Engineering an Open Web Syndication Interchange with Discovery and Recommender Capabilities

Adrian P. O'Riordan, Oliver O'Mahony
Computer Science Department,
Western Gateway Building,
University College Cork,
Cork, Ireland

Abstract

Web syndication has become a popular means of delivering relevant information to people online but the complexity of standards, algorithms and applications pose considerable challenges to engineers. This paper describes the design and development of a novel Web-based syndication intermediary called InterSynd and a simple Web client as a proof of concept. We developed format-neutral middleware that sits between content sources and the user. Additional objectives were to add feed discovery and recommendation components to the intermediary. A search-based feed discovery module helps users find relevant feed sources. Implicit collaborative recommendations of new feeds are also made to the user. The syndication software built uses open standard XML technologies and the free open source libraries. Extensibility and re-configurability were explicit goals. The experience shows that a modular architecture can combine open source modules to build state-of-the-art syndication middleware and applications. The data produced by software metrics indicate the high degree of modularity retained.

1. Introduction

Syndication has become a popular means of delivering relevant timely information to people online. In general, syndication is the supply of information for re-use, for example print syndication, where newspapers or magazines license articles or comic strips. Web syndication is based on a publish-subscribe system where XML-based formats such as RSS are used for the syndication of Web content such as blogs and news to Websites as well as directly to users. It is a low-cost way for information providers to deliver information only to those who are likely to be interested, in a timely manner. The familiar orange icon, signalling an available feed, is now ubiquitous on Websites. Web syndication has been identified as one of the key technologies of Web 2.0 (O'Reilly, 2005).

Web syndication is one component of the *two-way Web*, so named because of the two-way flow of information (Kuman *et al.*, 2004). Popular Web 2.0 sites, such as delicious.com, Flickr, and digg, exploit Web syndication as well as collaborative tagging and shared activity, using a lightweight *RESTful* API (Fielding, 2000) for integration. Web syndication is now used for a myriad of purposes, including publishing, marketing, news updates, bug-reports, sharing community based data, search, podcasting, and messaging. Microsoft has integrated syndication technology into its new operating system Windows 7, to give just one indicator of the technology's maturity and growing popularity. Web syndication is a set of simple lightweight techniques based on publish/subscribe (but not necessarily using a push mechanism). Conceptually, updates are published through a Web *feed*, and notifications are

sent to each subscribed user. The overall effect is that it becomes possible for a user to stay up-to-date with a potentially much larger set of Websites and content providers than a user could feasibly read through browsing alone. Whereas there are some differences in terminology, a single notification is commonly referred to as a feed; a feed source produces multiple instances of feeds in one location. Blog or news sites consist of lists of entries, usually chronologically ordered, where each entry has a unique URL/URI (called a permalink). The underlying mechanism is usually based on polling; clients have to keep issuing requests to see if there is an update.

Feed readers or aggregators amalgamate a collection of subscribed web feeds, usually contacting different sources to do so, and allowing users to read the content. An aggregator might be desktop software or a Web application. Web-based feed readers such as Google Reader or Bloglines (http://www.bloglines.com) allow users to read content with a browser. The content itself can be kept on the server. In contrast, standalone email-style applications, such as Thunderbird, download content to your computer.

There has been a proliferation of different feed formats and technologies proposed and in operation. The onus these competing standards place on the programmer is a source of motivation for this work. Many content providers publish information using a single format: for example the BBC and CNN both publish online using RSS but not Atom (whereas Google uses both). A more detailed discussion of the various Web syndication technologies is given in the next section. The proliferation of formats has led to the development of different programs and libraries for publishing and processing. Thus many websites and software programs are written to support only a particular flavour of RSS or Atom; for example Userland's Manila software (http://manila.userland.com) caters for the Userland format only. Libraries for dealing with a wide range of RSS versions and Atom are in development, and include ROME (see Section 4.1) and Jakarta Feedparser (http://feedparser.org).

Another source of motivation for this work is the so-called RSS bandwidth problem (Sandler *et al.*, 2005): Websites that make feeds available can see a marked increase in traffic owing to constant polling from clients. A study has found that 55% of the RSS feeds surveyed and monitored updated within the hour (Liu *et al.*, 2005), hence the frequent updating by clients. Web server administrators have consequently developed strategies to avoid a spike in traffic bringing a site to a halt, such as limiting the size of feeds and limiting the access; but these are not scalable. Intermediaries or middleware can significantly reduce this load problem by allowing client sharing of feed data.

In this project we build upon the ROME library (Java.net, 2005), an open source Java library for processing feeds in a format-neutral way. We implemented a simple Web client that can read feeds of different formats transparently. All details of the syndication technology are hidden from the user. We extended the functionality of the system by adding a recommendation component and a feed discovery mechanism through search. We used only open source libraries and open standards in this work.

The paper is structured as follows. Section 2 provides context with background technical material on Web syndication and recommender systems. Section 3 presents our overall solution, a syndication interchange that makes recommendations and illustrates the overall architecture of our approach. Based on this overview, section 4 provides details of the implementation. Section 5 gives an overview of related work. Finally, section 6 summarizes the main achievements of the paper and suggests areas of future work.

2. Background

We begin this section by providing an overview of selective dissemination of information, a precursor to Web syndication. This is followed by a technical discussion of the main Web syndication technologies in use today. These are compared and contrasted. We then introduce recommender systems, information-based systems that recommend or suggest content to users.

2.1 Selective Dissemination and Notification

H.P. Luhn (1958) proposed an automatic system to facilitate the selective dissemination and retrieval of information in the pre-Internet era. His system would have: (i) automatic digesting of documents; (ii) encoding of documents; and (iii) creation and updating of *action-point profiles*. Actions are carried out on the basis of the degree of similarity between incoming documents and profiles. Luhn suggested inferring a user's interests from feedback on the documents the system had sent the user. Unfortunately the hardware and software infrastructure was not in place at the time to implement these ideas.

One of the earliest deployed forms of automated electronic information notification and update system was Selective Dissemination of Information (SDI) (Packer and Soergel, 1979). SDI aimed at keeping scientists up-to-date on the latest scientific publications of potential interest. These systems differed from information retrieval systems by having persistent queries representing long-term interests. The objective of SDI is to have the user participate as a recipient of timely and relevant information without the need for continual querying. These systems never became popular for various reasons; including technical reasons such as the lack of both standardized data formats and lightweight non-intrusive protocols at that time. Technical requirements in the large-scale deployment of SDI include support for information timestamps, a notification mechanism, federation, mediation/proxying and caching (O'Neill, 1991).

A similar technological solution emerged in other areas of computing such as in active databases (Dittrich *et al.,* 2005). Here event-condition-action rules are employed for triggering updates and alerts. In another field, various Internet event notification systems have been developed for software interoperability in distributed systems; many based on CORBA technology, for example Rosenblum and Wolf (1997). Other decentralized mechanisms for notifications and group communication have been developed such as (Carzaninga *et al.,* 2001). All these systems have the disadvantage of not using, or being incompatible with, current Web standards such as HTTP and XML.

