國立交通大學

資訊科學與工程研究所

碩士論文

一個方便應用程式使用 RSS/Atom 的中介軟體
An Easy-To-Use Feed Middleware for Application Development with RSS/Atom Feeds

研 究 生：脫志曜

指導教授：袁賢銘　教授

中華民國　九十六年七月

一個方便應用程式使用 RSS/Atom 的中介軟體
An Easy-To-Use Feed Middleware for Application Development with RSS/Atom Feeds

研 究 生：脫志曜　　Student：Chi-Io Tut

指導教授：袁賢銘　　Advisor：Shyan-Ming Yuan

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

一個方便應用程式使用 RSS/Atom 的中介軟體

學生：脫志曜　　　　　指導教授：袁賢銘

國立交通大學資訊科學與工程研究所

摘要

RSS 和 Atom 網摘是用可擴展置標語言來呈現經常更新的網頁中的項目的一種形式，這種形式讓使用者可以透過網摘閱讀器來訂閱聯播的內容。隨著網摘因爲部落格的普及而越來越受採用，各式各樣讓網摘包含更多語意資訊的延伸方式被提出來，只把網摘當作網路上簡單的可擴展置標語言文件來看待的普通工具不足以用來開發應用程式。

本論文提出一個幫助開發 RSS/Atom 相關應用程式的中介軟體，它爲開發者取得、解析和儲存網摘，並提供一套方便使用的介面讓開發者可以編寫程序化或事件觸法式的應用程式。對比起視窗 RSS 平台，此中介軟體較具有彈性也較容易使用。當換去具工業強度的資料庫和伺服器，它可以比組織用來解決現實世界中的整合問題。

An Easy-To-Use Feed Middleware for Application Development with RSS/Atom Feeds

Student: Chi-Io Tut Advisor: Shyan-Ming Yuan

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

RSS and Atom feeds are XML representation of the entries in frequently updating websites, which enable users to subscribe to those syndicated contents using feed readers. As feeds are gaining more and more adoptions due to the ubiquity of blogs, various extensions are written for them to carry more semantic information. Ordinary tools which treat them as simply XML documents on the web are not sufficient for application development.

In this paper, a middleware is proposed to aid application development involving RSS/Atom feeds. It handles fetching, parsing and storage of feeds for developers and provides them with a set of easy-to-use interfaces to write procedural and event-driven applications. Compared with the Windows RSS Platform, it is more flexible and easier to work with. When extended with industrial-strength databases and servers, it can be use by an organization to solve real world integration problems.

Acknowledgements

首先我要感謝袁賢銘教授給我的指導，在我的研究領域裏給我很多的意見，並且給多我最大的空間來發揮我的創意。也感謝所有幫助我的學長葉秉哲、邱繼弘、吳端祥、鄭明俊，在我研究的過程中給我不少的指導跟建議。還有感謝實驗室同學宋牧奇、彭品勻、尤喜夫、熊家媛跟蔡宗穎，在這一年來跟我一起做了很多研究討論，也激盪出不少的想法。也感謝我的心愛女友楊婷詠，在我最無力的時候給我打氣，讓我一直有動力來完成我的研究。最後我要感謝我的爸媽和妹妹，給予我這個良好的環境讓我求學生涯毫無後顧之憂，專心於學業，謹以這篇小小的學術成就來感謝您們的養育之恩。

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Preface

During the past few years, we have been experiencing a transition from Web 1.0 to the so-called Web 2.0 era. Instead of being an accurate specification, Web 2.0 is only a collective concept describing the technical features and social behaviors of some of the famous Web 2.0 websites [1]. Among them, there are blogs and feeds. A blog is a website with reverse chronologically-ordered entries, usually news or diaries. A feed is the XML-formatted content of them. These technologies combined enable the publish/subscribe semantics of the web and transform it from a static web into a "live web".

This paper proposes an easy-to-use feed middleware for application development involving RSS/Atom feed technology. The following sections explain why such a system should be built, what capabilities it should have, the problems to face, and their respective solutions.

1.2 Motivation

According to the quarterly report of Technorati, one of the leading blog search engines, the number of blogs they are tracking is 70 million, and it keeps growing rapidly [2]. Nowadays, almost all frequently updating websites have feeds. There are two reasons for this fast adoption. First, there are many blog hosting services providing blogging tools with easy-to-use editor for users to publish their thoughts without having to understand any HTML at all, which allow users to focus on creating more and more contents. Second, both server and client side feed readers are available for users to read those contents on many different blogs without having to check them out one by one constantly. However, for programmers, there are no easy-to-use tools for them to develop applications based on the underlying feed technology.

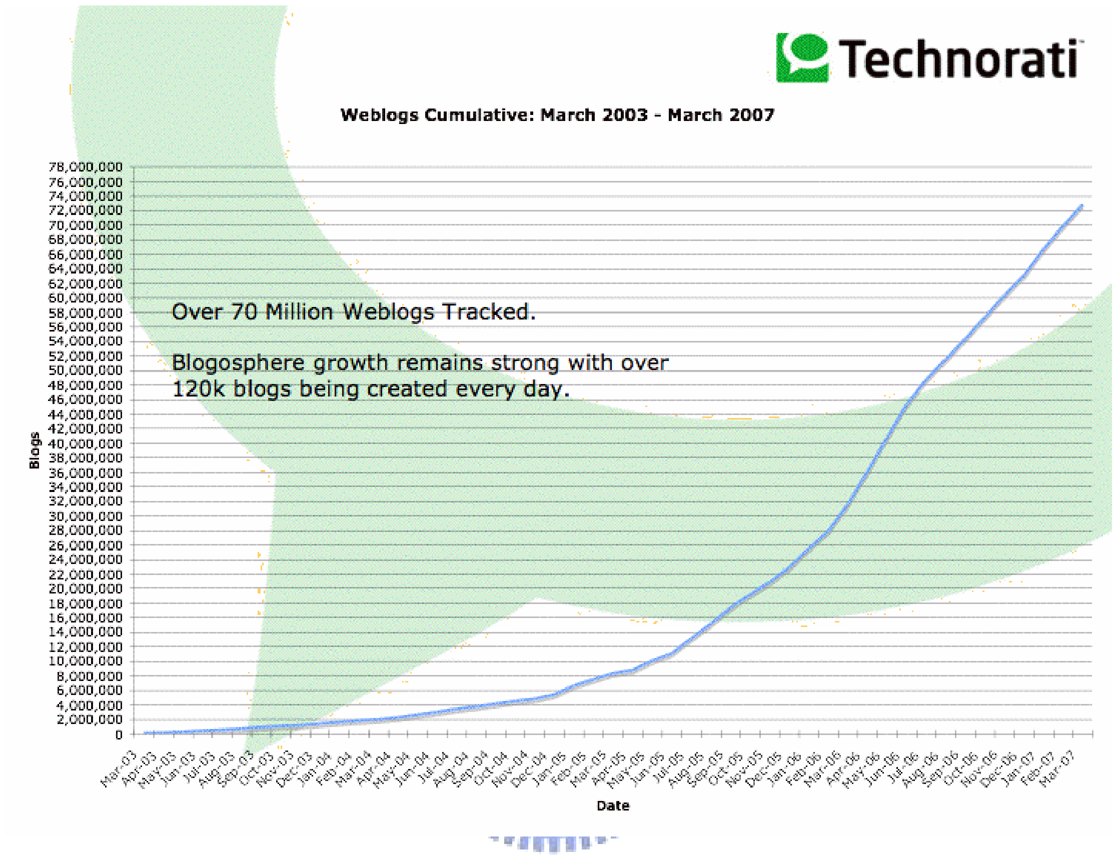


Figure 1-1 Technorati is now tracking over 70 million weblogs

Moreover, since feeds are XML documents, they can be extended to include other semantic data besides titles and descriptions of news, such as stock quotes, weather forecasts, and multimedia resources. There are already many existing extension specifications for feeds [3]. Besides, microformats [4] are also embedded in many feeds to add extra semantics. Regardless of being formatted as RSS or Atom, feeds are the transitional objects from the web of documents to the web of data. Since it is in widely used today to hold semantic data, we must have better tools to manipulate them before we actually turn into the Semantic Web era [5].

For enterprises, there is a growing need for feed technology because they produce and consume a large amount of information every day. While emails are filled with spam and portals are hard to integrate with, internal systems begin to use feeds as the data carrier for information. For example, bug reports, software updates and code revisions are good candidates to be unified using feed formats. There are already enterprise solutions for feed subscription and reading in heterogeneous environment, but tools for integration and application development are yet to be built.

1.3 Objectives

The most important things for application developers are APIs, the interfaces to interact with a library or another system. For feeds, two sets of interfaces should be provided. One of them is a pulled-based interface for procedural applications. The other one is a pushed-based interface for event-driven applications.

Feeds provided by websites usually contain only about 10 to 40 of the most recently updated data. But for applications to do some significant things, they may need more than that. Therefore, outdated feeds must also be available for applications to retrieved. Besides, feeds from different content sources may be of the same interest to some specific applications, retrieving them one by one and mixing them manually is a tedious task. It is better to have a way to specify a number of feed sources and then get the entries of all of them.

Last but not least, the resulting tool should be platform and language neutral since different systems often employ different technologies and they will keep on changing. Attributes such as simplicity and extensibility are very much desired because a simple and open tool means a bigger chance of being integrated with existing systems and greater possibility to be put into practical use.

1.4 Problems and Solutions

One of the problems of feed technology is formats. There are two families of feed formats, RSS and Atom. For RSS, there are nine incompatible versions. Atom, on the contrary, is an IETF-backed standard format [6]. Although RSS 2.0 and Atom 1.0 are the most prevalent ones, many of them are still heavily in use. But thanks to the open source community, there are already some good feed parsers available. The problem left is to choose a suitable one.

