DOI:10.1478/C1A0802011

Atti dell'Accademia Peloritana dei Pericolanti
Classe di Scienze Fisiche, Matematiche e Naturali
Vol. LXXXVI, C1A0802011 (2008)
Adunanza del 29 novembre 2007

ADAPTIVE WEB DATA EXTRACTION POLICIES

GIACOMO FIUMARA,* MASSIMO MARCHI, AND ALESSANDRO PROVETTI

(Nota presentata dal Socio Ordinario Paolo V. Giaquinta)

ABSTRACT. Web data extraction is concerned, among other things, with routine data accessing and downloading from continuously-updated dynamic Web pages. There is a relevant trade-off between the rate at which the external Web sites are accessed and the computational burden on the accessing client. We address the problem by proposing a predictive model, typical of the Operating Systems literature, of the rate-of-update of each Web source. The presented model has been implemented into a new version of the Dynamo project: a middleware that assists in generating informative RSS feeds out of traditional HTML Web sites. To be effective, i.e., make RSS feeds be timely and informative and to be scalable, Dynamo needs a careful tuning and customization of its polling policies, which are described in detail.

1. Introduction

Web data extraction is concerned, among other things, with routine data accessing and downloading from continuously-updated dynamic Web pages. There is a relevant trade-off between the rate at which the external Web sites are accessed and the computational burden on the accessing client. We address the problem by proposing a predictive model, typical of the Operating Systems literature, of the rate-of-update of each Web source. The presented model has been implemented into a new version of the Dynamo project: a middleware that assists in generating informative RSS feeds out of traditional HTML Web sites. This article describes Dynamo, an experimental architecture for automated data collection and RSS delivery of data from traditional HTML Web sites. Data extraction and caching is done by a third-party Web service that i) runs a Dynamo process of each Web site being monitored and ii) supplies subscribers with RSS feeds containing the extracted data. To be effective (i.e., deliver information timely) and scalable (i.e., handle Web sites that change fast, e.g. news and community portals) Dynamo needs and accurate fine-tuning of its *polling policy,* i.e., the frequency at which it downloads and examines a given Web page. Such problem, however, is not specific to Dynamo and it can be observed in any Web data extraction platform, e.g., LiXto, where publishing and data extraction are asynchronous.

This article describes the adaptive, ad-hoc polling polices that we have developed for Dynamo in the experimental setting of two popular (and busy) News portals: *CNN Most*

Recent[1] and *ANSA Top News*[2]. Differently from our previous work on Web Services connection policies [3, 15], in this context we consider *policies* as strategies by which the Dynamo Web Service minimizes the computational/networking burden without compromising on service. As will be described next, such strategies must be adaptive w.r.t. the dynamics of the Web site which is being monitored.

Dynamo is another effort on the wider topic of information extraction from HTML web sites (see, e.g. [7] for a review). Even though Dynamo, unlike LiXto [13], does not support fully automated and unsupervised data extraction, it is indeed for generating RSS feeds from legacy, i.e. closed-source, content management systems. Finally, Dynamo exploits the expressivity of the XQuery query language that allows the final user to submit complex queries over the collected data.

Overall, Dynamo provides a complete layout for the implementation of RSS Web services that interact with the traditional Web in an almost seamless way. Even though Dynamo is still a research project, a robust proof-of-concept implementation is under deployment for the large (more than 10,000 members) *Rete Civica Milanese* (RCM) on-line community[3]. Given the sheer size and the rate of change of that community site, Dynamo polling will be feasible only by the deployment of careful, *adaptive* polling policies, which are presented and discussed in this article.

This article is organized as follows: the next section describes the basic operation (input/output) of the Dynamo Web service. Then, in Section **4** we describe the software architecture in detail; hopefully such description will illustrate the potential of the *vision* underlying Dynamo. The following Section briefly reports of the two main applications now available on the Web. Finally, we describe the experiment with news portal and the process by which we could determine the ad-hoc policies.