To meet an increasing need for a simple mechanism for electronic update, mailing list software based on email broadcast protocols, such as LISTSERV (L-Soft, 1996), became a popular mechanism for disseminating information on Inter-networks; but these facilitated very little or no processing or personalization. SIFT (Stanford Information Filtering Tool) was one of the first dissemination services to use the HTTP protocol for transport (Yan and Garcia-Molina, 1995).

2.2 Web Syndication

The basic Web syndication concept was developed at Apple in 1995 in the form of the MCF (Meta Content Framework) (Andresssen, 1999) and the sample application HotSause. The motivation was to make the Web 'more like a library and less like a messy heap of books.' An XML version of MCF became the RDF (Rich Description Framework) (Klyne and Carroll, 2004). The W3C published an RDF specification in 1999. In RDF Web resources are represented as subject-predicate-object expressions and typically identified by means of an URI (Universal Resource Identifier). Around this

time also a company called Pointcast developed a commercial service to deliver live news and stock quotes over the Internet. This was based on *push* technology where a request originated with the publisher, in contrast to client *pull*.

RSS is the most popular form of Web syndication at present (Hammersley, 2003). It is considered to be a *lightweight* syndication format using HTTP for transport. A heavyweight alternative is Web services technology such as WS-Notification, message-oriented middleware such as Java Message Service (JMS) or CORBA's Notification Service, but these heavyweight options (Jhingran *et al.*, 2002) are not considered further here; see also the discussion in Section 5.2.

RSS is actually a family of related technologies. There has been a proliferation of different versions of RSS. The RSS 2.0 specification, the basis for many extensions, is copyrighted by Harvard University (2003) and is frozen so that no significant changes can be made to it. RSS 2.0 is a simple yet highly extensible format where feed items contain plain text or escaped HTML. Extensions by modules allow RSS to carry multimedia payload (RSS enclosures), support electronic commerce (ecommerce RSS), and geographical information (GeoRSS). A module is a standard way of extending the core RSS specification.

Briefly stated, an RSS 2.0 feed consists of a channel with a number of items (content) within this channel. The compulsory <rss> tag element delimits the root element in the XML document structure. Channels have three required fields: <title>, <link> and <description>. There are established (lightweight) publishing protocols for RSS such as the MetaWeblog and Blogger APIs that are all based on the HTTP Web transport protocol. RSS autodiscovery, although implemented in some browsers, has not been standardized; for example it uses the application/rss+xml MIME type in Internet Explorer 7 but not in Firefox.

ATOM is the main alternative to RSS currently, and was born out of the perceived limitations of RSS. Atom is a proposed IETF standard (Nottingham 2005). Atom, like RSS, is an XML specification. It contains <feed> elements after the XML declaration, stating metadata about the feed source. Feeds are composed of a number of items, known as entries, in <entry> XML tags. An Atom feed document is thus a representation of an Atom feed, including metadata about the feed, and some or all of the entries associated with it. Atom 1.0 requires that both feeds and entries include a title, a unique identifier, and a last-updated timestamp. Entries can contain text, XHTML, various defined content types, or binary data in base 64 format. Atom has an associated publishing protocol called AtomPub.

There are many technical differences between the RSS family and Atom (Atom wiki, 2008). Proponents of Atom believed that the RSS Specification was too loose and unclear and the content model was too weak. Here are listed some of the principal differences: (i) RSS 2.0 may contain either plain text or escaped HTML, with no way to indicate which of the two is provided whereas Atom allows detailed payload metadata; (ii) Atom allows standalone entries whereas RSS does not; (iii) RSS elements are not generally reusable in other XML vocabularies whereas Atom entries are; and (iv) RSS has no internationalization support.

2.3 Recommender Systems

The idea of harnessing personalized information to deal with information overload is not new; Fischer and Stevens (1991) where among the first researchers to investigate personalized mechanisms for electronic information sharing. Their work was inspired by the classic Information Lens project (Malone *et al.*, 1987), a rule-based information-sharing system. The basic premise was that information

producers will not expend the effort necessary to classify messages, but readers will put in a 'limited amount of effort restructuring' (Fischer and Stevens, 1991). In their work the dissemination of information was based on a model containing various predictors of the relevance/usefulness of a message source: frequency (number of such items read), recency (time elapsed since last read) and spacing (distribution across time of exposures).

Recommender systems or collaborative filtering systems produce personal recommendations by predicting items of interest based on users' behaviour. The opinions of users can be obtained explicitly from the users or by using some implicit measures. Collaborative approaches to filtering, or recommendation, exploit the profiles of a community of users (McSherry, 2002). This simulates the word-of-mouth process that occurs socially. The recommendations occur in the context of a social network whether explicit or implicit. One of the first such systems was the Tapestry project at Xerox PARC in the early 1990s where document recommendations from a corpus of documents were made based on *likeit* or *hateit* annotations of a community of users (Goldberg *et al.*, 1992).

Recommendations for a user are generated from the profiles of other users who are deemed to be related. In the case of feed subscriptions, these would be feeds that are not already in the user's subscription list but which are in the subscription lists of users with similar profiles. While the content of a feed may be a strong indicator of relevance to a user, other factors such as quality, importance, trust, timeliness, or novelty could be harnessed also. Collaborative approaches can harness this. Collaborative recommendation can be user-based (memory-based) or item-based (model-based) depending on whether user information is rated and used as parameters or items have such ratings attached (Breese *et al.*, 1998). Collaborative methods need to maintain large quantities of data about users such as subscription lists and usage data. Ratings are usually weighted; a popular weighting scheme is the tf.idf (term frequency/ inverse document frequency) weighting scheme from information retrieval (Sparck-Jones, 1972). Other computational techniques for recommendation include Bayesian classifiers, decision trees, machine learning, clustering, and artificial neural networks (McSherry, 2002).

Recommendation technology has now entered the mainstream with foremost users including amazon.com and NetFlix (http://www.netflix.com). Vendors of recommendation technologies include ChoiceStream (http://www.choicestream.com) and Mavice (http://mavice.com). User-driven classification of Web resources had been identified as one of the key emerging technologies for organizing and processing digital information (Chowdhury and Chowdhury, 2007). Users are increasingly classifying information themselves through tagging, creating user-defined *folksonomies*. Web 2.0 sites such as delicious.com and furl.net encourage users to tag and share resources.

3. Syndication Interchange

This section motivates the syndication interchange project by showing how it fits with existent technological solutions. An overview of the user interface is then given. A brief description of the system architecture completes this section.

3.1 InterSynd Overview

InterSynd is middleware placed between the content source and the user application. The situation is complicated by the existence of blog or ping servers that can sit between the source and the reader or aggregator. Weblogs.com (Verisign) and blo.gs (Yahoo!) are two such servers. The *pinging* is

done by an XML-RPC-based push mechanism. Ping servers push a notification when there is updated content or new content available. Hence clients do not have to wait for the scheduled crawl to notice an update; however ping server technology has yet to be standardized. Server side readers such as Bloglines (http://www.bloglines.com/) and portals such as MyYahoo! (http://my.yahoo.com/) can sit between content source and user. These applications store user subscription information and related content online. InterSynd is thus positioned conceptually as shown in Figure 1.

