTIERI. 2005. Fast detection of xml structural similarity. *In*: *IEEE Transactions on Knowledge and Data Engineering*., pp.160-175.

FONTAINE, R.L. 2002. Merging XML files: A new approach providing intelligent merge of XML data sets. *In*: *Proceedings of XML Europe '02*. Barcelona Spain.

FOX, E. A. 1983. *Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types*.

FRASINCAR, F., G. HOUBEN, and C. PAU. 2002. XAL: an algebra for XML query optimization. *In*: *Proceedings of the 13th Australasian Database Conference*. Melbourne, Victoria, Australia: In Proceedings of the 13th Australasian Database Conference, pp.49-56.

GARCIA, I. and Y. NG. 2006. Eliminating Redundant and Less-Informative RSS News Articles Based on Word Similarity and a Fuzzy Equivalence Relation. *In*: *the 18th IEEE international Conference on Tools with Artificial intelligence*., pp.465-473.

GETAHUN, F. and R. CHBEIR. 2010. RSS Merger. *In*: *Extraction et gestion des connaissances EGC 2010*. Hammamet, Tunisie, pp.637-638.

GETAHUN, F. and R. CHBEIR. 2010. RSS Query Algebra: Towards Better News Management. *Information sciences*., p.submitted.

GETAHUN, F and R CHBEIR. 2010. SEMANTIC AWARE RSS QUERY ALGEBRA. *In*: *12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010)*. Paris , France.

GETAHUN, F., J. TEKLI, S. ATNAFU, and R. CHBEIR. 2007. Towards Efficient Horizontal Multimedia Database Fragmentation using Semantic-based Predicates Implication. *In*: *SBBD 2007*., pp.68-82.

GETAHUN, F., J. TEKLI, R. CHBEIR et al. 2009. Relating RSS News/Items. *In*: *9th International Conference on Web Engineering ICWE 2009*. San Sebastian, Spain: Springer Verlag LNCS, pp.442-45.

GETAHUN, F., J. TEKLI, R. CHBEIR et al. 2009. Semantic-based Merging of RSS items. *World Wide Web: Internet and Web Information Systems Journal Special Issue: Human-Centered Web Science*. **13**, pp.169-207.

GETAHUN, F., J. TEKLI, M. VIVIANI et al. 2009. Towards Semantic-based RSS Merging. *In*: *International Symposium on Intelligent Interactive Multimedia Systems and Services*., pp.53-64.

GHOSH, S. and P. MITRA. 2008. Combining content and structure similarity for XML document classification using composite SVM kernels. *In*: *ICPR 2008*., pp.1-4.

GOLDING, A. 2008. *Psst.secrets of Google News exposed!* [online]. [Accessed 09 June 2010]. Available from World Wide Web: <http://googlenewsblog.blogspot.com/2008/04/psstsecrets-of-google-news-exposed.html>

GONZÁLEZ, J. L. V. and A. F. RODRÍGUEZ. 2000. A Semantic Approach to Question Answering Systems. *In*: *TREC 2000*. Gaithersburg, USA, pp.13-16.

GOWER, J. C. and G. J. S. ROSS. 1969. Minimum Spanning Trees and Single Linkage Cluster Analysis. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*. **18**(1), pp.54-64.

GULLI, A. 2004. *AG's corpus of news articles*. [online]. [Accessed January 2009]. Available from World Wide Web: <http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html>

GUO, L., F. SHAO, C. BOTEV, and J. SHANMUGASUNDARAM. 2003. XRANK: Ranked Keyword Search over XML Documents. *In*: *SIGMOD Conference 2003*., pp.16-27.

GUSTAFSON, N. and M. S., Ng, Y. PERA. 2008. Generating Fuzzy Equivalence Classes on RSS News Articles for Retrieving Correlated Information. *In*: *ICCSA*., pp.232-247.

HALEVY, A. Y. 2001. Answering queries using views: A survey. *In*: *The VLDB Journal, The International Journal on Very Large Data Bases*., pp.270-294.

Bibliography

HAMMER, J., H. GARCÍA-MOLINA, S. NESTOROV et al. 1997. Template-based wrappers in the TSIMMIS system. *In*: *Proceedings of ACM SIGMOD'97*. ACM.

HAMMERSLEY, B. 2003. *Content Syndication with RSS*. San Francisco: O'Reilly & Associates Publishers.

HARINARAYAN, V., A. RAJARAMAN, and J. D. ULLMAN. 1996. Implementing data cubes efficiently. *In*: *SIGMOD Rec.*, pp.205-216.

HARTIGAN, J. A. and M. A. WONG. 1979. A k-means clustering algorithm. *Applied Statistics*. **28** (1), pp.100-108.

HERNÁNDEZ, M. A. and S. J. STOLFO. 1995. The merge/purge problem for large databases. *In*: *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, pp.127-138.

HOEBER, O., X. D. YANG, and Y. YAO. 2005. Conceptual Query Expansion. *In*: *AWIC.*, pp.190-196.