2. The Dynamo project

The Dynamo project was started with the aim of creating an RSS service for readers of technical Web sites that were not offering such feature [4]. The RSS service would remain independent from the relative Web site but for the insertion of meta-tags in the pages. These meta-tags remain transparent to the reader but are essential to guide Dynamo in the data extraction phase. The next subsections describe in detail the input and output of Dynamo, the inner structure of the project and the interface of the REST-style service that Dynamo provides.

2.1. Structure of the HTML input. HTML documents contain a mixture of published text i.e., meaningful to humans, and of directives, in the form of tags, intended for browsers interpretation. Moreover, since the HTML format is designed for visualization purposes only, its tags do not allow sophisticated machine processing of the information contained therein.

The approach underlying Dynamo is to define a set of annotations in form of meta-tags, that will be inserted inside an HTML document in order give it semantic structure and highlight informational content. Dynamo meta-tags are used as annotations, to describe

[1]Accessible at *http://www.cnn.com/*

[2]Accessible at *http://www.ansa.it/*

[3]The RCM network is accessible from *http://www.retecivica.milano.it/*

Meta-tag	Description
<channel:title> ... </channel:title>	Channel title
<channel:description > ... < /channel:description>	Channel description
<channel:image url=" link=" title=" />	URL, link and title of an image associated to the channel
<channel:extension uri=" prefix="> ... </channel:extension>	Channel extension (e.g., publication date)
<item:link index="> ... </item:link >	item link
<item:description index="> ··· </item:description >	item description
<item:extension uri=" prefix="> ... </item:extension >	item extension (e.g., item publication date)

TABLE 1. The Dynamo meta-tags

and mark all interesting pieces of information. Such tags will drive the extraction and so-called *XML-ization* phases (see next section). The set of meta-tags Dynamo uses is listed in Table 1 below. Please see [5] for a detailed description of the Dynamo meta-tags. The meta-tags are enclosed in HTML comment tags, so they remain transparent to Web browsers and do not alter the original HTML structure of the document. The tagging schema of Dynamo is but an early instance of what are now called *microformats,* [2].

2.1.1. *Meta tags vs. dynamic XSLT transformations.* An obvious alternative to our approach to the treatment of existing HTML structures is that of applying, after the polling phase, some clever XSLT transformation [18] to the HTML file. Gottlob et al. LiXto Suite applies the schema described above, combined with sophisticated extraction policies that exploit the underlying tree-like structure of HTML. It should be considered, however, that applying XSLT transformations is possible (unless human intervention) only when the [X]HTML document is syntactically well-formed. Regrettably, such well-formedness assumption seems rather unrealistic to us, exp. for old documents developed against early HTML rendering engines e.g. MS Explorer. Vice versa, our solution relieves the webmasters from any time-consuming translation of her HTML documents into well-formed XHTML ones, which would then make a subsequent XSLT transformation successful.

3. Structure of the XML output

Once HTML documents are processed by our application, annotated semantic structures are extracted and organized into a simple XML format which will be stored and used as a starting point for document querying and transformation. This XML format has been simply called *XMLData*. This *neutral* format has also been introduced in order to avoid committing to one of the several RSS formats now available, e.g. RSS 1.0 and RSS 2.0. Indeed, we found it convenient for our application to create RSS feeds *on the fly* rather than storing them. This approach is also more flexible: supporting of new syndication formats (see for example, the Atom format) won't require re-designing of the lower levels of the application. The structure of the XML output is defined by the following XML Schema Definition [20]:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified">

  <xsd:complexType name="imageType">
    <xsd:all>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="link" type="xsd:anyUri"/>
      <xsd:element name="url" type="xsd:anyUri"/>
    </xsd:all>
  </xsd:complexType>