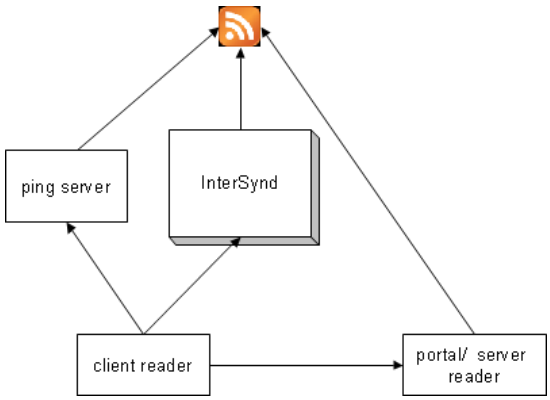

Figure 1. Syndication Middleware

Many aggregators/feed readers also incorporate a filtering system and/or search capabilities. However, with a recommender system in place you can both filter and recommend/present syndication feeds based on user interests. InterSynd is middleware that could be utilized in conjunction with a feed reader. Presently we have a simple InterSynd client that serves as both reader and interchange. The system allows an end-user to register by providing personal information such as name, username, and password. The system maintains user records and feed histories. Each user is able to add to or remove from his/her subscriptions. The system also provides recommendations. We identified the principal functional requirements for an interchange and recommender client: (i) allow users to subscribe to and read Web feeds; (ii) provide non-intrusive recommendations; (iii) store a user's profile privately; (iv) be able to manage large numbers of requests from clients simultaneously; (v) support a high level of performance; (vi) have a content discovery subsystem; (vii) include data caching; and (vii) be free open source software. In addition the library itself had the following non-functional requirements. The system should be: (i) easy to extend and modify; (ii) robust and flexible; (iii) general purpose; and (iv) based on standard formats. Flexibility is to be provided by a modular architecture and in-built extensibility and re-configurability.

A responsive Web client interface, implemented in DHTML and Javascript/Ajax, uses XMLHttpRequest to make HTTP requests and receive responses quickly without the delay of full page refreshes. The top right hand corner of the main page has the following links: Home | Manage Feeds | Menu | Register (| logout). When a user is logged in, his/her subscriptions, sorted by date, are shown in the main body of the Web page. The recommendations are shown on the right hand side of the screen. Search results appear below this. All results are displayed using the feed source's title and the titles of each sub-entry in that feed.

For recommendations we wanted a robust scheme that could handle a broad set of users, casual or dedicated/proactive. No assumptions were made about the information sources or the timeline of the information; for example, some types of information such as stock quotes and breaking news have a very

short time span whereas academic and entertainment materials are long lasting. A recommender could be tailored to work better in a more constrained domain.

We employed a simple implicit hybrid user/item-based collaborative algorithm based on (Wang *et al.,* 2006). A scheme based only on user ratings requires a large user community to be effective and has a bootstrapping problem; a pure item-based approach does not fully harness the social connections. The algorithm employs counting techniques to find a set of users, known as neighbours that have a history of agreement with the target user. A neighbour is defined as any other user who has twenty percent or more feeds in common. It then calculates the number of occurrences of each feed source in the neighbourhood set that is not in the user's subscription list. A weighted average of the occurrence data for the subscriptions for that subgroup is calculated where the weights are based on the inverse of the feed list sizes. The resulting function is used to recommend feeds for which the user has expressed no personal interest. Feed entries are indexed using the title text of the entry and any associated tags/keywords. Frequent low significance terms (stop words) are not included but stemming is not performed.

We also include a diversity feature in the recommendation engine. Recommender systems can lack diversity if based entirely on similarity algorithms (McSherry, 2002); evaluations have shown improved effectiveness of such techniques in terms of system precision/recall overall (Bridge and Kelly, 2006). In InterSynd a simple algorithm introduces items just outside the engine's initial list of top recommendations instead of some of the top recommendations. This includes items, differing from each other, which would not otherwise be recommended so as to broaden recommendations. The recommendation function is modified to factor in an adjustable diversity term, which is a measure of item specificity across all subscriptions; thus a positive bias is given to less common feeds that are partially relevant.

3.2 Feed Discovery

The system includes a feed discovery module called *Disco* that is separate from the main library. We addressed the problem of feed discovery by search. Search has become a central mechanism by which people access information on the Web. Conventionally search engines employ programs called crawlers or spiders to discover new Web documents. We employed the Open Search protocol (a9.com, Inc.) and Nutch (http://wiki.apache.org/nutch/), a free open source library for building search engines that supports OpenSearch. Nutch is an active project of the non-profit Nutch Organization with support from Yahoo! Research Labs, is widely used and has a large active community of users and developers.

OpenSearch developed by A9, an amazon.com, Inc. subsidiary, is a technology freely available under a Creative Commons license that enables search aggregation in a standardized format. Alternatives to OpenSearch are SRU, developed by the digital library community, and MXG, developed by a consortium of metasearch developers (LeVan, 2006). OpenSearch does not specify how a query should be formulated. A description record specifies only the location of the underlying search engine. Only a small number of general purpose search engines support OpenSearch, such as YaCy (www.YaCy.net), and mozDex (www.mozDex.com), but there are also topic-specific and desktop search applications that conform to OpenSearch. mozDex, implemented using Nutch, is the largest current deployment indexing 100 million pages supported by advertising revenue. OpenSearch 1.1 allows results to be returned in either RSS 2.0 or Atom 1.0 format. We format all results in the extended RSS format. More details on the feed discovery module are given in section 4.2.

3.3 InterSynd Architecture

Our software architecture, shown in Figure 2, is based on Sun's Java 2 Enterprise Edition (J2EE) thin client Web platform which provides technologies such as Servlets, the JSP (Java Server Pages) extension, and JavaBeans that can be composed in a modular fashion using the MVC (Model View Controller) Web architecture. Web MVC is a well-know architectural pattern used in large Web applications based on the classic MVC pattern developed at Xerox PARC for modeling the separation of data, presentation and control.