HONG-MINH, T. and D SMITH. 2008. Word Similarity In WordNet. *In*: *Proceedings of the Third International Conference on High Performance Scientific Computing*. Hanoi, Vietnam: Springer, p.293–302.

HORN, A. 1951. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*. **16**, pp.14-21.

HUBERT, L.J. and J.R. LEVIN. 1976. A general statistical framework for accessing categorical clustering in free recall,. *Psychol. Bulletin*. **83** , p.1072–1082.

HUMPHREYS, B. L. and D. A. B LINDBERG. The UMLS Project: Making the Conceptual Connection between Users and the Information They Need. *In*: *Bulletin of the Medical Library Association.*, p.170.

HUNG, E., Y. DENG, and V. S. SUBRAHMANIAN. 2004. TOSS: an extension of TAX with ontologies and similarity queries. *In*: *SIGMOD '04*. Paris, France: ACM, pp.719-730.

HUNTER, A. and W. LIU. 2006. Fusion rules for merging uncertain information. *In*: *Information Fusion*., pp.97-134.

HUNTER, A. and W. LIU. 2006. Merging uncertain information with semantic heterogeneity in XML. *In*: *Knowledge and Information Systems*., pp.230-258.

HUNTER, A. and R. SUMMERTON. 2003. Propositional fusion rules. *In*: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. LNCS, pp.502-514.

HUNTER, A. and R. SUMMERTON. 2004. Fusion rules for context-dependent aggregation of structured news reports. **14(3)**(Journal of Applied Non-Classical Logics), pp.329-366.

HUNTER, A. and R. SUMMERTON. 2006. A knowledge-based approach to merging information. *In*: *Knowledge Based Systems*., pp.647-674.

HUYNH, D., S. MAZZOCCHI, and D. KARGER. 2007. Piggy Bank: Experience the Semantic Web inside your web browser. *Web Semant.* **5**(1), pp.16-27.

HUYNH, D. F., R. C. MILLER, and D. R. KARGER. 2008. Potluck: Data mash-up tool for casual users. *Web Semantics: Science, Services and Agents on the World Wide Web*. **6**(4), pp.274-282.

IGNAT, C. L. and C. N. MOIRA. 2005. Flexible Merging of Hierarchical Documents. *In*: *Proceedings of the Seventh International Workshop on Collaborative Editing, GROUP`05*. Sanibel Island, Florida, USA: IEEE Distributed Systems, p.18–25.

IGNAT, C.L. and C. N. MOIRA. 2005. Operation-based Merging of Hierarchical Documents. *In*: *Proceedings of the CAiSE 2005 Forum, 17th International Conference on Advanced Information Systems Engineering*. Porto, Portugal.

Bibliography

JAGADISH, H. V., L. V. LAKSHMANAN, D. SRIVASTAVA et al. 2001. TAX: A Tree Algebra for XML. *In*: *Proceedings of the 8th International Workshop on Database Programming Language*. London: Springer-Verlag, pp.149-164.

JAMES, W. 1890. *Principles of psychology*. New York, NY, US: Henry Holt.

JARDINE, N. and R. SIBSON. 1971. *Mathematical Taxonomy*. New York : John Wiley & Sons, Inc..

JIANG, J. J. and D. W. CONRATH. 1997. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. *In*: *In Proceedings of International Conference Research on Computational Linguistics (ROCLING X)*. Taiwan, pp.19-33.

JOHANNES GEHRKE, Flip Korn, Divesh Srivastava. 2001. On Computing Correlated Aggregates Over Continual Data Streams. *In*: *SIGMOD.*, pp. 13-24.

KADE, A. M. and C. A. HEUSER. 2008. Matching XML documents in highly dynamic applications. *In*: *Proceeding of the eighth ACM symposium on Document engineering*. Sao Paulo, Brazil: ACM, pp.191-198.

KANNE, C. and G. MOERKOTTE. 2000. Efficient storage of XML data. *In*: *In Proceedings of the 16th International Conference on Data Engineering*. Society Press, pp.198 - 198.

KAPPEL, G., E. KAPSAMMER, S. RAUSCH-SCHOTT, and W. RETSCHITZEGGER. 2000. X-Ray - Towards Integrating XML and Relational Database Systems. *In*: *Proc. of the 19th Int. Conf. on Conceptual Modeling (ER)*. Salt Lake City, USA: Lecture Notes in Computer Science, pp.9-12.

KIM, W. 2008. XML document similarity measure in terms of the structure and contents. *In*: *Proceedings of the 2nd WSEAS international Conference on Computer Engineering and Applications CEA'08*. Acapulco, Mexico, pp.205-212.

KIM, T., J. , Song J. LEE, and D. KIM. 2007. Similarity Measurement of XML Documents Based on Structure and Contents. *In*: *ICCS 2007*. LNCS, p.902–905.

KING, B. 1967. Step-wise Clustering Procedures. *American Stat. Assoc*. **69**, pp.86-101.

KOLOVOS, D. S., R. F. PAIGE, and F. A.C. POLACK. 2006. Merging Models with the Epsilon Merging Language (EML). *In*: *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*. Genova, Italy, pp.215-229.