There are two issues to be solved in order to keep outdated feed available for applications to retrieve: bandwidth and storage. Although feeds enable the publish/subscribe model of the web, the underlying technology is polling, i.e., clients have to keep asking for the same feed to see if there is an update. Therefore, various HTTP caching, conditional retrieval, and compression techniques must be implemented [7]. To store a large number of feed entries with frequent updates and retrieval, a database with efficient caching mechanism is the simple answer. We will discuss more in details on the chapter of implementation.

After feeds are parsed and stored, the final problem left is to expose an interface for others to use. Since the proposed tool is positioned as a middleware instead of simply a set of library functions, REST and XML-RPC are used as pull and push interfaces respectively. The reason for this choice is that they are simple and every major language has good implementations for them.

2 Background and Related Works

2.1 Background

2.1.1 Blogging and Syndication

Blog is the combination of the two words, ‘web’ and ‘log’, meaning to write chronologically on the web [8]. The blogging phenomenon started at late 90’s and took off around 2000, when hosted blogging platforms became widely available. Today, blogs are so ubiquitous and influential that some high profile blogs have more visitors than many main stream media websites. Besides the chronological nature, a blog is a special type of websites with some more technical characteristics. First, every entry can be access by a unique URL – Permalink. Second, a blog provides a feed of recently added content for others to subscribe – Syndication, which revolutionizes user experiences of the web by shifting the task to check websites for updates from users to the machines via unified and machine-understandable representations of those websites.

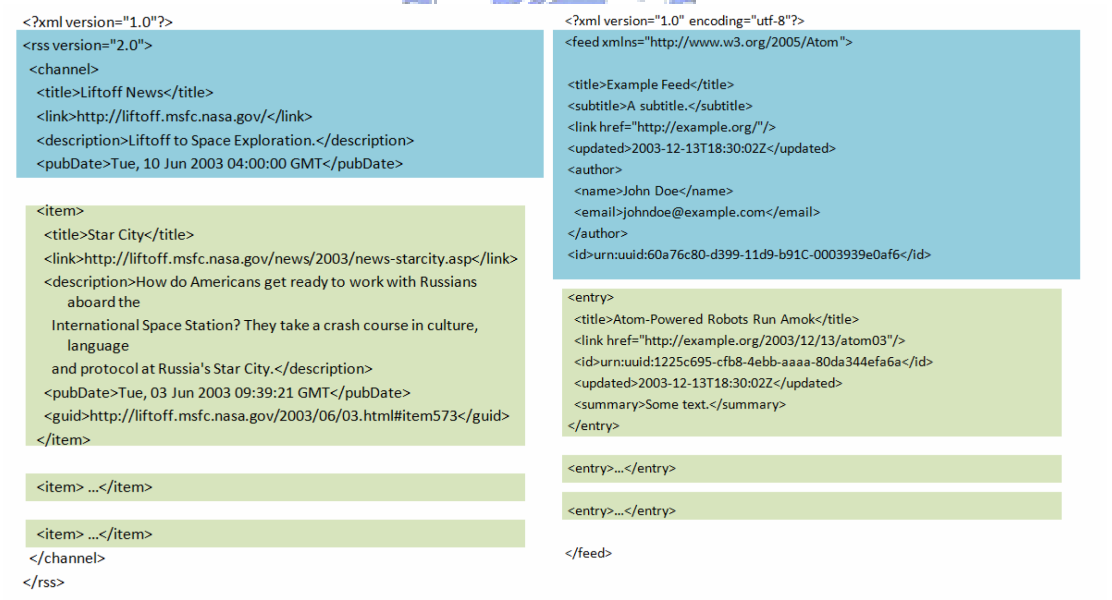
2.1.2 Feed Formats

Throughout this paper, the term feed refers to both the RSS and Atom XML feed formats. RSS (RDF Site Summary) is originally created by Netscape to describe news stories in RDF (Resource Description Framework), which in turn is defined using XML (eXtensible Markup Language). That version is known as RSS 0.9. After minor modifications to remove RDF elements to make it 0.91, it split into two branches, the

RDF branch and the simple branch. The RDF branch is advocated by RSS-DEV Group, where RSS means Rich Site Summary (RSS 1.0). The simple branch is advocated by famous blogger Dave Winer, where RSS means Really Simple Syndication (RSS 0.92, 0.93, 094, and 2.0) [9].

Daunted by the incompatibilities of RSS, a group of people started to re-invent a completely new and open feed format and get it through the IETF standardization process, which later becomes an RFC standard – the Atom Syndication Format [11]. It is accompanied by a draft on a REST-based protocol called Atom Publishing Protocol to further specify the message exchange mechanism between blog servers and clients.

No matter what format a feed use, a feed is composed of the same things conceptually: a header section describing the whole feed and a list of entries having similar attributes, such as unique identifier, title, description, published date and time. It is illustrated on the figure below:



```
<?xml version="1.0"?>
<rss version="2.0">
 <channel>
  <title>Liftoff News</title>
  <link>http://liftoff.msfc.nasa.gov/</link>
  <description>Liftoff to Space Exploration.</description>
  <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>

  <item>
   <title>Star City</title>
   <link>http://liftoff.msfc.nasa.gov/news/2003/news-starcity.asp</link>
   <description>How do Americans get ready to work with Russians
      aboard the
    International Space Station? They take a crash course in culture,
      language
    and protocol at Russia's Star City.</description>
   <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
   <guid>http://liftoff.msfc.nasa.gov/2003/06/03.html#item573</guid>
  </item>

  <item> ...</item>

  <item> ...</item>
 </channel>
</rss>
```

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

<title>Example Feed</title>
<subtitle>A subtitle.</subtitle>
<link href="http://example.org/"/>
<updated>2003-12-13T18:30:02Z</updated>
<author>
  <name>John Doe</name>
  <email>johndoe@example.com</email>
</author>
<id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>

<entry>
  <title>Atom-Powered Robots Run Amok</title>
  <link href="http://example.org/2003/12/13/atom03"/>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <summary>Some text.</summary>
</entry>

<entry>...</entry>

<entry>...</entry>

</feed>
```

Figure 2-1 File layout of a RSS (left) and an Atom (right) feed

2.1.3 Ping Servers

Many blog publishing systems send an XML-RPC request to one or more ping servers like those of Technorati and Google when a new post is submitted. The reason for this is to minimize the time between an actual update and those services' scheduled crawling of the updated content, thus providing users with fresh search results as soon as possible. Some of those ping servers like Weblogs.com and Google make the list of updated blogs available as an XML file with the name changes.xml usually for other services to leverage. Although there is no official specification of it, the format is usually as follows [12].