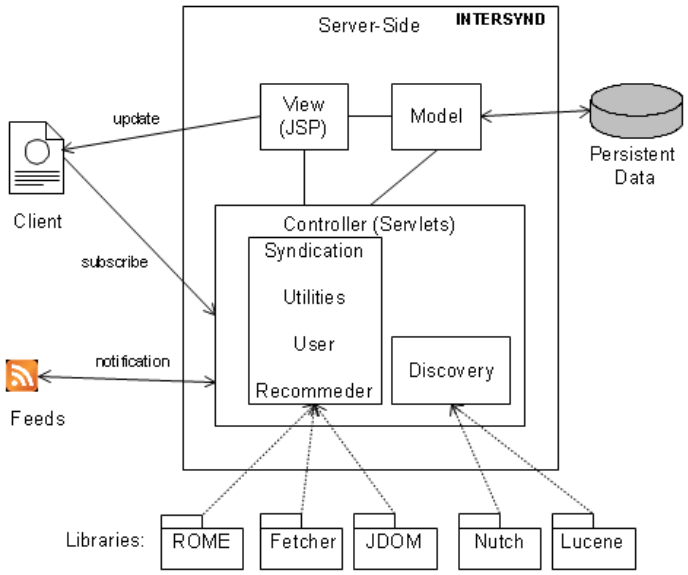

Figure 2. InterSynd Software Architecture

The extended Web server handles client requests and also sends the responses back. This is deployed using the free Apache Tomcat 6 server. JavaBeans encapsulate the Model, accessing a MYSQL database using the Java Database Connectivity (JDBC) API. The specifics of database access are left to the JDBC handler. JavaBeans are Java classes that follow predefined rules to allow standardized property access. The View of the application system displays the data retrieved from the model and is implemented in JSP. User events consist of HTTP post or get methods handled by the Controller. The Controller (a Servlet in a Web server container) consists of syndication, utilities, user, discovery (Disco), and recommender components. A single Servlet acts as a front controller for multiple views. The use of a Front Controller pattern, as for example in the Struts framework, promotes reuse (Alur *et al.*, 2003). In effect the same Web address is visible whether logging in, searching, or reading, indicating that the same Servlet serves all the generated pages. Session tracking is done with the HttpSession Interface.

4. InterSynd Implementation

Here we give detailed information related to the system implementation of the ROME library extension and the feed discovery subsystem. Section 4.1 gives an overview of the ROME software infrastructure and our particular deployment. Section 4.2 describes on the feed discovery module. All of the implementation technologies we employed are free open source software. We developed the software to be modular and have well-defined interfaces, facilitating the composition of software modules to create new applications.

4.1 ROME Infrastructure

ROME (RSS and atOM utilitiEs) is an open source Java library developed by Sun/Java.net and freely available under an Apache license for reading and publishing feeds in a format-neutral way. The API is lightweight, extensible, efficient, and also supports parsing of multiple feed formats. The design of ROME keeps the simplicity of the RSS format while handling all the leading Web syndication formats in an extensible Java-based abstraction.

ROME supports the Dublin Core (Beged-Dev *et al.,* 2000a) and Syndication (Beged-Dev *et al.*, 2000b) metadata element sets using Modules as in RSS version 1.0. You can define your own modules too. Note that RSS 2.0 and Atom use XML namespaces for extensibility. It defines a simple pluggable architecture for extensions; see the ROME documentation for details on the plug-in mechanism. ROME uses JDOM 1.0 (http://www.jdom.org) for parsing. Parsers and converters can be added or replaced as needed without any changes in ROME. ROME Fetcher is the module used for receiving feeds. We used ROME API version 0.9 and ROME Fetcher version 0.9. ROME itself consists of 120 interfaces and classes grouped into four packages. In ROME feeds are manipulated by implementing the SyndFeed interface, an abstract and idealized model of feeds (Johnson, 2004). Real formats such as Atom are referred to as *wire* feed formats. The use of SyndFeed, a format independent Java object, makes ROME independent of any particular syndication format. Any particular format (such as Atom) is converted into a generic SyndFeed. The default implementation is serializable. Any future XML formats could be implemented as plug-ins for ROME. ROME throws exceptions if the XML is ill-formed. SyndFeed and all implementations are Java beans and thus specify (serializable) properties in a standard way. Each SyndFeed contains a number of SyndEntries – data that will be utilised to display a feed Title, URI and Description at the very minimum. The SyndFeedInput class parses a feed (object of class SyndFeed) using its build method. The URI is wrapped using an XmlReader. XmlReader is a character-based reader that uses MIME types to resolve the encoding. Note that URIs are solely used to identify and access the underlying data.

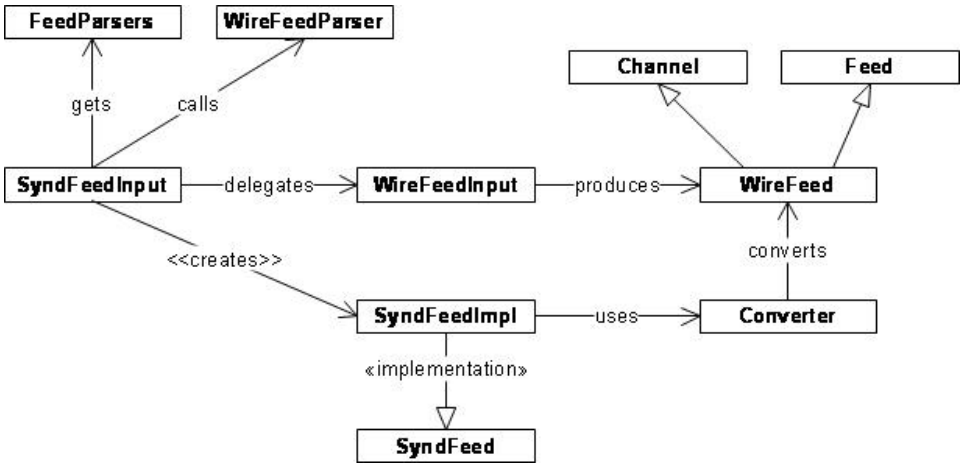

Figure 3. Feed Processing Mechanism

Figure 3 has an overview of how ROME processes feeds:

1. SyndFeedInput delegates to WireFeedInput to do the parsing. SyndFeedInput examines the syndication feed structure to determine the wire format.
2. WireFeedInput uses a FeedParser, which in turn employs JDOM to parse the feed into a WireFeed. If it is an RSS feed, the WireFeed is of class Channel and contains other elements defined in the package com.sun.syndication.feed.rss. If it is an Atom feed the WireFeed is of class Feed and assigns the values from the com.sun.syndication.atom package. The WireFeedInput returns a WireFeed either way.
3. SyndFeedInput creates a SyndFeedImpl from the WireFeed. SyndFeedImpl converts WireFeed (RSS or Atom format) to a SyndFeed (no particular format) and returns a SyndFeed.

The ROME Fetcher provides a simple means of retrieving feeds using HTTP conditional GET handling HTTP response codes (for example 404 Not found) including unrecognized HTTP response codes. It can be used with or without a cache. The HttpURLFeedFetcher class performs the actual HTTP request. The retrieveFeed method then creates a SyndFeed. We cached values using a Hash Map implementation (HashMapFeedInfoCache) for efficiency. Before the feed is retrieved from the source, it examines the last modified date. If this has changed it retrieves the feed, otherwise it ignores the unchanged content, saving bandwidth.