  <xsd:complexType name="extensionsType">
    <xsd:sequence>
      <xsd:any namespace="##any"
       processContents="skip"
          minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="channelType">
    <xsd:all>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="link" type="xsd:anyUri"/>
      <xsd:element name="description"
       type="xsd:string"/>
      <xsd:element name="image" type="imageType"
          minOccurs="0" maxOccurs="1"/>
      <xsd:element name="extensions"
       type="extensionsType"
       minOccurs="0" maxOccurs="1"/>
    </xsd:all>
  </xsd:complexType>

  <xsd:complexType name="itemType">
    <xsd:all>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="link" type="xsd:anyUri"/>
      <xsd:element name="description"
       type="xsd:string"/>
      <xsd:element name="image" type="imageType"
          minOccurs="0" maxOccurs="1"/>
      <xsd:element name="extensions"
       type="extensionsType"
          minOccurs="0" maxOccurs="1"/>
    </xsd:all>
    <xsd:attribute name="index" type="xsd:integer"
     use="required"/>
    <xsd:attribute name="id" type="xsd:string"
     use="required"/>
  </xsd:complexType>
  <xsd:complexType name="resourceType">
    <xsd:sequence>
      <xsd:element
       name="channel" type="channelType"/>
      <xsd:element
       name="item" type="itemType"
          minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="url" type="xsd:anyUri"
     use="required"/>
    <xsd:attribute name="rssId"
     type="xsd:string" use="required"/>
    <xsd:attribute name="timestamp"
     type="xsd:dateTime" use="required"/>
  </xsd:complexType>

  <xsd:element name="resource" type="resourceType">
    <xsd:key name="itemId">
      <xsd:selector xpath="item"/>
      <xsd:field xpath="@id"/>
    </xsd:key>
    <xsd:key name="itemIndex">
      <xsd:selector xpath="item"/>
      <xsd:field xpath="@index"/>
    </xsd:key>
  </xsd:element>