```
<weblogUpdates version="2" updated="Mon, 10 Oct 2005 14:10:00 GMT"
count="1384779">
   <weblog name="Weblogs.com" url="http://www.weblogs.com" when="1"/>
   <weblog name="My Blog site" url="http://www.myblogsite.com" when="2"/>
   <weblog name="Another site" url="http://www.anothersite.com" when="3"/>
</weblogUpdates>
```

2.2 Related Works

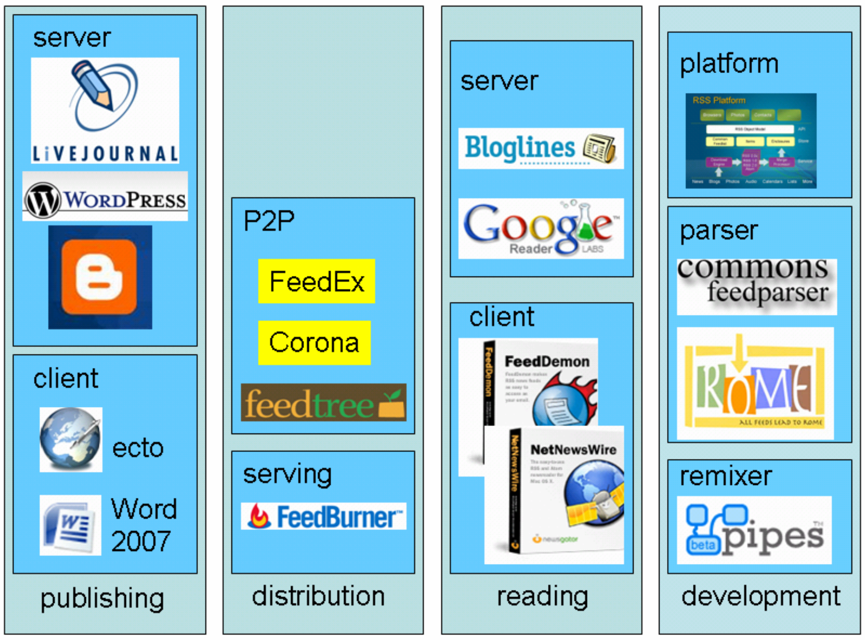


Figure 2-2 Overview of the feed space

The figure above depicts the feed space, divided by four columns which represent different tasks to do with feeds. Within it each column there are specific fields with their respectively players, some of whom will be further described in the following sections.

2.2.1 Windows RSS Platform

The Windows RSS Platform [13] is Microsoft's answer to the changing web experience from pure browsing to searching and subscribing after the release of Internet Explorer 6.0 in 2001. Though being an integral part of IE7, the Windows RSS Platform provides APIs for other applications in the same environment to access feeds and subscriptions, which is a similar idea to the one proposed by this paper, a platform instead of only a library. More will be discussed on the comparison and discussion section.

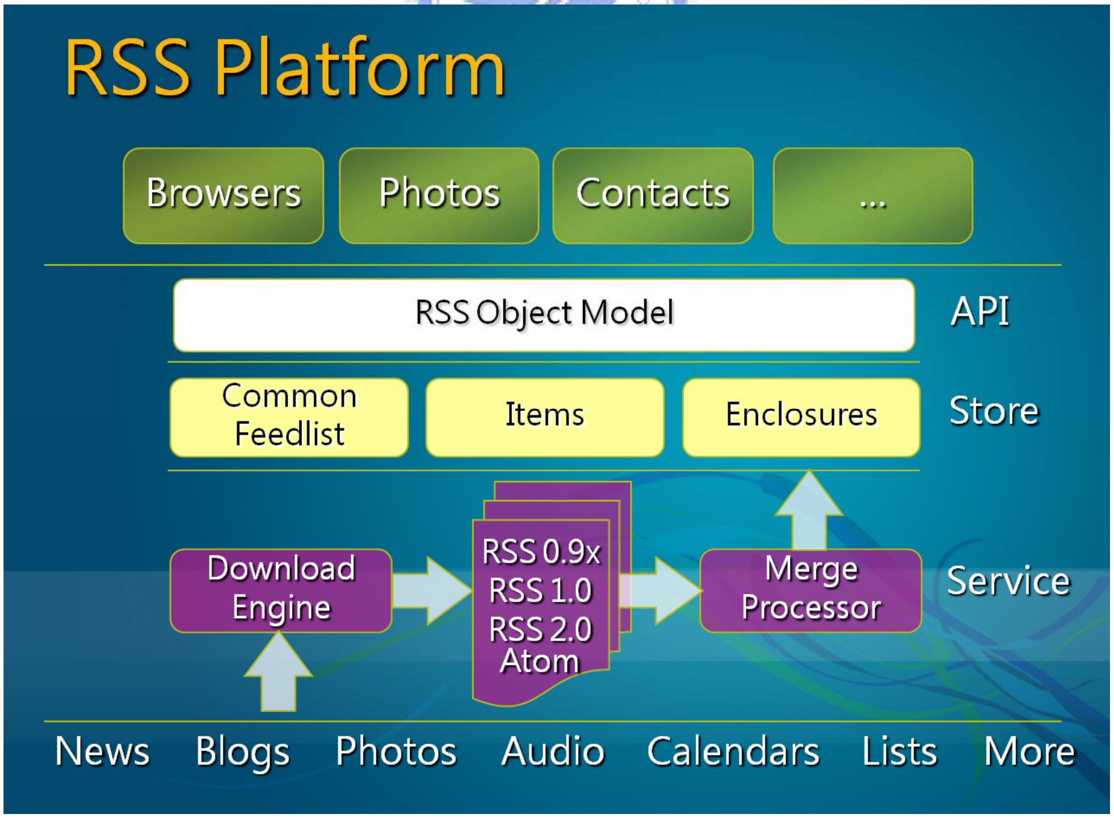
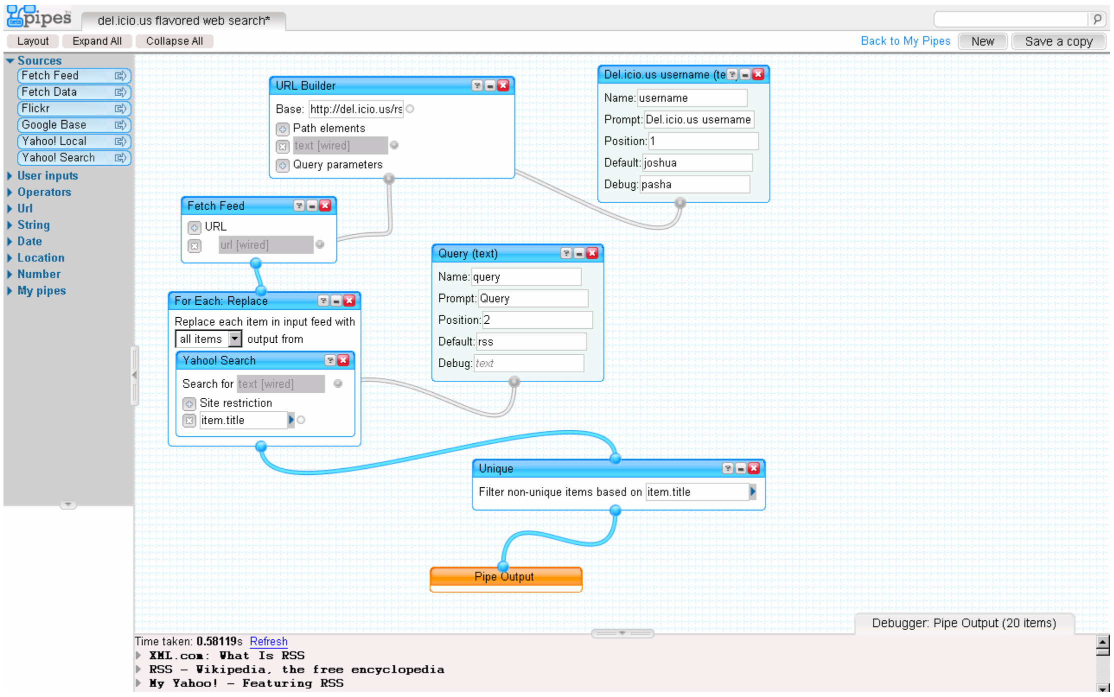


Figure 2-3 Architecture of the Windows RSS Platform [14]

2.2.2 Yahoo Pipes

Yahoo Pipes is a web application for non-programmer to aggregate and manipulate feeds [15]. It provides users with a GUI editor to connect inputs and outputs of different functional blocks, each having a specific use like URL building fetching feeds, or replacing text.



Base: http://del.icio.us/rs
text [wired]
url [wired]

Del.icio.us username (te
Name: username
Prompt: Del.icio.us username
Position: 1
Default: joshua
Debug: pasha

Query (text)
Name: query
Prompt: Query
Position: 2
Default: rss
Debug: text

Replace each item in input feed with all items output from
Search for text [wired]
item.title

Filter non-unique items based on item.title