Following is a brief description of how the recommendations are generated in InterSynd. The Recommender cycles though the user sessions to determine who is logged in. A Common List of feeds is also generated. A SyndFeed object is constructed for each user. Recommendations in the form of lists of recommended feeds are produced for each user using the hybrid algorithm described previously in section 3. If there are not enough recommendations, the list is populated with the most popular feeds to a maximum of five. All recommended FeedIDs are converted to their specific URLs for retrieval purposes. Next feeds are retrieved using ROME's Fetcher and stored as an ATOM feed. All feed entries are stored and delivered to the client. The Fetcher is multi-threaded, and can fetch many feeds at once.

4.2 Feed Discovery Subsystem

The feed discovery subsystem (Disco) operates by means of Web search, the OpenSearch protocol and the Nutch Web search library. A restricted Web crawl is employed to source new information for users. Nutch builds on Lucene Java, a Java implementation of Apache's text search engine, adding Web-specifics, such as a crawler, a link-graph database, and parsers for HTML. We deployed Nutch version 0.9.

An important practical consideration with search is response time; users do not want long waits for results to be returned. In our current implementation each search is run as a separate thread. The search engine has a crawler which discovers and fetches Web pages. These are generated by a fetcher component and stored in a custom database called WebDB, containing known URLs and fetched page contents. WebDB is a specialized persistent data structure that mirrors the structure and properties of the Web graph being crawled where nodes are pages and edges are links. Note that WebDB is only used by the crawler and does not play any role during searching. WebDB stores information on Web pages such as the title, URL, a hash of the contents, the number of links in the page, and fetch information (such as when a page is due to be re-fetched). A *fetchlist* is generated from WebDB based on these details. All fetching is *polite*, observing the Robots Exclusion Protocol for Web crawlers. Fetching is also polite in

the sense that all the links in a single Website are added to a single fetchlist preventing unnecessary polling of a Web server.

Web pages are analyzed for all outgoing links which are placed on a queue by the fetcher for further crawling. A parameter *topN*, representing the number of links to extract from any one page, is set to 400. We used a shallow depth of ten for link following since we were interested in content at or near certain Web portals. The portal sites are listed in a configuration text file and can be easily changed at any time. The indexer (based on the Lucene index) dissects pages and builds keyword-based indexes from them. The set of sites to be crawled are stored in a Nutch configuration file. Automatic re-crawl is set at the default of thirty days. The Nutch crawl command employed is:

```
>bin/nutch crawl urls -dir crawl -depth 10 -topN 400
```

The average runtime of one of these crawls on a single desktop PC is about 100 minutes and resulted in an index of about 500MB. At present we restrict the crawler to a set of blog sites and news and blogging Web portals using Nutch's PrefixURLFilter facility. While re-crawling, a URL already in the database will not be injected again.

A segment is a collection of pages fetched and indexed by the crawler in a single run. The full inverted index is created by merging all of the individual segment indexes. Lucene tools and APIs were available to interact with this generated index. Lucene is well suited to small development projects because it indexes incrementally using minimum memory. The Nutch dedup command is run to eliminate duplicate documents across segments and the Nutch index is joined to the Web server extension (and stored in the ./crawl directory relative to the Tomcat start directory). Access to the index is by means of a Nutch JavaBean, called the NutchBean, added to the Servlet container. Further technical details such as server configuration and timeout limits are not given here.

When searching for the first time in a session the NutchBean object opens the index it is searching against in read-only mode. The Nutch query is translated into an optimized Lucene query to carry out a regular Lucene search. A Nutch Hits object represents the top matches for a query and each result is an XML item element. The results are in the OpenSearch RSS 1.0 format, the 'bare minimum of additional functionality required to provide search results over RSS channels' (quoted from A9 Website). The OpenSearch namespace is specified in the opening <rss> XML element. The only new OpenSearch elements are OpenSearch:totalResults, OpenSearch:itemsPerPage and OpenSearc:startIndex. The results also contain Nutch specific elements that we do not use, such as nutch:site, nutch:cache, nutch:explain, and nutch:boost and stardard RSS elements for enhanced semantics. Ranking in Lucene uses a combination of content-based ranking (based on tf.idf) and PageRank style link-analysis and can be tweaked (Khare *et al.*, 2005).

All the user logins are stored in a user_subscriptions MYSQL table. Feeds get added to and returned from the data store by generating SQL SELECT or UPDATE statements. Figure 4 shows the entities and relationships of the data in an Entity Relationship diagram. The global_feed_list and user_details only contain unique entries dictated by the primary keys FeedID and login, respectively. The tables user_subscriptions and user_preferences can contain many entries as a user can subscribe to more than one feed and can have more than one preference. Preferences are not used in the current version but could allow for more detailed personalisation.

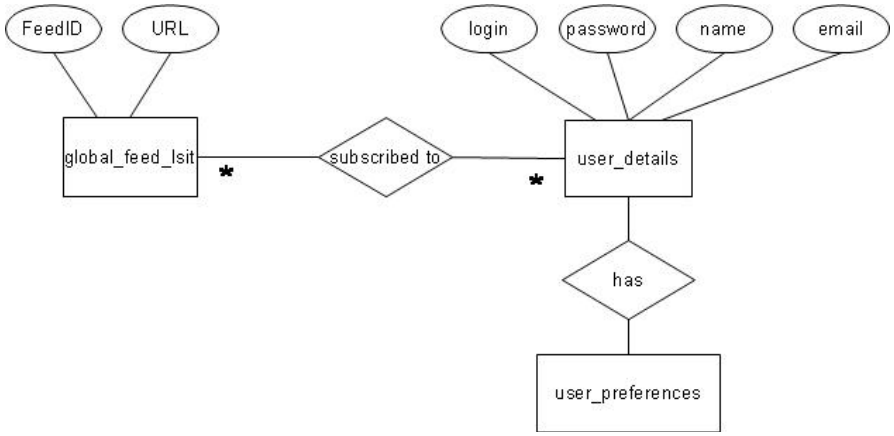

Figure 4. ER Diagram of Database Tables

4.3 Measurement of Modularity

A key objective of this work was to examine the modularity of the new software and measure the extent to which we could build a library and application by re-using free open source code. This section presents summary software metrics for the code base and comments on these issues.

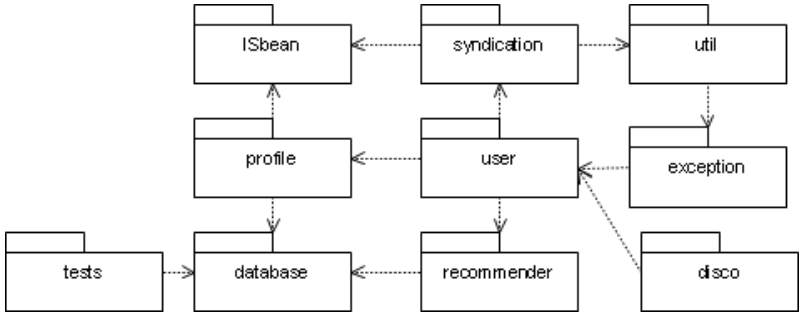

Figure 5. Package Diagram of InterSynd Implementation

Figure 5 shows how InterSynd's classes are partitioned into namespaces in a UML 2.0 structural model. Table 1 below gives summary metrics on the number of lines of code for this implementation. Note that these numbers also exclude library code which is part of ROME, ROME Fetcher, Nutch, or JDOM.