KONIECZNY, S., J. LANG, and P. MARQUIS. 2004. DA2 merging operators. *Artif. Intell.* **157**(1-2), pp.49-79.

KORN, F., B. PAGEL, and C. FALOUTSOS. 2001. *On the 'Dimensionality Curse' and the 'Self-Similarity Blessing'*. IEEE Trans. Knowl. Data Eng.

KOSTAS P., Timos K. S. 2006. Window Specification over Data Streams. *In*: *EDBT Workshops*., pp.445-464.

KROGSTIE, J., A.L. OPDAHL, and G. SINDRE. 2007. Generic Schema Merging. *In*: *19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*. LNCS, p.127–141.

LAMBRECHT, E., S. KAMBHAMPATI, and S. GNANAPRAKASAM. 1999. Optimizing recursive information-gathering plans. *In*: *IJCAI '99: Proceedings of the 16th International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA, pp.1204-1211.

LANDAUER, T. K. and S. T. DUMAIS. 1997. Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge. *In*: *Psychological Review*., pp.211-240.

LAU, H. and W. NG. 2005. A Unifying Framework for Merging and Evaluating XML Information. *In*: *DASFAA 2005*. LNCS, pp.81-94.

LAU, H. and W. NG. 2007. Towards Adaptive Information Merging Using Selected XML Fragments. *In*: *DASFAA 2007*., pp.1013-1019.

Bibliography

LEACOCK, C. and M. CHODOROW. 1998. Combining local context and WordNet similarity for word sense identification. *In*: *Fellbaum.*, pp.265-283.

LEE, K., M. KIM, K. LEE et al. 2002. Conflict Classification and Resolution in Heterogeneous Information Integration based on XML Schema. *In*: *IEEE TENCON'02*. IEEE Computer Society, pp.93 - 96.

LENZERINI, M. 2002. Data integration: a theoretical perspective. *In*: *PODS '02: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM, pp.233-246.

LESK, M. E. 1986. Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. *In*: *5th International Conference on Systems Documentation*. Toronto, Ontario, pp.24-26.

LEVY, A. Y., A. RAJARAMAN, and J. J. ORDILLE. 1996. Querying heterogeneous information sources using source descriptions. *In*: *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp.251-262.

LIAN, W., David, Mamoulis,N. WAI-LOK C., and S. YIU. 2004. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *In*: *IEEE Trans. Knowl. Data Eng.*, pp.82-96.

LIN, D. 1998. An information-theoretic definition of similarity. *In*: *CA Proceedings of the Fifteenth international Conference on Machine Learning*. San Francisco, , pp.296-304.

LINDHOLM, T. 2003. XML three-way merge as a reconciliation engine for mobile data. *In*: *MobiDe '03: Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*. New York, NY, USA: ACM, pp.93-97.

LINDHOLM, T. 2004. A three-way merge for XML documents. *In*: *DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering*. New York, NY, USA: ACM, pp.1-10.

LIN, Y., J. ZHANG, and J. GRAY. 2005. A Testing Framework for Model Transformations. *In*: *Model driven Software Development*. Springer Heidelberg, p.219–236.

LIU, J. and L. BIRNBAUM. 2008. LocalSavvy: aggregating local points of view about news issues. *In*: *ACM*. LOCWEB '08, pp.33-40.

LIU, H., V. RAMASUBRAMANIAN, and E. G. SIRER. 2005. Client Behavior and Feed Characteristics of RSS, a Publish-Subscribe System for Web Micronews. *In*: *In the proc. of ACM Internet Measurement Conference*.

LI, X., J. YAN, Z. DENG et al. 2007. A novel clustering-based RSS aggregator. *In*: *WWW '07*. New York, NY: ACM, pp.1309-1310.

LOCKER, L Jr, GB SIMPSON, and M YATES. 2003. Semantic neighborhood effects on the recognition of ambiguous words. Mem Cognit, pp.505-15.

LUDÄSCHER, B., Y. PAPAKONSTANTINOU, and P. VELIKHOV. 1999. A Framework for Navigation-Driven Lazy Mediator. *In*: *ACM Workshop on the Web and Database*.

MA, Y. and R. CHBEIR. 2005. Content and Structure Based Approach For XML Similarity. *In*: *Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05)*. IEEE Computer Society, pp.136-140.

MAGUITMAN, A. G., F. MENCZER, H. ROINESTAD, and A. VESPIGNANI. 2005. Algorithmic Detection of Semantic Similarity. *In*: *In Proc. of the 14th Inter. WWW Conference*. Japan, pp.107-116.

MANOLESCU, I., D. FLORESCU, D. KOSSMANN et al. 2000. Agora: Living with XML and relational. *In*: *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB)*. Cairo, Egypt, pp.623-626.

Bibliography

MCBRIDE, B. 2004. *logics and RDF schema (RDFS) (W3C Recommendation)*. [online]. [Accessed 05 Jun 2010]. Available from World Wide Web: <http://www.w3.org/TR/rdf-mt/>

MCGILL, M. J. 1983. *Introduction to Modern Information Retrieval*. New York: McGraw-Hil.