```
Time taken: 0.58119s Refresh
XML.com: What Is RSS
RSS - Wikipedia, the free encyclopedia
My Yahoo! - Featuring RSS
```

Figure 2-4 Yahoo Pipes

2.2.3 Enterprise Solutions

Enterprises begin to adopt RSS to fight information overload with their portals and emails. Three commercial products focusing on helping enterprises to take advantages of the feed technology are Attensa Feed Server [16], NewsGator Enterprise Server [17], and KnowNow Enterprise Syndication Solution [18]. All of them share similar features: being a central server aggregating different feed sources on behalf of the organization, providing an easy-to-use interface for management of subscriptions, delivering news for reading using email clients, browsers or mobile devices. However, all of them have the same constraints of being only for feed consumption rather than development, and integration is hard if not totally impossible. Besides, they are all proprietary platforms and are selling at the price of over thousands of US dollars.

2.2.4 Academic Researches

Three researches are directly related to the feed technology. FeedEx [19] is a feed exchanging system, in which hosts not only fetch feeds but also exchange them with neighbors of similar interests to reduce time lag and increase coverage. Based on Scribe and Pastry, FeedTree [20] provides software for subscribers and publishers to join a structured overlay to let them distribute feeds in a multicast way and poll for updates cooperatively. Also based on Pastry, Corona [21] does almost same thing as FeedTree but focus more on load balancing of nodes in the overlay to achieve better performance. In short, all of them are P2P-related researches which focus on the scalability of feed dissemination.

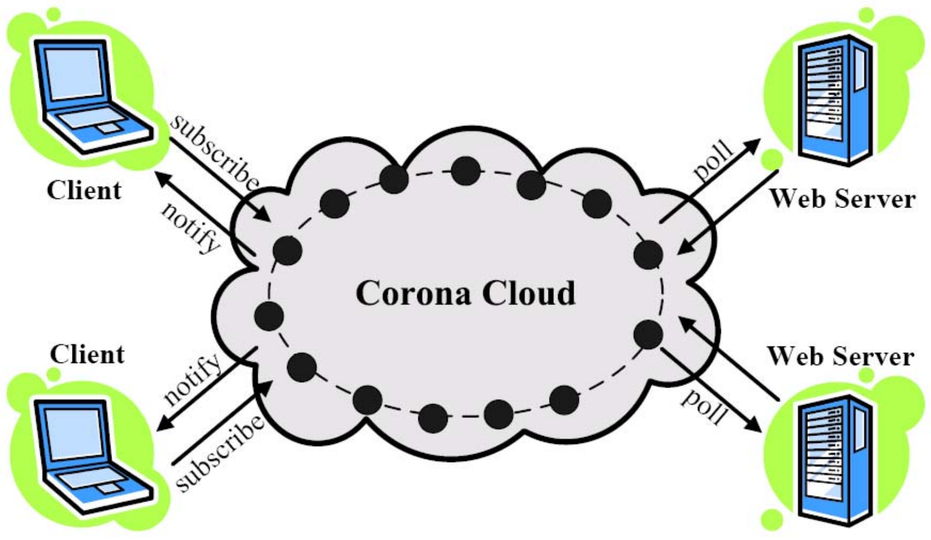


Figure 2-5 Corona Architecture

3 System Architecture

3.1 Overview

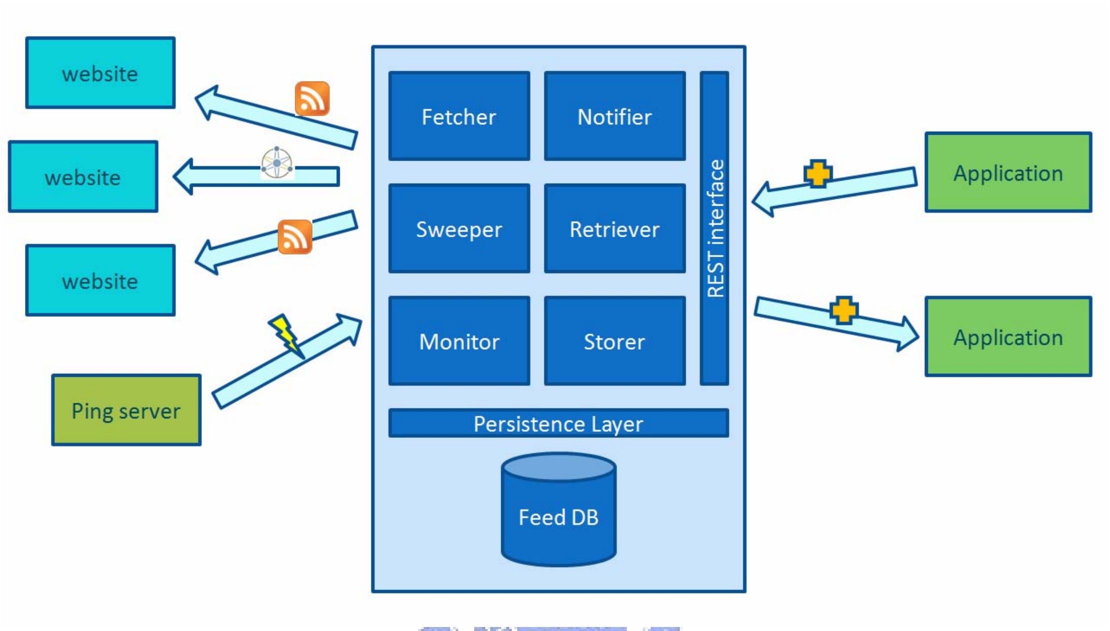


Figure 3-1 System Architecture

The diagram above depicts the components of Feed Middleware, which will be described in details in the following sections.

3.2 Feed DB, Storer and Retriever

Feed DB is a database to store all entries of all feeds and other relevant information. Regardless of what format a feed is in, entries of all subscribed feeds are stored in two different ways. First they are stored in a normalized form which only captures the essence of an entry including its unique identifier, title, link, description and timestamp. Second, they are stored in a serialized form which preserves all of its attributes. The rationale behind these redundant stores is that both performance and flexibility are desired, and that storage is inexpensive and it can be easily expanded.

A list of subscribed feeds with their attributes including their last updated time, fetch frequencies are also stored. The following figure is the database schema.

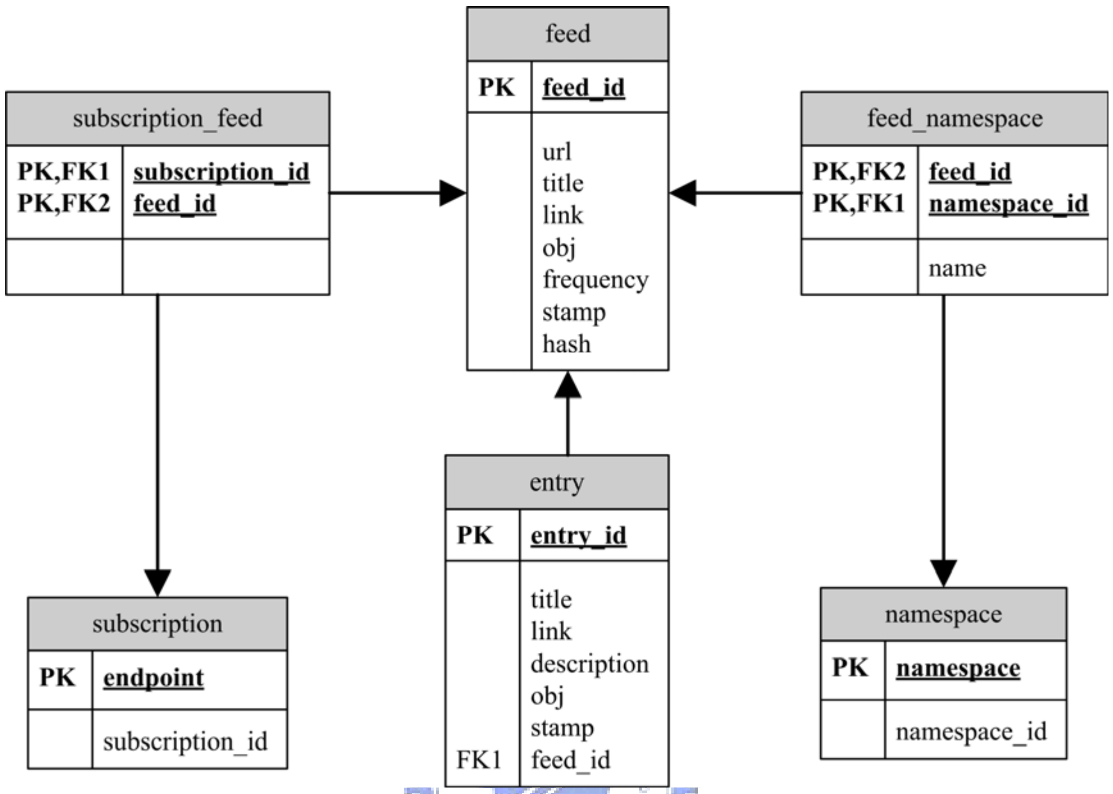


Figure 3-2 Database schema

Feed Storer has the knowledge of both the object representation and the database schema. First, it filters out old entries having the same ids. Then, it transforms only the updated ones into tuples suitable to be inserted into the Feed DB.

Feed Retriever is responsible for retrieving feed entries from database and formatted them in the form requested by client applications. Frequent retrievals are alleviated by using a memory caching system so as to provide fast response. Since all entries are stored in the database, merging different feeds into a single one can be done but using a SQL SELECT statement with an IN expression constraint test for inclusion in a specified set of feed IDs. This mechanism also enables the use of OPML (Outline Processor Markup Language) which is often used as an XML format of subscription lists.

3.3 Feed Sweeper, Monitor and Fetcher

Feed Sweeper is a scheduled process to constantly examine the status of every feed, marking it dirty if the current time is later than its last updated time plus its fetch frequency. Dirty feeds are then put into a queue for Feed Fetcher to re-fetch.

Instead of guessing if there is update for a blog, feed Monitor leverage the knowledge of ping servers by downloading change logs from them, scan through them for interested feeds that are updated, and put them into queue for Feed Fetcher to re-fetch.

Feed Fetcher is responsible for fetching feeds, parsing them into objects, and storing them into Feed DB using Feed Storer. It contains a pool of worker threads to do these processes concurrently in order to achieve a higher throughput. Feed Fetcher begins to fetch a feed when notified by Feed Sweeper of a feed being marked as dirty or by Feed Monitor of feeds being updated. It uses various HTTP techniques which will be mentioned in the implementation section later to reduce bandwidth usage. A hash code is also kept for each feed to compare content freshness besides the HTTP ETag header to ensure further processing is needed only for updated feeds. A feed parser is used to parse different XML-based feed formats into a consistent object model.

Instead of using mathematical or heuristic methods to dictate the fetch frequencies of feeds, which is complicated and not in the scope of this text, aids are provided to the users to determine the frequency of the feeds of interest. Fetch frequency can be divides into different levels. Level 0 is set for those feeds which updates have been sent to ping servers, and in turn realized by Feed Monitors. The fetch frequency is 1 day for level 0. Level 1 is the default one for every feeds, which is 30 minutes. This level is suitable for blogs or non-frequently updating sites. People normally do not mind if they are 30 minutes late to know some trifles of their friends. When an update is received from ping servers for level 1 feeds, it is set to level 0 because it can be assumed that subsequent updates will also be received from ping servers so there is no need to fetch that often. Conversely, if daily fetch for a level 0 feed finds missed updates, the fetch frequency of the respective feed is set to level 1 because ping server may not be reliable for that feed any more. Level 2 is 5 minutes for news or real-time updating sites. Finally, users can always set the exact fetch frequency directly to values other than these three levels.