Package	*#Lines of Code*
User Interface	88 java; 1069 jsp
bean	202 java
database	660 java
exception	26 java
recommender	171 java
syndication	756 java
user	434 java
util	368 java
disco	412 jsp
total:	*2705 java; 1481 jsp*

Table 1. InterSynd Code Summary Data

In Table 2 below we give data related to various software quality metrics for InterSynd and also ROME and ROME Fetcher for comparative purposes. Since a great deal of project effort went into preserving the clean design and extensibility of ROME these measures are evidence that this is achieved. The metrics employed are commonly used simple object-oriented measures such as number of classes, number of method invocations and method complexity. The Average Complexity of InterSynd (2.01) was very similar to ROME (1.96) and below that of ROME Fetcher (2.17) where lower values are more desirable.

Library	*Files*	*Lines*	*Statements*	*Percentage Branches*	*Calls*	*Classes*	*Methods /Class*	*Avg. Stmts. /Method*	*Avg. Depth*	*Avg. Complex*
InterSynd	35	2,705	1,372	9.5	816	35	4.49	5.85	1.9	2.01
ROME 0.9	117	22,884	8,291	12.4	5,214	120	12.1	3.53	1.85	1.96
fetcher 0.9	14	1,632	614	11.1	313	17	5.71	3.75	1.85	2.17

Table 2. Library Metrics Summary Data

Table 3 provides more detail on the ten more complex classes in InterSynd based on the average method complexity. Note that some of these are small classes.

File Name	*Lines*	*Stmts.*	*Percent Branches*	*Calls*	*Methods /Class*	*Avg. Stmts. /Method*	*Avg. Complexity*
user\NewUserServlet.java	108	54	14.8	62	4	10.75	5
util\OrderByDate.java	32	15	20	3	1	9	5
syndication\FetcherEventListenerImpl.java	28	12	25	10	1	7	4
syndication\AddFeedServlet.java	105	49	18.4	30	4	9	3.75
database\FeedHandler.java	241	165	15.2	98	11	12.73	3.27
ajax\UserValidateServlet.java	89	62	11.3	40	5	8.8	3.25
recommender\RecommendMeServlet.java	148	80	7.5	62	4	14.25	2.75
user\LoginServlet.java	103	49	12.2	39	4	9.25	2.75
database\RecommenderHandler.java	149	97	13.4	55	9	8.56	2.56
syndication\AlternativeDisplayServlet.java	119	68	8.8	54	4	11.75	2.5

Table 3. Complex Classes Metrics

Some of the metrics warrant further explanation. The number of lines of code measure excludes all XHTML and JSP, i.e. it refers to Java code only. The number of classes metric measures all Java classes and interfaces including inner classes and anonymous inner classes. The Average Statements per Method measure excludes comment lines and blank lines. Complexity is measured as defined by McConnell (McConnell, 1993), a metric based on the number of execution paths in a method based on McCabe's classic *cyclomatic complexity* measure. Switch statements add to the complexity count for each exit from a case; each catch after a try block (but not the try or finally statements) also adds to this measure. Average Depth is the average of the maximum block level depth found in the methods. Any {} pair is considered a block. Percent Branches measures the ratio of statements that cause a break in the sequential execution of statements to those that do not. In the Calls metric all method calls are counted, those in statements as well as those within expressions.

5. Related Work

The most relevant emerging technologies and related work with respect to InterSynd are now described.

5.1 Applications

Several projects involving both RSS and recommendation have been published recently. Chen *et al.* (2007) developed a recommender system for personalized advertising in a RSS reader application. This system creates private dynamic user profiles.

Celma *et al.* (2005) describe a collaborative music recommendation system called *Foafing the Music* based on social connections where information is extracted from music related RSS feeds. The authors use the Friend of a Friend (FoaF) concept from social networking, RSS vocabularies, and content-based descriptions to generate music recommendations from music related feeds. A common ontology (specified using OWL) describes the music domain and content is specified in the XML-based XSPF playlist format. Similarity is computed with the aid of queries over this ontology. Like InterSynd, various RSS formats and Atom are supported in the fetching stage; RSS 1.0 is used for results. This system differs from InterSynd in a number of regards. First, Foafing the Music is constrained to the music domain. Second, it uses semantic information about music as well as indexing RSS tags. Kobayashi and Saito (2007) describe an RSS-based information recommender that does not depend on user profile data, but on topical news information. Jun and Ahamad (2006) describe a feed exchange system, where hosts can exchange feeds with similar neighbours. This is based on a distribution overlay network and is aimed at tackling the problem of scalability through collaboration (as is InterSynd) but by means of a different mechanism. The system they developed, called FeedEx, also supports recommendations using a similarity-based algorithm. Webster and Vassileva (2007) describe an RSS recommender system called KeepUP but the paper does not state which version or versions of RSS are supported. KeepUP is based on negotiation in an implicit social network. All these projects only cater for RSS feed formats.

5.2 Implementation Alternatives

The main Java alternative to ROME is Informa (http://informa.sourceforge.net), a software library that predates ROME. While Informa is complicated and does not support RSS 2.0 output or Atom 1.0, it does have other interesting features such as good support for persistence (Hibernate and Castor) and search (Lucene). In addition Informa has good support for server-side development with JSP taglibs. Programming with lower-level APIs such as JDOM, JAXB, or SAX2 is another alternative, but would involve reinventing functionality already in ROME and Informa.

While the Atom and RSS syndication formats dominate on the Web, other XML-based formats for syndication exist, such as ICE. ICE is a powerful heavyweight protocol that aims to ‘automate the scheduled, reliable, secure redistribution of any content’ (Brodsky *et al.*, 2003). TwICE is a Java implementation (also hosted on Sourceforge code repository). ICE (Information and Content Exchange) aims to automate the establishment of syndication relationships as well as deal with data transfer and results analysis. This makes ICE more powerful but heavier than simple Web syndication. ICE gives content providers more control over delivery, schedule and reliability without deploying a full-scale Web services solution. Compatible technologies include PRISM (http://www.prismstandard.org) - a discovery mechanism for content to syndicate - and the Channel Definition Format (CDF) (Ellerman, 1997) for

push channels. In contrast, InterSynd has a feed discovery module but currently no support for push; see section 6.

Feedsync, formerly known as Simple Sharing Extension, developed at Microsoft and available under a Creative Commons License extends the RSS and Atom feed formats to enable the aggregation of information by the 'bi-directional, asynchronous synchronization of new and changed items amongst two or more cross-subscribed feeds' (Ozzie, 2007). Feedsync has been implemented as a ROME module and is managed as a ROME subproject. Another Microsoft technology, the Windows RSS Platform, is an API for applications developers to access feeds and subscriptions supported by Internet Explorer 7. The platform includes a data store, a sync engine, and a feed list. The constituent Feed Download Engine downloads feeds and merges them into a feed store. Windows RSS is callable from both .NET and unmanaged code. Abdera (http://abdera.apache.org), an Apache incubator project contributed by IBM, is a STAX-based Atom parser written in Java. It supports many Atom extensions, AtomPub, Atom to JSON, but has no RSS support as yet.