MCGUINNESS, D. L. and F. HARMELEN. 2004. *OWL Web Ontology Language Overview (W3C Recommendation)*. [online]. [Accessed 05 Jun 2010]. Available from World Wide Web: <http://www.w3.org/TR/owl-features/>

MELTON, Jim and Stephen BUXTON. 2006. *XQuery, XPath, and SQL/XML in context*. The Morgan Kaufmann Series in Data Management Systems.

MIHALCEA, R., C. CORLEY, and C. STRAPPARAVA. 2006. Corpus-based and knowledge-based measures of text semantic similarity. *In*: *Proceedings of the American Association for Artificial Intelligence*., pp.775-780.

MILLER, G. A., R. BECKWITH, C. FELLBAUM et al. 1990. Nouns in WordNet: a Lexical Inheritance System. *In*: *International Journal of Lexicography*.

MILLER, G. and W. CHARLES. 1998. Contextual correlates of semantic similarity. *In*: *Language and Cognitive Processes*., p.1–28.

MILLIGAN, G. W. and M. C. COOPER. 1985. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*. **50**, p.159–179.

MINSKY, M. 1975. Minsky. A framework for representing knowledge. *In*: *The Psychology of Computer Vision*. New York: McGraw-Hill, pp.211-277.

MOHANIA, M. and M. BHIDE. 2008. New trends in information integration. *In*: *ICUIMC '08: Proceedings of the 2nd international conference on Ubiquitous information management and communication*. New York, NY, USA: ACM, pp.74-81.

NANNO, T. and M. OKUMURA. 2006. HTML2RSS: automatic generation of RSS feed based on structure analysis of HTML document. *In*: *Proceedings of the 15th international Conference on World Wide Web*. Edinburgh, Scotland: ACM.

NASROLAHI, S., M. NIKDAST, and M. M. BOROUJERDI. 2009. The Semantic Web: a New Approach for Future World Wide Web. *In*: *World Academy of Science, Engineering and Technology*., pp.1149-1154.

NAUGHTON, J. F., D. J DEWITT, D. MAIER et al. 2001. The Niagara Internet query system. *In*: *IEEE Data Eng. Bulletin*., pp.27-33.

NEJATI, S., M. SABETZADEH, M. CHECHIK et al. 2007. Matching and Merging of Statecharts Specifications. *In*: *ICSE 2007*. IEEE Computer Society, pp.54-64.

NIERMAN, A. and H. V. JAGADISH. 2002. Evaluating Structural Similarity in XML Documents. WebDB, pp.61-66.

NOVAK, L. and A. V. ZAMULIN. 2006. An XML Algebra for XQuery. *In*: *In 10th East European Conference on Advances in Databases and Information Systems (ADBIS 06)*. Thessaloniki, Greece: LNCS, p.4–21.

OGAWA, Y., T. MORITA, and K. KOBAYASHI. 1991. A fuzzy document retrieval system using the keyword connection matrix and a learning method. *Fuzzy Sets and Systems*. **39**(2), pp.163-179.

PAPAKONSTANTINOU, Y., V. BOKAR, M. ORGIYAN et al. March 2003. XML queries and algebra in the Enosys integration platform. *In*: *Data & Knowledge Engineering*., pp.299-322.

PAPARIZOS, S., Y., Lakshmanan, L. V. WU, and H. V. JAGADISH. 2004. Tree Logical Classes for Efficient Evaluation of XQuery. Conference. *In*: *ACM SIGMOD international Conference on Management of Data*. Paris, France, pp.71-82.

PENG, X. and B. CHOI. 2005. Document classifications based on word semantic hierarchies. *In*: *International Conference on Artificial Intelligence and Applications*., p.362–367.

Bibliography

PERA, M. S. and Y NG. 2007. Finding Similar RSS News Articles Using Correlation-Based Phrase Matching. *In*: *KSEM 2007*., pp.336-348.

PERA, M. S. and Y. NG. 2008. Utilizing Phrase-Similarity Measures for Detecting and Clustering Informative RSS News Articles. *In*: *Integrated Computer-Aided Engineering*. Amsterdam, The Netherlands: IOS Press, pp.331-350.

PÉREZ, J., M. ARENAS, and C. GUTIERREZ. 2009. Semantics and Complexity of SPARQL. *In*: *International Semantic Web Conference*. ACM Trans. Database Syst., pp.1-45.

PORTER, M. F. 1980. An algorithm for suffix stripping. Program., p.130—137.

POTTINGER, R. A. and P. A BERNSTEIN. 2003. Merging Models Based on Given Correspondences. *In*: *Proceedings of the Twenty-Ninth International Conference on Very Large Databases (VLDB)*. San Francisco: Morgan Kaufmann Publishers, pp.862 - 873.

POULOVASSILIS, A. and P. MCBRIEN. 1998. A general formal framework for schema transformation. *In*: *Data and Knowledge Engineering*., p.47–71.

QIN, P., Z. LU, Y. YAN, and F. WU. 2009. A New Measure of Word Semantic Similarity Based on WordNet Hierarchy and DAG Theory. *In*: *wism*., pp.181-185.