3.4 Feed Notifier

Feed Notifier is initiated by Feed Fetcher with only updated feeds entries. It checks the subscription tables in the database to see if there is anyone who is interested in those updates. It one is found, a separated thread is dispatched to push those entries to the respectively endpoint using XML-RPC. XML-RPC is a simple way to communicate with a remote entity. It is possible to use more reliable mechanisms like message-oriented middleware directly or through adapters.

3.5 Interfaces

Applications access feeds by sending simple HTTP requests to the Feed Middleware similar to retrieving feeds from any web servers. But Feed Middleware allows developers to specify how feeds should be served using arguments. The following tables list all operations provided by Feed Middleware with their function descriptions:

Resource	/feed
HTTP Method	GET

Description	retrieve a single feed of entries	
Arguments	url	a single URL
	type	rss, atom, json
	len	how many entries to retrieve
Example	GET /feed/?url=http://digg.com/rss/index.xml&type=atom&len=50	

Table 3-1 Retrieve a single feed of entries

Resource	/feeds	
HTTP Method	GET	
Description	retrieve a set of feeds of entries	
Arguments	url	comma-separated list of URLs
	type	rss, atom, json
	len	how many entries to retrieve
Example	GET /feeds/?url=http://digg.com/rss/index.xml, http://rss.slashdot.org/Slashdot/slashdot&type=json&len=100	

Table 3-2 Retrieve a set of feeds of entries

Resource	/opml	
HTTP Method	GET	
Description	retrieve a set of feeds of entries defined by an OPML file	
Arguments	url	an URL of an OPML file
	type	rss, atom, json
	len	how many entries to retrieve
Example	GET /opml/?url=http://share.opml.org/opml/top100.opml&type=rss	

Table 3-3 Retrieve a set of feeds of entries defined by an OPML file

Resource	/sub

HTTP Method	GET	
Description	retrieve all subscribed feeds as an OPML file	
Example	GET /sub	

Table 3-4 Retrieve all subscribed feeds as an OPML file

Resource	/sub	
HTTP Method	POST	
Description	subscribe to a lists of feeds	
Arguments	url	comma-separated list of URLs
Example	POST /sub/?url= http://digg.com/rss/index.xml, http://rss.slashdot.org/Slashdot/slashdot	

Table 3-5 Subscribe to a lists of feeds

Resource	/sub	
HTTP Method	DELETE	
Description	unsubscribe a lists of feeds	
Arguments	url	comma-separated list of URLs
	id	subscription_id
Example	DELETE /sub/?url= http://digg.com/rss/index.xml, http://rss.slashdot.org/Slashdot/slashdot&id=1	

Table 3-6 Unsubscribe a lists of feeds

3.6 Program Flow

Assume that there are already some feeds in the database, all with different fetch frequencies. A work queue is maintained for Sweeper and Monitor to communicate with Fetcher in the producer-consumer paradigm. Feed Sweeper is scheduled to put outdated feeds into the queue. By outdated it only means that the feed has not been fetched for some specific period time. It does not necessary mean that it is updated.

On the contrary, Feed Monitor leverages update logs by ping servers to put actually updated feeds into the queue. Upon receiving fetch requests from the queue, Feed Fetcher fetches those feeds, parses them into objects, filters out old entries, stores new ones into the Feed DB using Feed Storer, and dispatches notification threads using Feed Notifier.

From the point of the view of the developers, they only have to send HTTP requests in order to subscribe, unsubscribe to feeds, or retrieving them directly in a couples of different ways. If they subscribe to a feed, updates will be pushed to them.

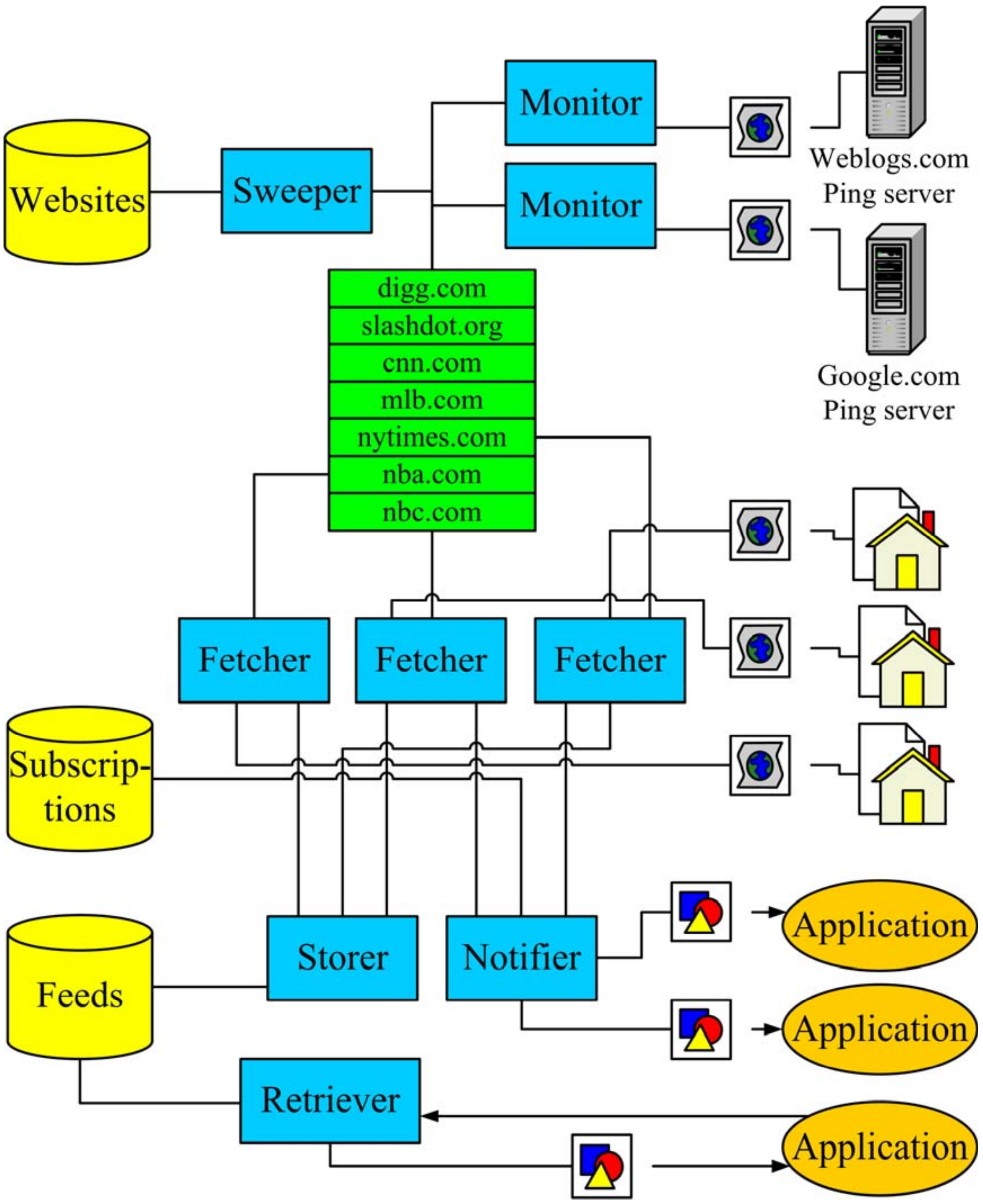


Figure 3-3 Program flow of Feed Middleware

4 Implementation Details

4.1 Overview

As mentioned in these two articles on middleware “dark matter” [23] [24], Python is one of many tools to solve real world integration problems when EAI, MOM, Corba, and J2EE are just too complex and over killed. Python [25] is a dynamic object-oriented programming language that can be used for many kinds of software development. It is well known that Google used Python intensively for many of its systems. There are also extensive standard libraries and many 3rd party tools like the brilliant Universal Feed Parser. Besides, Python is available for Windows, Macintosh, Linux and a lot other platforms. Due to these reasons, Feed Middleware is developed entirely in Python.

4.2 Feed discovery

Feed discovery is to get the feed URL of a website given its own URL. This feature can be handy when retrieving a feed for the first time because users will not need to know the feed URL in advance. Instead of having default names like index.html or index.php for the entrance of a websites, there is no similar convention for feeds. However, webmasters use a way similar to referencing external stylesheets and scripts to associate a feed with a website by adding a link tag within the head section of a webpage. Therefore, the following steps can be used to get the feed URL:

- retrieve the HTML file of a website
- use regular expression to find all link tags

- for each link tag if its type is “application/rss+xml”, get its href attribute
- use the href attribute and the original URL to form a feed URL

4.3 Feed fetching

A feed is just yet another object transferred over HTTP like a HTML document. Techniques used by browser and other HTTP clients can be directly employed to speed up fetching and reduce bandwidth usage. One of them is caching with validation. For example, a client issues a request for a feed and the server responds with the XML document in the payload and optional ETag (entity tag) and/or Last-Modified headers. The client may cache the document so when it wants to request for the same feed next time, it can attaches If-None-Match and/or If-Modified-Since headers with previous values to check if its cached version is still valid. If it is, a status code 304 Not Modified is returned without payload; otherwise, a normal response is returned. ETag is a strong content hash validator which will change accordingly with the content itself. Last-Modified is a weak validator derived implicitly from the last modified time of the content. They both serve as good mechanisms to reduce unnecessary requests.