Yahoo! Pipes (http://pipes.yahoo.com/pipes/) and Microsoft's Popfly (Griffin, 2008) are Web applications that provide a graphical editor for non-programmers to create mashups including the aggregation of feeds. Users write rules for how that content should be filtered. For instance, Yahoo! Pipes consists of a connection graph of operations, such as text extractors, filters, and feed display. The mashups can be deployed as Web applications or Web services. Unlike free open-source systems such as the ROME Fetcher module, Yahoo! does not allow unrestricted feed access by policy. The Google Mashup Editor also allows users to retrieve, combine, and display RSS feeds in a similar way. Apache Cocoon (http://cocoon.apache.org) is a framework based on the notion of component pipelines that can be used for a different Web application development tasks including Web publishing and search (using Lucene).

Google Data API or GData (http://code.google.com/apis/gdata/) is a an HTTP-based protocol proposed by Google that combines XML-based syndication formats (Atom and RSS) with a feed-publishing system based on the Atom Publishing Protocol. GData, unlike RSS and Atom, has some support for queries and updates. Any Web service can provide a GData feed. It also extends AtomPub for authentication, batch requests, and supports an alternate format to XML (JSON).

Here we briefly mention some proposals for standardizing the mark-up of subscription lists and user interests. APML (Attention Profiling Mark-up Language) (http://www.apml.org) is an XML-based format supported by Bloglines that can be used to store a user's interests. This protocol is not widely supported at present. XOXO (http://microformats.org/wiki/xoxo/) enables publishing of outlines and blogrolls where XOXO outlines are defined as hierarchical, ordered lists. OPML (Outline Processor Markup Language) (http://opml.org) is a proposed standard for the exchange of lists of feeds between Web feed aggregators. Simple List Extensions (Lyndersay, 2006) specifies extensions of existing formats, such as RSS 2.0 and Atom, which allow the definition of ordered collections of entries where this is useful.

6. Summary and Future Work

This paper describes middleware for developing applications that transparently use feed technologies like RSS and Atom. A collaborative recommender component and a feed discovery module were included. The goal of this project was not to develop completely new software from scratch, but

rather to base our work on already existing free and open systems and standards. This we achieved with an extensible library implementation and a fully operational client application.

A review of the background material and related work shows that a number of efforts are underway to process feeds with greater ease and flexibility. There is also considerable interest in applying ideas from Web search and collaborative filtering in this area. To demonstrate our extension of ROME a simple feed interchange application called InterSynd was developed. This is open source and extensible in itself and will hopefully serve as the basis for future work.

It remains to use InterSynd with an existing fully-featured feed reader such as Thunderbird. This is an area of current work. Another improvement would be to allow the import and export of subscription lists as OPML. The ROME Aqueduct-Prevayler module (Java.net, 2006) or some other mechanism could be included to enable persistence. InterSynd currently doesn't process multimedia content. This could be added because ROME (as of version 0.9) supports enclosures and the mediaRSS module. Another area for future work is privacy issues that were not given much consideration as they lay outside the scope of the project goals. Privacy is an important issue if Web 2.0 technologies such as Web syndication are to be deployed in critical applications.

Here we suggest some technical improvements that should lead to better performance. Feed fetching is done by InterSynd for updating subscriptions and recommendations as well as in Disco for receiving search results. The code could be refactored to share the basic fetching mechanism such as the establishment of connections. Support for ping servers is a feature that is lacking. Allowing XML-RPC updates would reduce the amount of polling done by InterSynd for existing subscribers. All storage could be switched from a MySQL relational database to the MapReduce platform to support a distributed and/or parallel server implementation. Nutch can use the Hadoop APIs (http://hadoop.apache.org) for data intensive distributed applications, a technology based on Google's MapReduce that enables the reliable storage of large files. Hadoop is also open source software managed as an Apache project.

The emphasis of this work has primarily been of practical issues such as system architecture, build and deploy considerations, application of free open source software and system extensibility. Tests were undertaken to validate the software design and implementation both through exposition and evaluation by appropriate software engineering metrics. In terms of the algorithms used for recommendation, search and crawling, every effort was made to utilize state-of-the art methods that have been empirically tested in the literature.

7. Acknowledgments

We would like to thank our colleagues David Murphy, Fergal Lane and the reviewers for feedback on this work.

References

A9.com, Inc. "OpenSearch RSS 1.0 specification". Retrieved 1 July, 2010 from http://a9.com/-/spec/opensearchrss/1.0/

Alur, D., Crupi, J. & Malks, D. (2003) *Core J2EE patterns: best practices and design strategies (2nd ed.)* (Englewood Cliffs, NJ: Prentice Hall).

Andreesen, M. (1999) "Innovators of the Net: Ramanathan V. Guha and RDF". *Netscape Techvision*. Retrieved 1 July, 2010 from http://wp.netscape.com/columns/techvision/innovators_rg.html

Atom wiki (2008) "RSS 2.0 and Atom 1.0 compared". Retrieved 1 July, 2010 from http://intertwingly.net/wiki/pie/Rss20AndAtom10Compared/

Beged-Dev, G. *et al.* (2000 A) "RDF Site Summary 1.0 Modules: Dublin Core". Retrieved 1 July, 2010 from http://web.resource.org/rss/1.0/modules/dc/

Beged-Dev, G. et al. (2000 B) "RDF Site Summary 1.0 Modules: Syndication". Retrieved 1 July, 2010 from http://web.resource.org/rss/1.0/modules/syndication/

Breese, J. S., Heckerman, D. and Kadie, C. (1998) "Empirical analysis of predictive algorithms for collaborative filtering". In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 43-52.

Bridge, D., Kelly, J.P. (2006) "Ways of computing diverse collaborative recommendations". In *Adaptive Hypermedia and Adaptive Web-based Systems*, edited by V. Wade, H. Ahsman and B. Smyth, Springer, pp. 41-50.

Brodsky, J. *et al.* (Eds.). (2003) "The Information and Content Exchange (ICE) protocol". Working Draft, Version 2.0. Retrieved 1 July, 2010 from http://xml.coverpages.org/ICEv20-WorkingDraft.pdf

Carzaniga, A., Rosenbloom, D.S., and Wolf, A.L. (2001) "Design and evaluation of a wide-area event notification service". *ACM Transactions on Computer Systems*, **19**(3): 332-383.

Celma, O., Ramirez, M., and Herrera, P. (2005) "FOAFing the music: A music recommendation system based on RSS feed and user preferences". In *Proceedings of the 6th International Conference on Music Information Retrieval* (London, UK, 11-15 Sept. 2005).

Chen, T., Han, W.-L., Wang, H.-D., Zhou, Y.-X., Xu, B., and Zang, B.-Y. (2007) "Content recommendation system based on a private dynamic user profile". In *Proceedings of the 6th International Conference on Machine Learning and Cybernetics* (Hong Kong, 19-22 Aug., 2007).