RADA, R. and E. BICKNELL. 1989. Indexing documents with a thesaurus. *In*: RANKING, (ed). *Journal of the American Society for Information Science*., pp.304-310.

RAFIEI, D., D. L. MOISE, and D. SUN. 2006. Finding Syntactic Similarities Between XML Documents. *In*: *DEXA Workshops*., pp.512-516.

RAJESWARI, V. and K. Dharmishtan K. VARUGHESE. 2009. Heterogeneous Database integration for Web Applications. *International Journal on Computer Science and Engineering*. **1**(3), pp.227-234.

RAJPAL, N. 2002. *Using the XML Diff and Patch Tool in Your Applications*. [online]. [Accessed 19 May 2010]. Available from World Wide Web: <http://msdn.microsoft.com/en-us/library/aa302294.aspx>

RESNIK, P. 1995. Using information content to evaluate semantic similarity in a taxonomy. *In*: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Montreal, Canada, p.448–453.

RICHARDSON, R. and A.F. SMEATON. 1995. Using WordNet in a Knowledge-Based Approach to Information Retrieval. *In*: *Proc. of 17th Colloquium on Information Retrieval*.

ROBIE, J., D. CHAMBERLIN, M. DYCK, and J. SNELSON. 2009. *XQuery 1.1: An XML Query Language W3C Working Draft*. [online]. [Accessed 27 January 2010]. Available from World Wide Web: <http://www.w3.org/TR/xquery-11/>

RODRÍGUEZ, M. A. and M. J. EGENHOFER. 2003. Determining Semantic Similarity among Entity Classes from Different Ontologies. IEEE Transactions on Knowledge and Data Engineering, pp.442-456.

RSS ADVISORY BOARD. 2009. *RSS 2.0 Specification*. [online]. [Accessed 12 November 09]. Available from World Wide Web: <http://www.rssboard.org/>

RUNDENSTEINER, X. and E. ZHANG. 2002. *XAT: XML Algebra for the Rainbow System*.

RYS, M. 2003. Full-Text Search with XQuery: A Status Report. *In*: *In Intelligent Search on XML*. Springer-Verlag, pp.39-57.

SALTON, G., A. WONG, and C. S. YANG. 1975. A vector space model for automatic indexing. *Communications of the ACM*. **18**(11), pp.613-620.

SAMPERA, J. Juan, A. Pedro CASTILLOB, Lourdes ARAUJOC et al. 2008. NectaRSS, an intelligent RSS feed reader. *Journal of Network and Computer Applications*. **31**(4), pp.793-806.

SARTIANI, C. and A. ALBANO. 2002. Yet Another Query Algebra For XML Data. *In*: *IDEAS.*, pp.106-115.

Bibliography

SCHLIEDER, T. and H. MEUSS. 2002. Querying and ranking XML documents. *In*: *Journal of the American Society for Information Science and Technology*., pp.489-503.

SCHMIDT, A., M. KERSTEN, M. WINDHOUWER, and F. WAAS. 2000. Efficient relational storage and retrieval of XML documents. *In*: *Proceedings of the 3rd International Workshop on the Web and Database*. Dallas, Texas, p.47–52.

SHANMUGASUNDARAM, J., K. TUFTE, C. ZHANG et al. 1999. Relational Databases for Querying XML Documents: Limitations and Opportunities. *In*: *Proceedings of 25th International Conference on Very Large Data Bases*. Edinburgh, Scotland, pp.302-314.

SHEPARD, R. N. 1987. Toward a universal law of generalization for psychological science. *Science, New Series*. **237**(4820), pp.1317-1323.

SHIMURA, T., M. YOSHIKAWA, and S. UEMURA. 1999. Storage and Retrieval of XML Documents using Object-Relational Databases. *In*: *Proc. of Database and Expert Systems Applications (DEXA)*., p.206–217.

SHIR, E. and J. AIZEN. 2005. *Dapper: The Data Mapper*. [online]. Available from World Wide Web: <http://www.dapper.net/>

SILBERSCHATZ, A., M. STONEBRAKER, and J. D. ULLMAN. 1991. Database systems: Achievements and opportunities. *In*: *Communications of the ACM*., pp.110-120.

SKIENA, S. S. 1998. *The Algorithm Design Manual*. Springer-Verlag.

SMEATON, R. and A. F. RICHARDSON. 1995. *Using wordnet in a knowledge-based approach to information retrieval.* Dublin, Ireland: Trinity College.

SMULLYAN, M. R. 1995. *First-Order Logic*. New York: Dover.

SNEATH, P. H. A and R.R. SOKAL. 1973. *Numerical Taxonomy: The Principles and Practice of Numerical Classification*. San Francisco.

SONG, I., J. PAIK, and U. KIM. 2007. Semantic-based Similarity Computation for XML Document. *In*: *the 2007 international Conference on Multimedia and Ubiquitous Engineering*. Washington, DC: IEEE Computer Society, pp.796-803.

SYEDA-MAHMOOD, T., P. TURAGA, D. BEYMER et al. 2010. Shape-based similarity retrieval of doppler images for clinical decision support. *In*: *IEEE Conerence on Computer Vision and Pattern Recognition (CVPR)*. San Fancisco, CA.