4.4 Feed parsing

There are a number of feed parsers available to tackle the problem of the chaotic feed formats. The Windows RSS Platform can be used for .Net environments. For Java, Rome [26] is probably the most promising one with a strong community and sub-projects to handle other issues such as fetch and store. Jakarta FeedParser [27] is an alternative for Java with a SAX instead of DOM-based API. For Python, Universal Feed Parser [28] is the one to use. UFP is chosen not only because the middleware is written in Python, but also because UFP is the most liberal parser of all. Being liberal is very important because feed publishers, being spoiled by browsers accepting all kinds of HTML documents, tend to produce ill-formatted feeds. Besides, they may also mix up entities of different formats which will fail parsers that are completely conforming to the specifications.

Universal Feed Parser tries to expose all values of non-standard extensions as possible. For example, each entry of the feed of the famous Web 2.0 news site Digg [29] comes with a digg count (*<digg:diggCount>42</digg:diggCount>*), that is, the number of votes it gets from the users of Digg. The value can simply be accessed by directly *d.entries[i].digg_diggcount*. However, as of the latest release of UFP, attribute values are not preserved. Therefore, a modification must be made for extension like the Buy.com RSS 2.0 Product Module Definition [30], which product information is formatted as attributes (*<product:content price="$2,021.99"/>*), so that values are stored in a dictionary that be accessed by a key composed of the tag name and an under scroll (*d.entries[i].product_content_['price']*).

4.5 Interfaces

The pull interface follows the REST style and is implemented using the web.py framework. REST (Representational State Transfer) [31] is an architecture style to design network-based software. The principle of REST is to model application states and functionalities as resources which can be addressed using a universal syntax and be interacted with by exchanging representations via a simple and uniform interface (often HTTP). web.py [32] is a framework written in Python for developing web applications with REST in mind. It uses Python classes and member functions to define resources and their respectively HTTP method interfaces. A Python tuple is used to map URLs to resources.

Feeds output formats can be RSS 2.0 or Atom 1.0 regardless of them originally formats. Other kinds of formats can be easily supported by simply defining respective templates. For object output, JSON (Javascript Object Notation) [33] is used to facilitate object exchange across different languages.

XML-RPC is used for the push interface for its simplicity and many implementations for different languages. XML-RPC [34] is simple a way to encode and decode method calls, arguments and return values in XML and transfer them over HTTP. A client registers its XML-RPC endpoint, interested feeds and attributes with Feed Middleware. When updates are available for those feeds, only updated entries are sent to the client by an XML-RPC method call with those entries being XML-formatted object arguments.

These two interfaces are put into use by two demo applications written in different languages as show in the demo chapter.

4.6 Tools and Libraries

Many open source tools and libraries are used in Feed Middleware. They are listed below:

Name	Usage	License
Python	Core	Python license
Universal Feed Parser	Parsing feeds	MIT
SQLite	Embedded database	Public domain
memcahed	Memory cache	BSD
web.py	Web framework	Public domain

Table 4-1 Tools and libraries used

5 Scenario Demonstrations

5.1 Ajax Product Spy

Ajax (Asynchronous Javascript and XML) is a term defined by Jesse James Garrett [35] referring to the combination of techniques, including the Javascript XMLHttpRequest (XHR) object, DOM manipulation and XHTML, involved in the development of interactive web applications. The core of Ajax is XHR [36], which enables Javascript embedded in a webpage to issue asynchronous HTTP requests without the needs to refresh the whole page, thus resulting in a fluid user experience.

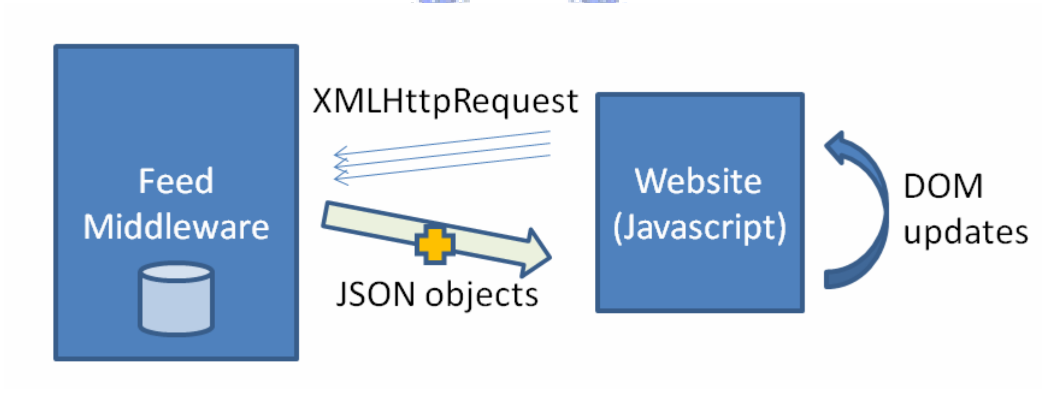
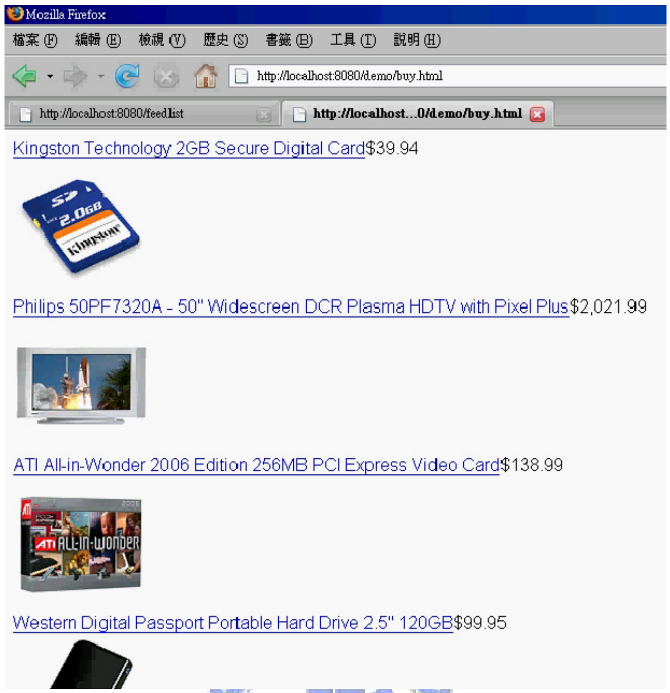


Figure 5-1 Architecture of the Ajax Product Spy

This demo is a single webpage which contains product information updating automatically without refreshes. Such an application can be integrated into an existing internal portal of an enterprise to show the inventory of itself or its competitors.

The webpage contains an Ajax Request object from the Prototype Javascript Framework [37] to make repeated requests to Feed Middleware asking for entries of the interested products serialized in JSON. The callback function of the request inserts new products to the webpage by updating the DOM tree. Also demonstrated is the flexibility of Feed Middleware directly returning elements like product prices and images of the Buy.com RSS 2.0 Product Module Definition [29]without any modifications.


Mozilla Firefox
檔案 (F) 編輯 (E) 檢視 (V) 歷史 (S) 書籤 (B) 工具 (T) 說明 (H)
http://localhost:8080/demo/buy.html
http://localhost:8080/feedlist
http://localhost...0/demo/buy.html

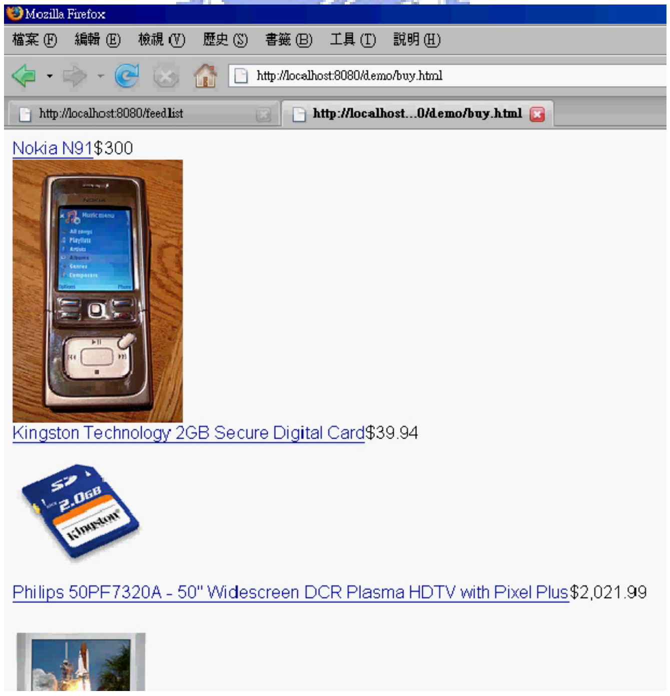
Kingston Technology 2GB Secure Digital Card$39.94

Philips 50PF7320A - 50" Widescreen DCR Plasma HDTV with Pixel Plus$2,021.99

ATI All-in-Wonder 2006 Edition 256MB PCI Express Video Card$138.99

Western Digital Passport Portable Hard Drive 2.5" 120GB$99.95

Figure 5-2 Screenshot of the Ajax Product Spy showing product information


Mozilla Firefox
檔案 (F) 編輯 (E) 檢視 (V) 歷史 (S) 書籤 (B) 工具 (T) 說明 (H)
http://localhost:8080/demo/buy.html
http://localhost:8080/feedlist
http://localhost...0/demo/buy.html

Nokia N91$300

Kingston Technology 2GB Secure Digital Card$39.94

Philips 50PF7320A - 50" Widescreen DCR Plasma HDTV with Pixel Plus$2,021.99

Figure 5-3 Screenshot of the Ajax Product Spy after an update