Chowdhury, G.G. and Chowdhury, S. (2007) *Organizing information from the shelf to the Web* (London: Facet Publishing).

Dittrich, K.R., Gatziu, S. and Geppert, A. (1995) "The Active Database Management System manifesto: A Rulebase of ADBMS Features." *Lecture Notes in Computer Science* ***985***, Springer.

Ellerman, C. (1997) "Channel Definition Format (CDF)". Retrieved 1 July, 2010 from http://www.w3.org/TR/NOTE-CDFsubmit.html

Fielding, R. (2000) *Architectural styles and the design of network-based software architectures*. Unpublished doctoral dissertation, University of California, Irvine, CA.

Fischer, G. and Stevens, C. (1991) "Information access in complex poorly-structured information spaces". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New Orleans, Louisiana, United States), pp. 63-70.

Griffin, E. (2008) *Foundations of Popfly: rapid mashup development* (Berkeley, CA.: Apress Inc.).

Goldberg, D., Nicols, D., Oki, B.M. and Terry, D. (1992) „Using collaborative filtering to weave an information tapestry". *Communications of the ACM*, **35**(12): 61-70.

Hammersley, B. (2003) *Content syndication with RSS* (Sebastopol, CA, USA: O'Reilly Media).

Harvard Law School (2003) "RSS 2.0 Specification". Permanently hosted at Berkman Center for Internet & Society, Harvard Law School, http://blogs.law.harvard.edu/tech/rss

Nottingham, M. (Ed.) (2005) "IETF Network Working Group. The Atom syndication format. RfC 4287". Retrieved 1 July, 2010 from http://www.ietf.org/rfc/rfc4287

Java.net. (2005) "ROME: RSS/Atom syndication and publishing tools". Retrieved 1 July, 2010 from https://rome.dev.java.net

Java.net (2006) "ROME Aqueduct-Prevayler implementation". Retrieved 1 July, 2010 from http://wiki.java.net/bin/view/Javawsxml/RomeAqueductPrevayler

Jhingran, A.D., Mattos, N. and Pirahesh, H. (Eds.) (2002) "Information integration special issue". *IBM Systems Journal*, **41**(4).

Johnson, D. (2004) "How ROME works". Retrieved 1 July, 2010 from http://rollerweblogger.org/emtry/how_rome_works

Jun, S. and Ahamad, M. (2006) "FeedEx: collaborative exchange of news feeds". In *Proceedings of the 15th International ACM Conference on World Wide Web*.

Khare, R., Cutting, R.D., Sitaker, K. and Rifkin, A. (2005) "Nutch: a flexible and scalable open-source web search engine" CommerceNet Labs Technical Report 04-0, May 10, 2005. www.commercenet.com/images/0/06/CN-TR-04-04.pdf

Klyne, G. and Carroll, J.J. (2004) "Resource Description Framework (RDF): concepts and abstract syntax". World Wide Web Consortium specification. Retrieved 1 July, 2010 from http://www.w3c.org/TR/2004/REC-rdf-concepts-20040210/

Kobayashi, I. and Saito, M. (2007) "A study on an information recommendation system that provides topical information related to user's inquiry for information retrieval". *New Generation Computing*, **26**(1).

Kuman, R., Novak, J., Raghavan, P. and Tomkins, A. (2004) "Structure and evolution of Blogspace". *Communications of the ACM*, **47**(12), 5-9.

LeVan, R. (2006) "OpenSearch and SRU: a continuum of searching". *Information Technology and Libraries*, Sept. 2006. Retrieved 1 July, 2010 from www.oclc.org/research/publications/library/2006/levan-ital.pdf

Liu, H. Ramasubramanian, V. and Siren E.G. (2005) "Client behavior and feed characteristics of RSS, a publish-subscribe systems for Web micronews", *USENIX IMC*. Retrieved 1 July, 2010 from www.cs.cornell.edu/People/egs/papers/rsssurvey.pdf

L-Soft (1996) "History of LISTSERV". Retrieved 1 July, 2010 from http://www.lsoft.com/products/listserv-history.asp

Luhn, H.P. (1958) "A business intelligence system". *IBM Journal of Research and Development*, **2**(4):314-319.

Lyndersay, S. (2006) "Simple List Extensions specification, Version: 1.0a". Retrieved 1 July, 2010 from http://msdn.microsoft.com/en-us/xml/bb190612.aspx

Malone, T.W., Grant, K.R., Turbak, F.A., Brobst, S.A. and Cohen, M.D. (1987) "Intelligent information-sharing systems", *Communications of the ACM*, **30**(5), 390-402.

McConnell, S. (1993) *Code complete: a practical handbook of software construction* (Redmond, WA: Microsoft Press).

McSherry, D. (2002) "Recommendation engineering". In *Proceedings of the 15th European Conference on Artificial Intelligence* (Lyon, France, July 2002) edited by van Harmelen, F., pp. 86–90.

O'Neill, E.K. (1991) "Selective dissemination of information in the dynamic web environment", Unpublished MSc thesis, University of Virginia. Retrieved 1 July, 2010 from www.cs.virginia.edu/~cyberia/papers/eko_thesis.pdf

O'Reilly, T. (2005) "What Is Web 2.0: design patterns and business models for the next generation of software". Retrieved 1 July, 2010 from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

Ozzie, J. *et al.* (2007) "Simple Sharing Extensions for Atom and RSS, Version: 0.93". Retrieved 1 July, 2010 from http://msdn.microsoft.com/en-us/xml/bb510102.aspx

Packer, K.H. and Soergel, D. (1979) „The importance of SDI for current awareness in fields with severe scatter of information". *Journal of the American Society for Information Science*, **30**(3), 125-35.

Sandler, D., Mislove, A., Post, A. and Druschel, P. (2005) "Feedtree: sharing Web micronews with peer-to-peer event notification". *Peer-to-Peer Systems IV: Lecture Notes in Computer Science **3640***, 141-151, Springer.

Sparck-Jones, K. (1972) "A statistical interpretation of term specificity and its application in retrieval". *Journal of Documentation*, **28**(1), 11-21.

Rosenblum, D.S. and Wolf, A.L. (1997) "A design framework for internet-scale event observation and notification". In *Proceedings of the 6th European Software Engineering Conference: Lecture Notes in Computer Science **1301***, edited by Jazayeri, M and Schauer, H., pp. 344-360.

Wang, J., deVries, A. and Reinders, M. (2006) "Unifying user-based and item-based collaborative filtering approaches by similarity fusion". In *Proceedings of ACM SIGIR*, pp. 501-508.

Webster, A. and Vassileva, J. (2007) "KeepUP recommender system", In *Proceedings of Recommender Systems*, October 19-20, Minneapolis, MN, USA.

Yan, T. and Garcia-Molina, H. (1995) "SIFT - a tool for wide area information dissemination". In *Proceedings of the USENIX Technical Conference*, 177-186.