</xsd:schema>
```

4. The Dynamo architecture

Figure 1 shows the overall architecture of Dynamo. Our application is based on a modular structure, that maximizes the flexibility and the extensibility of configuration. Conceptually, Dynamo modules can be arranged over three layers:

- Physical Data Storage Level It is the lowermost level, which stores resources, and provides a means for retrieving and querying them. It can be implemented in various ways, using also established technologies like relational or XML database [6].
- Core Level. It holds the core part of the entire architecture, including the software components which implement the logic of information management and processing; each component can be implemented using different strategies or algorithms, and plugged into the system without affecting other components, i.e., by simply tuning the application configuration files.
- Service Level. It is the highest level, interacting with Web clients by means of REST Web services [10].

A more detailed explanation follows, starting from the Core level, the foundation over which our application is based.

4.1. The Core Level. The Core level is composed by several components defining how the application i) retrieves HTML resources, ii) processes to extract information about

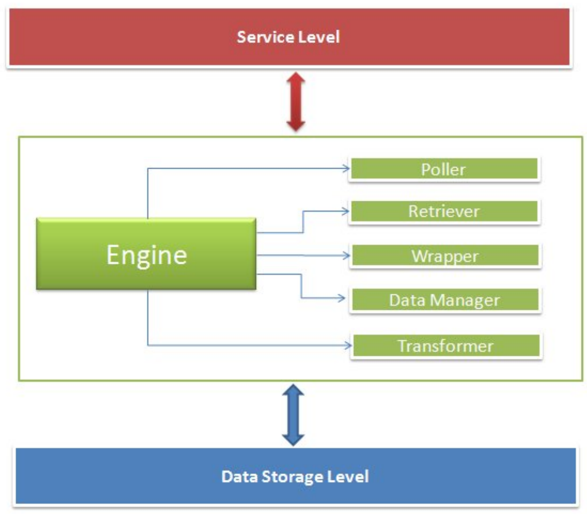


FIGURE 1. The module stack of Dynamo

channeling, iii) manages this new piece of information and finally iv)transforms and prepares it for client consumption:

- **engine:** the code that routinely invokes the Retriever and thus the whole polling process.
- **Poller**: it monitors changes in a set of HTML resources configured in a particular file, using some polling policy (see next section). Moreover, the poller has the important task of coordinating other components in the retrieving, extraction, and storing phases.
- **Retriever**: when invoked by the Poller, it captures the Web resource from its URL and makes it available to other components.
- **Wrapper**: it takes care of extracting the annotated semantic structures from the retrieved HTML resources, wrapping them in a new one, that is, assembling the extracted structures in a fresh, pure XML format, containing the desired informational content: the previously-described *XMLData* format. So, this component must produce a well formed XML document, ready to be stored by the Physical Data Storage level.
- **DataManager**: it acts as a gateway to the Physical Data Storage level, taking care of managing information in the form of the new XML documents previously created, storing them and permitting client components to query their contents.
- **Transformer**: it finally takes care of transforming the stored XML documents into the RSS format requested by clients, using XSLT transformations.

Typical parameters of this level can be changed simply modifying the corresponding parameters which are listed in some configuration files, in XML format. The configuration file of the Engine Component, for example, allows to set the type of polling policy of the

Collection path	Description
/db/resources	Root collection
/db/resources/headlines.rss	Collection holding XML resources related to the headlines.rss resource, i.e., its history
/db/resources/headlines.rss/123	XML resource, identified by its time-stamp, e.g., 123

TABLE 2. Collection examples

Web resources. Currently, the choice is between *flat,* i.e, constant over time, or *smart,* i.e., depending on the recent rate of updates. Other parameters are: the type of data manager (currently, the Exist native-XML database together with its connection parameters) and the format of the RSSes sent to Dynamo subscribers (currently RSS1 and RSS2).

4.2. The Physical Data Storage Level. The Physical Data Storage level can be implemented with various technologies: our choice has been to implement it using a native XML database. This choice allows us to store and manage XML documents produced by the Wrapper software component in their native format, and to use the powerful XQuery language for advanced content querying and aggregation. The native XML database is organized as a set of collections of XML resources, where the nesting of collections is allowed. In our application, we store XML resources as provided by the Wrapper software component, one collection for each resource. Each collection holds the various chronological versions of the resource: so, each collection effectively contains the history of the resource, all its informational content and a changelog.

When a new resource is to be stored, a check is done by the DataManager software component, in order to avoid duplicate resources. Two resources are considered to be different if their informational content changes. More precisely, they are different if changes to titles, links or descriptions of the resource channel or items are detected. Once stored, the resource is chronologically archived and ready for later retrieving and querying.

4.3. The Service Level. The Service level lets Web clients access the RSS feeds through the use of REST Web Services [17]. REST, an acronym for *Representational State Transfer,* is an architectural style which conceives everything as a resource identified by a URI. In particular, it imposes a restriction about of the URL defining the page info, that, in the REST view, are considered resources. Each resource on the Web, such as a particular part specification file, must have a unique URL (without GET fields after it), that totally represents it.

Thanks to this choice, the client (and all proxies/firewalls in between) may define the next state change by just inspecting the URL of current available forwarding links[4].

With respect to the well-known SOAP architecture[5], in REST we never access a method on a service, but rather a resource on the Web, directly using the standard HTTP protocol and its methods, To put it differently, in REST the hypertext linking controls the application state. This feature of REST allows greater simplicity and maximum interoperability with any Web client, either *thick,* like a desktop application, or *thin,* like a Web browser.

[4]For proxies/firewalls it suffices to inspect the header of the HTTP request.

[5]Please refer to [19] for an introduction to SOAP.

4.4. Accessing REST Web Services and resources. Adhering to the REST architecture and vision, everything is a resource and so any request and any search returns to the client an RSS resource, actually in the format of RSS 1.0 or 2.0, depending on the client choice.

In our application, these RSS resources are accessed through HTTP requests, using the GET method of HTTP 1.1 protocol; clients can ask for:

- A list of collections of RSS resources, each representing the chronological history of a resource.
- A list of RSS resources contained in a given collection.
- An RSS resource, identified by an index.
- An RSS resource containing only up to a given number of items, starting from the most recent one.
- An RSS resource obtained by querying a collection of resources, searching for keywords in titles, links or descriptions of items.

The GET method, in principle, should not modify the original resource. A detailed description of how REST Web Services and resources are accessed follows.

/resources[?type=rssType]. Accesses an RSS resource listing all collections of resources that clients can request and query. The optional *type* parameter identifies the RSS type of the requested resource.

/resources/rssId[?type=rssType]. Accesses an RSS resource listing all resources contained in the collection identified by the resource id, the *rssId* URL section. The optional *type* parameter identifies the RSS type of the requested resource.

/resources/rssId?index=n & [type=rssType]. Accesses an RSS resource identified by its *rssId* and the *index* parameter, that is the index number into the chronological history: use "1" for the first resource (the most recent one), "2" for the second and so on. The optional *type* parameter identifies the RSS type of the requested resource.

/resources/rssId?max=n & [type=rssType]. Accesses an RSS resource identified by its *rssId*, containing only up to *max* items. The optional *type* parameter identifies the RSS type of the requested resource.

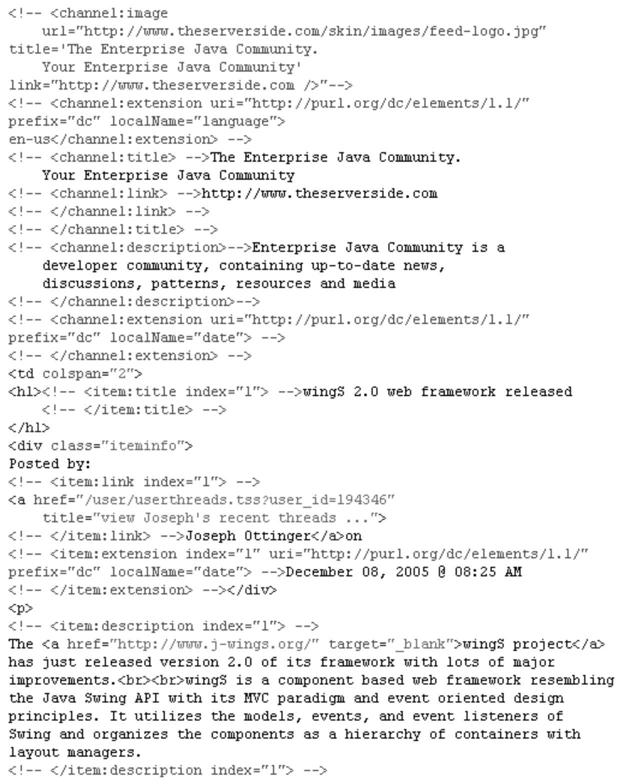
Complex queries. The following query:

```
/resources/rssId?max=n & [type=rssType]
                       & [(title| link | description | desc)=value]
                       & [op=(and| or)]
                       & [(title| link | description| desc)=value]
                       & ...
```

is intended to query all resources identified by the given *rssId*, requesting only up to *max* items and combining, using logical "and/or" operators, searches for title, link, or description of items. The optional *type* parameter identifies the RSS type of the requested resource.

5. The application at work

To illustrate how our application works we consider a fragment of a HTML document taken from the reference Web site *www.theserverside.com.* After the insertion of the meta-tags, the fragment looks as in Figure 2. Then the fragment is converted in XML format and, if not already present in the database, is stored in the appropriate collection of the database. Upon request from the client, the XML file is extracted and converted into one

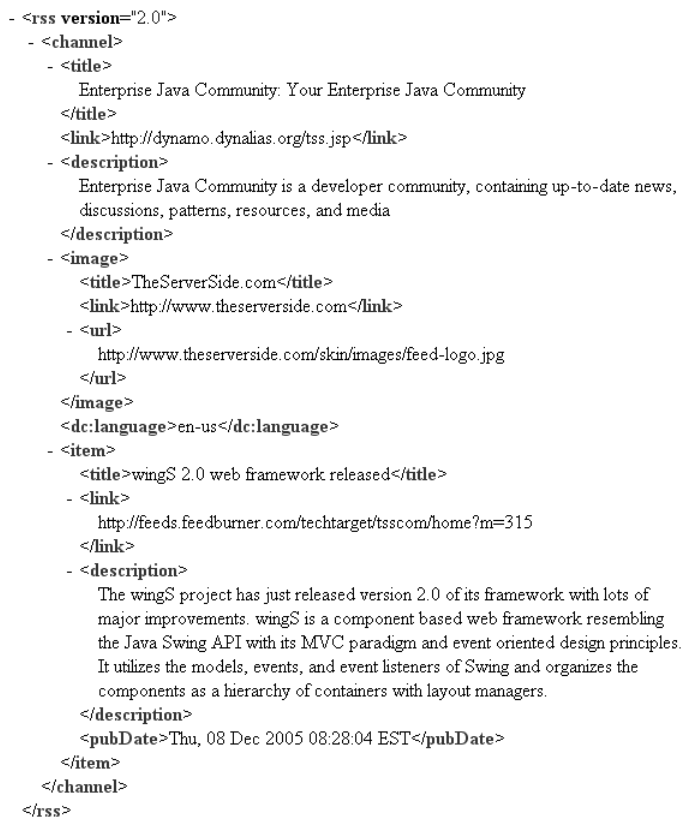