```
<script>

function fetch() {
new Ajax.Request('/feed?url=http://localhost:8080/’+
’demo/buy.xml&type=json&len=50&obj', {
    method: 'get',
    onSuccess: function(t) {
          entries = eval('('+t.responseText+')')
          for (var i=0; i<entries.length; ++i) {
              if (!$(entries[i].link)) {
                  new Insertion.Top('container',
                    '<div id="' + entries[i].link + '">' +
                    '<a href="' + entries[i].link + '">' +
                    entries[i].title + '</a>' +
                    entries[i].product_content_['price'] + '<br/>' +
                    '<img src="' + entries[i].product_content_['imageurl'] +
                    '"/>' + '</div>');
              }
          }
    }
});
setTimeout(fetch, 5000)
</script>
<script>fetch();</script>
```

5.2 Bug Notifier

As mentioned above, polling is used as the underlying technique to enable publish/subscribe semantics of feeds. In order to receive timely notification for urgent stuff like system outage reports, an event notifier is written to send feed updates to users via instant messages, emails or SMS messages.

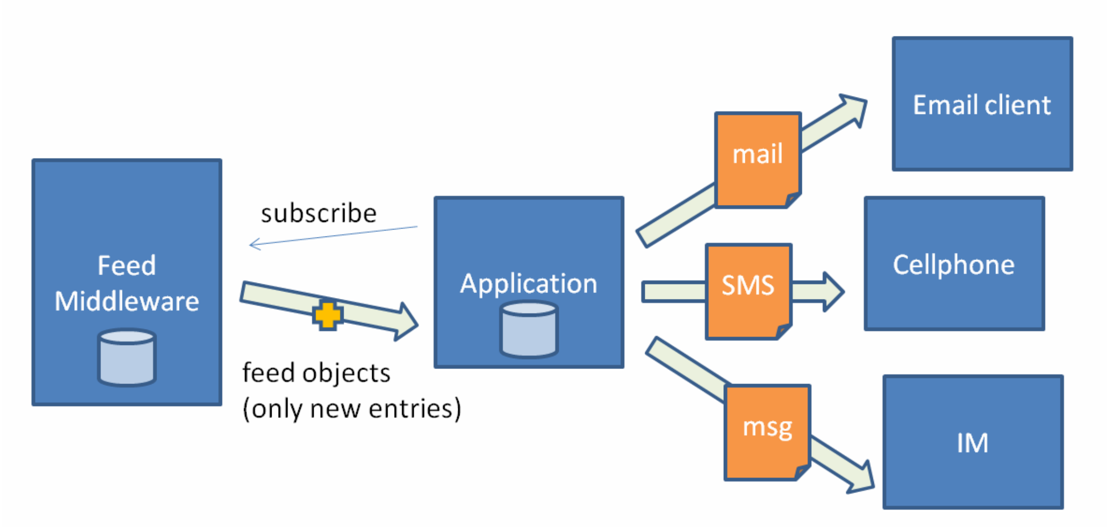
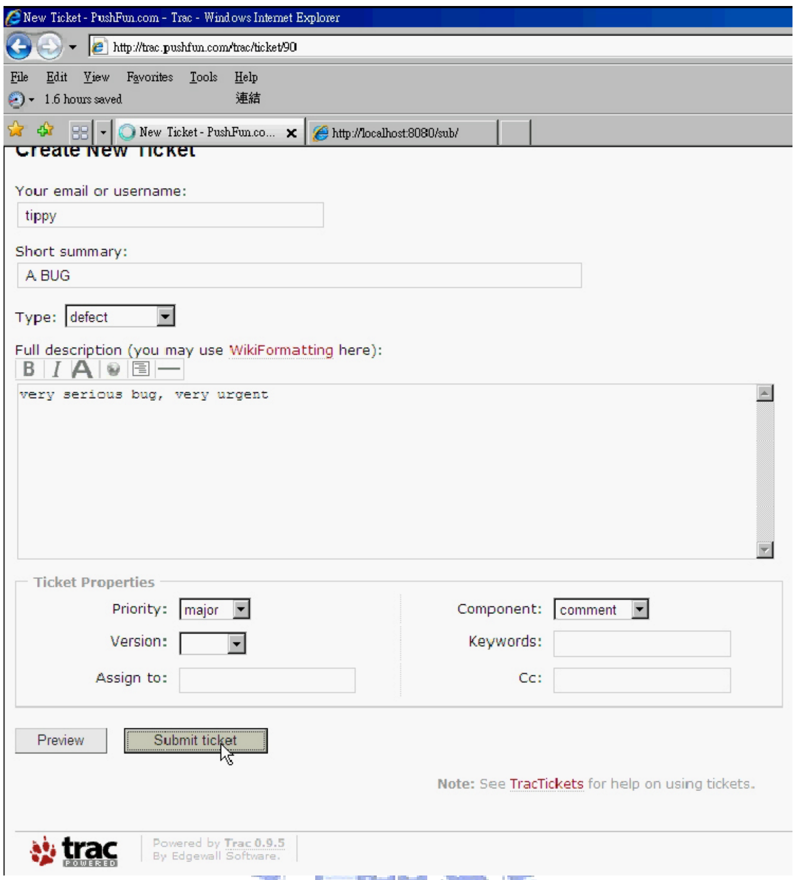


Figure 5-4 Architecture of the Bug Notifier

The Bug Notifier is written in Java to demonstrate that applications using Feed Middleware can be language neutral. It contains a simple XML-RPC server offered by Apache XML-RPC implementation [38] for Java to listen for new entries of subscribed feeds. Besides, it also leverages JMSN [39], a Java Microsoft MSN Messenger clone, to communicate with the MSN network and send feed entries as instant messages to users.

The following scenario shows that when a tester of a system reports a bug, the developer of that system, having subscribed to the bug feed, will be notified by a MSN robot with relevant information to lead him to that bug report page. Moreover, since the bug reporting system and its corresponding feed are password protected, this program also shows the capability to handle security feeds which is not supported by Windows RSS Platform.



New Ticket - PushFun.com - Trac - Windows Internet Explorer

http://trac.pushfun.com/trac/ticket/90

Create New Ticket

Your email or username: tippy

Short summary: A BUG

Type: defect

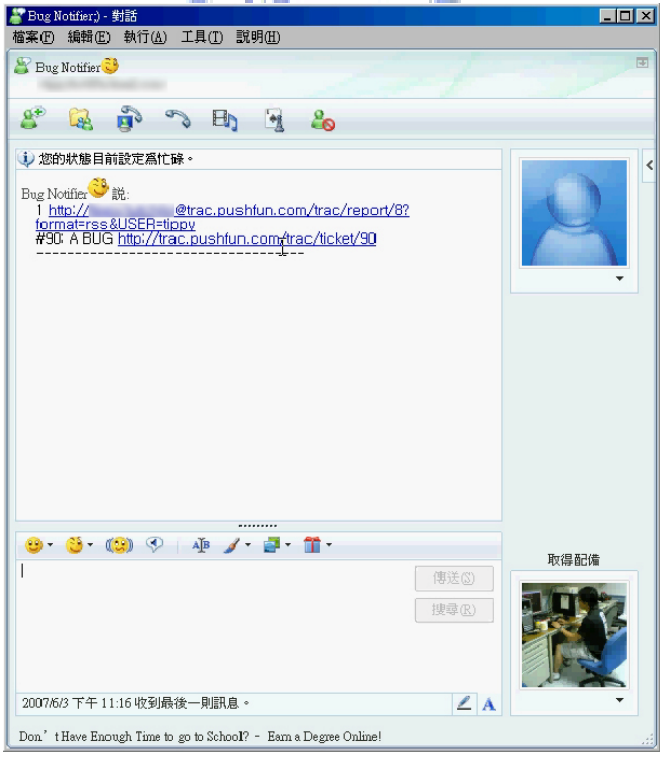
Full description (you may use WikiFormatting here):

```
very serious bug, very urgent
```

Ticket Properties

Priority:	major	Component:	comment
Version:		Keywords:	
Assign to:		Cc:	

Figure 5-5 Screenshot of a tester filling the form to file a bug report



Bug Notifier;) - 對話

Bug Notifier

[redacted]

您的狀態目前設定為忙碌。

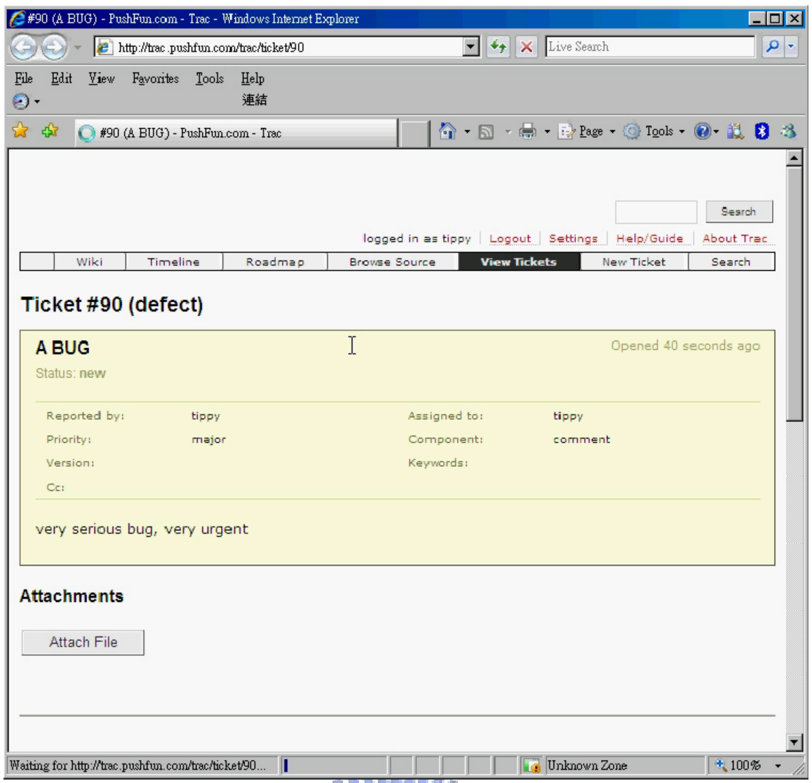
Bug Notifier 說:

```
1 http://[redacted]@trac.pushfun.com/trac/report/8?format=rss&USER=tippy
#90: A BUG http://trac.pushfun.com/trac/ticket/90
------------------------------------
```

2007/6/3 下午 11:16 收到最後一則訊息。

Don't Have Enough Time to go to School? - Earn a Degree Online!

Figure 5-6 Screenshot of the developer is notified for a bug report via the MSN Messenger


#90 (A BUG) - PushFun.com - Trac - Windows Internet Explorer
http://trac.pushfun.com/trac/ticket/90
Live Search
File Edit View Favorites Tools Help
連結
#90 (A BUG) - PushFun.com - Trac
Page Tools
Search
logged in as tippy | Logout | Settings | Help/Guide | About Trac
Wiki Timeline Roadmap Browse Source View Tickets New Ticket Search
Attach File
Waiting for http://trac.pushfun.com/trac/ticket/90...
Unknown Zone
100%

Ticket #90 (defect)

A BUG

Opened 40 seconds ago

Status: new

Reported by:	tippy	Assigned to:	tippy
Priority:	major	Component:	comment
Version:		Keywords:	
Cc:			

very serious bug, very urgent

Attachments

Figure 5-7 Screenshot of the developer clicking the link from the instant message to see the actual bug report

```
public int feedReceived(int subscriptionId, String url, Vector entries) {

    StringBuffer buffer = new StringBuffer();

    buffer.append(subscriptionId+" "+url+"\r\n");

    for (Iterator i=v.iterator(); i.hasNext();) {

        Hashtable h = (Hashtable)i.next();

        buffer.append(h.get("title")+"\r\n");

    }

    try {

        SwitchboardSession ss = null;

        if (!switchMap.containsKey(recipient)) msn.doCallWait(recipient);

        if (switchMap.containsKey(recipient)) {

        ss = (SwitchboardSession)switchMap.get(recipient);

        ss.sendInstantMessage(new MimeMessage(buffer.toString()));
    } catch (IOException e) {}

    return 1;