TEKLI, J., R. CHBEIR, and K. YETONGNON. 2007. A Hybrid Approach for XML Similarity. *In*: *SOFSEM 2007: Theory and Practice of Computer Science*. LNCS, pp.783-795.

THEOBALD, M., R. SCHENKEL, and G. WEIKUM. 2005. An Efficient and Versatile Query Engine for TopX Search. *In*: *VLDB 2005*., pp.625-636.

THEODORATOS, D. and T. K. SELLIS. 1997. Data warehouse configuration. *In*: *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp.126-135.

TICHY, W. 1985. RCS: A system for version control. *In*: *Software - Practice & Experience*., pp.637-654.

TOVE MANUAL. 1995. *Department of Industrial Engineering*. [online]. [Accessed 07 April 2010]. Available from World Wide Web: <www.ie.utoronto.ca/EIL/tove/ontoTOC.html>

TRAINA, C., A. J. M. TRAINA, M. R. VIEIRA et al. 2006. Efficient processing of complex similarity queries in rdbms through query rewriting. *In*: *CIKM*.

TRAN, Tien, Richi NAYAK, and Peter D BRUZA. 2008. Combining structure and content similarities for XML document clustering. *In*: *7th Australasian Data Mining Conference*. Glenelg, South Australia, pp.27-28.

TSENG, F. S.C. 2005. XML based heterogeneous Database Integration for Data warehouse Creation. *In*: *Ninth Pacific Asia Conference on Information Systems (PACIS) 2005*., pp.590-603.

Bibliography

TUFTE, K. and D. MAIER. 2001. Aggreation and Accumulation of XML Data. *In*: *IEEE Data Engineering Bulletin*., p.34–39.

TUFTE, K. and D. MAIER. 2002. Merge as a Lattice-Join of XML Documents. *In*: *the 28th VLDB Conference*. Hong Kong, China.

TURNEY, P. D. 2001. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. *In*: *Proceedings of the 12th European Conference on Machine Learning*. LNCS, pp.491 - 502.

TVERSKY, A. 1977. Features of similarity. *Psychological Review*. **84** , pp.327-352.

ULLMAN, J. D. 1997. Information integration using logical views. *In*: *ICDT '97: Proceedings of the 6th International Conference on Database Theory*. London, UK: LNCS, pp.19-40.

VIGLAS, S. D., L. GALANIS, D. J. DEWITT, and D. MAIER. *Putting XML query algebras into context.* [online]. Available from World Wide Web: <http://www.cs.wisc.edu/niagara/.>

VIYANON, W. and S. K. MADRIA. 2009. *XDI-CSSKA: System for Detecting XML Similarity on content and structure using relational database*.

VIYANON, W. and S. K. MADRIA. 2009. XML-SIM: Structure and Content Semantic Similarity Detection Using Keys. *In*: *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*. LNCS, pp.1183 - 1200.

VIYANON, W., S. K. MADRIA, and S. S. BHOWMICK. 2008. XML Data Integration Based on Content and Structure Similarity Using Keys. *In*: *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS*. LNCS, p.484–493.

VIYANON, W. and M. SANJAY. 2009. A System for Detecting XML Similarity in Content and Structure Using Relational Database. *In*: *the Proceedings of 18thACM International Conference*

on Information and Knowledge Management (ACM CIKM 2009). Hong Kong, China, pp.1197-1206.

WAGNER, R. and M. FISHER. 1974. The string to string correction problem. *In*: *Journal of ACM*., pp.168-178.

WANG, J., K. UCHINO, T. TAKAHASHI, and S. OKAMOTO. 2006. RSS Feed Generation from Legacy HTML Pages. *In*: *APWeb 2006*., pp.1071-1082.

WANG, X. H., D. Q. ZHANG, T. GU, and H. K. PUNG. 2004. Ontology Based Context Modeling and Reasoning using OWL. *In*: *Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*. Orlando, Florida, p.18.

WEGRZYN-WOLSKA, K. and S. P. SZCZEPANIAK. 2005. Classification of RSS-Formatted Documents Using Full Text Similarity Measures. *In*: *ICWE 2005*., pp.400-405.

WEI, W., M. LIU, and S. LI. 2004. Merging of XML Documents. *In*: *ER 2004*., pp.273-285.

WITTEN, I. H., G. W. PAYNTER, E. FRANK et al. 1999. KEA: practical automatic keyphrase extraction. *In*: *Proceedings of the Fourth ACM Conference on Digital Libraries*. Berkeley, California, United States , pp.254-255.

WORDNET 2.1. 2005. *A Lexical Database of the English Language*. [online]. [Accessed 01 Janauary 2009]. Available from World Wide Web: <http://wordnet.princeton.edu/online/>

WU, S., U. MANBER, G. MYERS, and W. MILLER. 1990. An O(NP) sequence comparison algorithm. *In*: *Information Processing Letters*., pp.317-323.

WU, Z. and M. S. PALMER. 1994. Verb Semantics and Lexical Selection. *In*: *ACL*., pp.133-138.