```
<!-- <channel:image
    url="http://www.theserverside.com/skin/images/feed-logo.jpg"
title='The Enterprise Java Community.
    Your Enterprise Java Community'
link="http://www.theserverside.com />"-->
<!-- <channel:extension uri="http://purl.org/dc/elements/1.1/"
prefix="dc" localName="language">
en-us</channel:extension> -->
<!-- <channel:title> -->The Enterprise Java Community.
    Your Enterprise Java Community
<!-- <channel:link> -->http://www.theserverside.com
<!-- </channel:link> -->
<!-- </channel:title> -->
<!-- <channel:description>-->Enterprise Java Community is a
    developer community, containing up-to-date news,
    discussions, patterns, resources and media
<!-- </channel:description>-->
<!-- <channel:extension uri="http://purl.org/dc/elements/1.1/"
prefix="dc" localName="date"> -->
<!-- </channel:extension> -->
<td colspan="2">
<hl><!-- <item:title index="l"> -->wingS 2.0 web framework released
    <!-- </item:title> -->
</hl>
<div class="iteminfo">
Posted by:
<!-- <item:link index="l"> -->
<a href="/user/userthreads.tss?user_id=194346"
    title="view Joseph's recent threads ...">
<!-- </item:link> -->Joseph Ottinger</a>on
<!-- <item:extension index="l" uri="http://purl.org/dc/elements/1.1/"
prefix="dc" localName="date"> -->December 08, 2005 @ 08:25 AM
<!-- </item:extension> --></div>
<p>
<!-- <item:description index="l"> -->
The <a href="http://www.j-wings.org/" target="_blank">wingS project</a>
has just released version 2.0 of its framework with lots of major
improvements.<br><br>wingS is a component based web framework resembling
the Java Swing API with its MVC paradigm and event oriented design
principles. It utilizes the models, events, and event listeners of
Swing and organizes the components as a hierarchy of containers with
layout managers.
<!-- </item:description index="l"> -->
```

FIGURE 2. An HTML fragment after the insertion of meta-tags

of the two formats currently supported by our application, that is to say RSS 1.0 or RSS 2.0. For sake of brevity we present here only the RSS2 version of the output (see Figure 3). It should be noted that in order to work properly our application strongly relies upon the insertion of meta-tags, which can be accomplished with a very little effort and/or change in currently available content management and publishing systems. It is beyond the scope of our application to be able to discover the appropriate patterns inside the HTML documents and *automatically* insert the meta-tags, which can be successfully done by our application only if the HTML document never changes in its internal structure.

Let us now see how a user interacts with the application. First of all, a user can verify the available RSS resources through his Web browser. She obtains a list of the available resources which can be formatted in one of the two currently supported formats, namely RSS 1.0 or 2.0. Following the link, the user gets the archive of the resource, chronologically ordered from the newest to the oldest. Our application allows also to aggregate RSS items and to query them. It is then possible to keep up-to-date by requesting a fixed number of the newest items. It is also possible to request the newest items containing a certain keyword in the title or in the description.