}
```

6 Comparison and Discussion

6.1 Comparison

6.1.1 Qualitative

Both Feed Middleware and the Windows RSS Platform aimed at providing developers with a tool to work with feeds, which is more than a feed parser, but there are some significant differences between them. First, the Windows RSS Platform is for personal use only, whereas Feed Middleware can be used both personally and by an organization because it has a web service interface. An enterprise may use Feed Middleware as a hub to internal and external information. Second, the Windows RSS Platform is tightly-coupled with Microsoft's proprietary technologies like IE and .Net, which is not available to use by open source developers for Linux or Java. Even for .Net developers, there is no way for them to extend its functionalities to keep up with the changing feed space. Third, the Windows RSS Platform is more difficult to use with a more complex API and there no support for fundamental things like feed auto-discovery, password-protected or ill-formatted feeds.

However, the Windows RSS Platform has a good feature which is to download enclosures, embedded reference to multimedia resources like MP3 files, automatically in the background and replace the foreign URLs with local file system ones.

The following table summarizes the difference between Feed Middleware and Windows RSS Platform.

	Feed Middleware	Windows RSS Platform

Style	Middleware	API
Scope	Personal, Organizational	Personal
Language	Neutral	.Net Languages only
Source	Open	Closed
Feed parsing	Liberal	Strict
Feed discovery	Yes	No
Extensions	Supports a lot of standards and flexible with non-standards	Parse XML by oneself, invent two new extensions
Security feeds	Yes	No
Enclosure downloading	No	Yes

Table 6-1 Qualitative comparison between Feed Middleware and Windows RSS Platform

6.1.2 Quantitative

In this section, two groups of code fragments are listed. One of them uses Windows RSS Platform. The other one uses Feed Middleware. The first group of code fragment is simply printing the titles for a specific feed. The second group subscribes to a feed, and print titles of it when notified for updates.

Procedural feed printing using Windows RSS Platform:

```
string url = "http://www.digg.com/rss/index.xml";

FeedsManager fm = new FeedsManager();

IFeedFolder rootFolder = (IFeedFolder)fm.RootFolder;

IFeed feed = (IFeed)rootFolder.CreateFeed(url, url);

foreach (IFeedItem item in (IFeedsEnum)feed.Items)

Console.Out.WriteLine(item.Title);
```

Procedural feed printing using Feed Middleware:

```
d = urllib.urlopen('http://localhost:8080/feed/\

?url=http://digg.com/&type=json').read()

o = simplejson.loads(d)

for e in o['entries'] : print e['title']
```

Event-driven feed printing using Windows RSS Platform:

```
FeedsManager fm = new FeedsManager();

IFeedFolder rootFolder = (IFeedFolder)fm.RootFolder;

FeedFolderEvents_Event fw = (IFeedFolderEvents_Event)rootFolder.GetWatcher(

FEEDS_EVENTS_SCOPE.FES_ALL, FEEDS_EVENTS_MASK.FEM_FEEDEVENTS);

fw.FeedItemCountChanged += new

IFeedFolderEvents_FeedItemCountChangedEventHandler(FeedItemCountChanged);

void FeedItemCountChanged(String path, int itemCountType) {

   IFeed feed = (IFeed) fm.GetFeed(path);

   if (feed.url != "http://www.digg.com/rss/index.xml") return;

   foreach (IFeedItem item in (IFeedsEnum)feed.Items)

Console.Out.WriteLine(item.Title);

}

Console.In.ReadLine();
```

Event-driven feed printing using Feed Middleware:

```
postdata = urllib.urlencode({'endpoint':'http://localhost:8000/',\

'url':'http://digg.com/'})

urllib.urlopen('http://localhost:8080/sub/', postdata)

def feedReceived(self, subscription_id, url, entries):

   for e in entries : print e['title']

server = SimpleXMLRPCServer(("localhost", 8000))
```

```
server.register_function(feedReceived)

server.serve_forever()
```

The number of lines of codes for the first group is 6 to 3 and the second group is 10 to 7. Although the Windows RSS Platform versions of both groups of code fragments have been reduced to its essence, which is not runnable codes compared to those using Feed Middleware, they already exhibit the complexity of the API. It is almost impossible to make use of it with before consulting the reference. On the contrary, the Feed Middleware version only comprises of standard library usage like the urllib and the SimpleXMLRPCServer modules, with the only exception being the simplejson library.

6.2 Discussion

According to the analysis above, Feed Middleware has achieved the objectives stated in 1.3. It is easy-to-use, flexible, liberal and open. It can be shown that by designing with simplicity in mind and using existing open source tools, a system can be built to meet the needs of certain people who want to tackle the problem at hand and do a rapid integration without having to set a lot up things up and learn a complex API with excessive and obscure functions.

7 Future Works and Conclusion

7.1 Future Works

Since Feed Middleware is a proof of concept prototype, there are several things that could be done to make it a better tool. First, Feed Middleware uses SQLite database and the embedded web server of web.py, which may not scale very well for production environments. Fortunately, it can be easily swapped with industry-strength counterparts like MySQL and Apache. Second, automatically enclosures downloading provided by the Windows RSS Platform should be implemented because it can further reduce bandwidth in scenarios such as syndication of e-learning video clips across different schools while each of them has a Feed Middleware to download those clips on behalf of the whole school for their students. Third, with the growing use of RSS and Atom feeds, more extensions will be developed to embed more semantics to them, parsing them one by one with plug-ins will be too tedious. Sooner or later, a specification of a format lying between RSS/Atom and RDF, which it is more general than the former and less general than the latter, will be needed.

7.2 Conclusion

In this paper, a tool for developing applications to take advantages of feed technologies like RSS and Atom is proposed. After reviewing the backgrounds and related works in the this space and what is lacking, Feed Middleware is proposed to treat feeds as streams of information instead of discrete XML documents that only a parser cannot solve the whole problem. Besides, learning from the simplicity of RSS, Feed Middleware has easy-to-use interfaces built with existing open source tools. To demonstrate its capabilities, two simple but practical applications are written in different languages and different styles. Finally, Feed Middleware is compared with Windows RSS Platform to show its advantages and future works that should be done to further improve it.

References

[1] Tim O'Reilly, What is Web 2.0, Sept 2005
http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[2] David Sifry, The State of the Live Web, April 2007
http://www.sifry.com/alerts/archives/000493.html

[3] RSS Extensions
http://rss-extensions.org/

[4] Microformats
http://microformats.org/

[5] Danny Ayers, From Here to There, IEEE Internet Computing, Volume 11, Issue 1, Jan-Feb 2007

[6] Dave Johnson, RSS and Atom In Action, Aug 2006

[7] Randy Charles Morin, HowTo RSS Feed State
http://www.kbcafe.com/rss/rssfeedstate.html

[8] Blog, Wikipedia
http://en.wikipedia.org/wiki/Blog

[9] Mark Pilgrim, The Myth of RSS compatibility, Feb 2004

[10] http://diveintomark.org/archives/2004/02/04/incompatible-rss

[11] Mark Nottingham and Robert Sayre, The Atom Syndication Format, IETF RFC 4287, Dec 2005

[12] Weblogs.com API
http://weblogs.com/api.html

[13] Windows RSS Platform
http://msdn2.microsoft.com/en-us/library/ms684701.aspx

[14] Amar Gandhi, RSS in Windows Vista, Microsoft Professional Developers Conference (PDC), 2005

[15] Yahoo Pipes
http://pipes.yahoo.com/pipes/

[16] Attensa Feed Server
http://www.attensa.com/products/server/

[17] NewsGator Enterprise Server
http://www.newsgator.com/Business/EnterpriseServer/

[18] KnowNow Enterprise Syndication Solution
http://www.knownow.com/article/?id=140

[19] Seung Jun and Mustaque Ahamad, FeedEx: collaborative exchange of news feeds, Proceedings of the 15th International Conference on World Wide Web (ACM WWW’06), 2006

[20] Dan Sandler, Alan Mislove, Ansley Post and Peter Druschel, FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification, Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS'05), 2005

[21] Venugopalan Ramasubramanian, Ryan Peterson and Emin Gun Sirer, Corona: A High Performance Publish-Subscribe System for the World Wide Web, Proceedings of Networked System Design and Implementation (NSDI’06), May 2006

[22] OPML (Outline Processor Markup Language)
http://www.opml.org/

[23] Steve Vinoski, Middleware “Dark Matter”, IEEE Internet Computing, Volume 6, Issue 5, Sept-Oct 2002

[24] Steve Vinoski, Dark Matter Revisited, IEEE Internet Computing, Volume 8, Issue 4, July-Aug 2004

[25] Python
http://python.org/

[26] Rome
https://rome.dev.java.net/

[27] Jakarta FeedParser
http://jakarta.apache.org/commons/sandbox/feedparser/

[28] Universal Feed Parser
http://www.feedparser.org/

[29] Digg.com
http://digg.com/

[30] Buy.com RSS 2.0 Product Module Definition Version 1.0
http://www.buy.com/rss/module/product/

[31] Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures, PhD dissertation, UC Irvine, 2000

[32] web.py
http://webpy.org/

[33] JSON (Javascript Object Notation)
http://www.json.org/

[34] XML-RPC
http://www.xmlrpc.com/

[35] Jesse James Garrett, Ajax: A New Approach to Web Applications
http://www.adaptivepath.com/publications/essays/archives/000385.php

[36] XMLHttpRequest
http://www.w3.org/TR/XMLHttpRequest/

[37] Prototype Javascript Framework

http://www.prototypejs.org/

[38] Apache XML-RPC

http://ws.apache.org/xmlrpc/xmlrpc2/

[39] JMSN

http://sourceforge.net/projects/jmsn/