XIA, X., Y. GUO, and J. LE. 2009. Measuring Similarities between XML Documents based on Content and Structure. *In*: *Information Processing, 2009. APCIP 2009. Asia-Pacific Conference on*. IEEE Computer Society, pp.459 - 462.

Bibliography

YAN, B., M. FRANK, P. SZEKELY et al. 2003. WebScripter: grassroots ontology alignment via end-user report creation. ISWC.

YANG, J., K. KARLAPALEM, and Q. LI. 1997. Algorithms for materialized view design in data warehousing environment. *In*: *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA, pp.136-145.

YANG, Dongqiang and David M. POWERS. 2005. Measuring semantic similarity in the taxonomy of WordNet. *In*: *ACSC '05: Proceedings of the Twenty-eighth Australasian conference on Computer Science*. Darlinghurst, Australia: Australian Computer Society, Inc, pp.315--322.

YOSHIKAWA, M., T. AMAGASA, T. SHIMURA, and S. UEMURA. 2001. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology*. **1 (1)**, pp.110-141.

YUAN, M., P. JIANG, and J. WU. 2008. A Quantitative Method for RSS Based Applications. *In*: *Proceeding of the 2008 Conference on Applications of Data Mining in E-Business and Finance*. Amsterdam, The Netherlands : ACM, pp.75-85.

ZHANG, Z., R. LI, S. CAO, and Y. ZHU. 2003. Similarity metric for XML documents. *In*: *Knowledge Management and Experience Management Workshop*. Karlsruhe, Germany, pp.255-261.

ZHANG, X., E. A. RUNDENSTEINER, G. MITCHELL, and W. LEE. 2001. Clock: Synchronizing Internal Relational Storage with External XML Documents. *In*: *11th International Workshop on research Issues in Data Engineering*. IEEE Computer Society, pp.111-118.

ZHOU, Z., Y. WANG, and J. GU. 2008. New Model of Semantic. Similarity Measuring in Wordnet. *In*: *Proceedings of the 2008 3rd International Conference on Intelligentg System and Knowledge Engineering*., pp.256 - 261.

ZIEGLER, P. and K. R. DITTRICH. 2004. Three decades of data integration – all problems solved? *In*: *IFIP Congress Topical Sessions*. Kluwer, pp.3-12.

Annex 1: Questionnaire

The following set of questions are prepared to access the capabilities of our *EasyRSSManager* particularly in providing merged, integrated and customisable news items extracted from your register feed sources.

1. Have you noticed the existence of redundant, similar, related news items same or different sources?
   a) Yes b) No

   If your answer is **Yes** continue with question #2 else go to question #4.
2. Do you have any experience in identifying redundant and similar news in Google Reader?
   a) Yes b) No
3. If yes, specify the tool: _____
4. The *EasyRSSManager* is easy to use and flexible?
   a) Strongly disagree b) disagree c) d) agree e) strongly agree
5. Have you noticed any difference on the result of *EasyRSSManager* compared to Google Reader?
   a) Yes b) No
6. Do you agree on the idea of providing customizable merging language to get personalised result?
   a) Strongly disagree b) disagree c) don't know d) agree e) strongly agree
7. The merging language -merging rule editor- in *EasyRSSManager* is easy to use?
   a) Strongly disagree b) disagree c) don't know d) agree e) strongly agree
8. The merging language in *EasyRSSManager* provides the minimal set of conditions and actions necessary to integrate news items:
   a) Strongly disagree b) disagree c) don't know d) agree e) strongly agree
9. The merging language provided in *EasyRSSManager* is complete and flexible in providing your expectation?
   a) Strongly disagree b) disagree c) don't know d) agree e) strongly agree

10. *EasyRSSManager* is capable to remove redundant and group similar/related news items using default merging option.
    a) Strongly disagree b) disagree c) don't know d) agree e) strongly agree
11. *EasyRSSManager* provides better result compared to Google Reader
    a) Strongly disagree b) disagree c) don't know d) agree e) strongly agree

Annex 2: Random Access Machine

One of the basic approaches to analyze the computational complexity of machine-independent algorithm is using a hypothetical machine called the Random Access Machine or RAM. In RAM (SKIENA, S. S., 1998), the following three assumptions are fundamental:

- Each “simple” operation (basic arithmetic, assignments, if-statements, etc.) takes a unit time.
- Loops and subroutines/functions are the compositions of many simple operations. The time it takes to run through loops and/or functions depends upon the number of steps the loop iterates and/or the number of simple operations.
- Each memory access takes exactly a unit time and we have an abundant memory as we need. Besides, the model doesn’t notice the existence of memory hierarchy and doesn’t distinguish different types of memory such as cache or external disk.

In addition, the RAM model has a single processor and processes each operation in sequential manner. Hence, using the RAM model, we measure the run time complexity of an algorithm by counting up the number of unit times it takes on a given problem instance. By assuming that our RAM executes a given number of operations per second, the operation count converts easily to the actual run time.

Annex 3: C-index -stopping rule

C¬index is a vector of pairs $\left((i_1, n_1), (i_2, n_2), \cdots, (i_p, n_p)\right)$, where $i_1, \cdots, i_p$ are the values of the index and $n_1, \cdots, n_p$ are the number of clusters in each clusters arrangement produced by varying the clustering level of a hierarchical clustering procedure in p different steps. Let l_1 be the first selected clustering level, which produces an arrangement of N_1 clusters (that is $n_1 = N_1$): C_1 with c_1 elements, C_2 with c_2 elements…C_{N1} with c_{N1} elements. The first index of C-index vector, i_1, is computed as follows:

$$i_1 = \left(d_w - min(d_w)\right) \Big/ \left(max(d_w) - min(d_w)\right)$$

where:

1. $d_w = Sum\left(d_{w_1}\right) + Sum\left(d_{w_2}\right) + \cdots + Sum\left(d_{w_{N_1}}\right)$, with $Sum\left(d_{w_i}\right)$ to be the sum of pairwise distance of all members of cluster (i.e. sum of within cluster distance) $C_i, 1 \leq i \leq n_1$,
2. $max(d_w)$: the sum of the n_d highest pairwise distance in the whole set of data (i.e., sort distances in decreasing order (highest first) and take the Top-n_d sum),
3. $min(d_w)$: the sum of the $n_d$lowest pairwise distance in the whole set of data (i.e., sort distances in decreasing order (highest first) and take the Bottom-n_d sum),

   with $n_d = \sum_{i=1}^{N_1} c_i \times (c_i - 1)/2$ (i.e., the number of all within cluster pairwise similarities). The C-index of all remaining different p clustering levels are calculated in similar method, and get the vector $\left((i_1, n_1), (i_2, n_2), \cdots, (i_p, n_p)\right)$. The number of clusters with the lowest C-index is chosen as the correct clustering.

Annex 4: First Order Logic

First Order Logic (FOL)(SMULLYAN, M. R., 1995) is a rich representation language that allows expressing the relationships among objects using predicates, functions and quantifiers. We define its basic components as follows.

Definition 8.1 [Atom]

An *atomic formula or Atom* is a predicate symbol *r* followed by a bracketed *n*-tuple of terms. It is denoted as: $r(u_1, \ldots, u_n)$, where u_i is a *Term*. A predicate symbol is represented with a lower case letter followed by a string of zero or more lower case letters and digits.

Definition 8.2 [Term]

A *Term* is either a constant, a variable or a function symbol followed by a bracketed *n*-tuple of terms.

Both *constants* and a *function symbols* consist of a lower case letter followed by a string of zero or more lower case letters and digits, whereas a *variable* is an upper case letter followed by a string of zero or more lower case letters and digits.

Definition 8.3 [Formula]

A is a formula if and only if there is a finite sequence of expressions or terms such that each term is either Atom or is the negation of earlier term, conjunction or implication of two earlier terms or is the existential or universal qualification of some of earlier terms with respect to some variable X. It is represented in Figure 8.1 using EBNF grammar.

Expression ::= Atom | Expression Connective Expression | (Quantifier Variable) Expression |
~Expression | Expression

Atom ::=Predicate (Term [, ..., Term])

Term ::= Constant | Variable | Function (Term [, ... , Term])

Connective ::= ∧ | ∨ | ⇒

Quantifier ::= ∀ | ∃

Figure 8.1: Syntax of FOL formula

FOL expressions/formulas are categorized into two: *Sentences* and *Implications*.

A Sentence can be either a single Atom or a set of FOL formulas connected by the connectives '∧' and/or '∨'.

An Implication consists of two FOL formulas that are connected by the connective '⇒'. The first of the two formulas of an implication is called the *antecedent*, whereas the second is called the *consequent* of the implication. The *consequent* happens if the *antecedent* (which is a sentence) is evaluated and found to be True.

For instance, given two set of news items f and g extracted from CNN and BBC feeds respectively, the formula: $\forall X \in f, \forall Y \in g, issimilar(X, Y, True, 0.8) \Rightarrow keeplatest(X, Y)$ results in the execution of the function keeplatest - keeps the latest of any two news items from CNN and BBC semantically related with 80%.

Definition 8.4 [Clause]

A *clause* is a disjunction of finite terms. In logic programming, clause can be written as the implication of body term, conjunction of finite terms (B_j), to head term, disjunction of finite terms (A_i). It can be written as a formula of the form:

$$B_1 \wedge B_2 \cdots \wedge B_n \Rightarrow A_1 \vee A_2 \cdots \vee A_m$$

Definition 8.5 [Horn Clause]

A *Horn clause* (HORN, A., 1951) is a clause that contains exactly one positive term, head, and can be written as $B_1 \wedge B_2 \ldots \wedge B_n \Rightarrow A$

The Horn clause with n > 0 body terms is used as a rule, as the execution of the head formula depends on the result of the body terms.

Notice that, a clause with an empty body is called a *fact.* For instance, for any news item, $\forall X$ hastimestamp(X) is a fact. It is equivalent to True $\Rightarrow \forall X$ hastimestamp(X).