```
- <rss version="2.0">
  - <channel>
    - <title>
        Enterprise Java Community: Your Enterprise Java Community
      </title>
      <link>http://dynamo.dynalias.org/tss.jsp</link>
    - <description>
        Enterprise Java Community is a developer community, containing up-to-date news,
        discussions, patterns, resources, and media
      </description>
    - <image>
        <title>TheServerSide.com</title>
        <link>http://www.theserverside.com</link>
      - <url>
          http://www.theserverside.com/skin/images/feed-logo.jpg
        </url>
      </image>
      <dc:language>en-us</dc:language>
    - <item>
        <title>wingS 2.0 web framework released</title>
      - <link>
          http://feeds.feedburner.com/techtarget/tsscom/home?m=315
        </link>
      - <description>
          The wingS project has just released version 2.0 of its framework with lots of
          major improvements. wingS is a component based web framework resembling
          the Java Swing API with its MVC paradigm and event oriented design principles.
          It utilizes the models, events, and event listeners of Swing and organizes the
          components as a hierarchy of containers with layout managers.
        </description>
        <pubDate>Thu, 08 Dec 2005 08:28:04 EST</pubDate>
      </item>
  </channel>
</rss>
```

FIGURE 3. The fragment in RSS 2.0 format

6. Applying Dynamo

The first deployment of the Dynamo has been *dynamo.dynalias.org* Web service. This service, which is free to subscription, gathers news and announces from the Web portals *www.serverside.com* and *www.java.net,* each of which publishes about 4-5 news (in plain HTML format) every day. By now Dynamo publishes the news feeds, in both RSS1 and RSS2 formats, taken In order to avoid any interaction with the portals we resorted to download the HTML pages containing the news, insert the meta-tags we defined and submit them to the entire procedure of extraction, storing and publishing.

The second and most important application of Dynamo has been the implementation of RSS feed service for the Milan Civic Network[6], an on-line community of more than 10.000 citizens. Until now, the on-line community has been relying of a closed-source content-management product, developed in the 90's, that serves large communities from the inception. Since adding an RSS service from within the software was not possible, Dynamo was applied to provide it as a third-party service. Even though this service is still

[6]*http://retecivica.milano.it/*

being tested [8] the results are encouraging and show that the application can escalate up to the management of a large community Web site with hundreds of posting per day.

7. Scalability issues

A typical problem in the design of an architecture like ours consists in the forecast of all possible critical elements that can raise as work loads become bigger and bigger. First of all it must be considered that an instance of Dynamo can be installed for each Web server. That is, in order to extract and aggregate data from several sources, an equal number of Dynamo processes should be executed, albeit not necessarily on the same server. Another, even more severe, possible limitation to the performance of the proposed architecture is represented from the bandwidth required to forward the requests for updates, because in those cases of non regular updates a lot of requests would be useless thus resulting in wasting bandwidth. This is the reason of an improvement we are studying, that is a polling policy able to fit the frequency of the updates of the news from the Web servers: this policy, we called smart polling policy, adjusts the frequency of the requests for updates to the frequency with which Web portals generate new information.

Another factor that may affect the overall performances of DynamoNews is the host database management system, which is in our implementation is *eXist,* an Open Source native XML database whose performance seems not up to those of the DBMS normally adopted to service Web portals. To avoid long response times even for simple queries, we have implemented a cache engine where the most frequently requested queries are stored.

8. Polling policies

Polling policies are mainly concerned with the choice of the frequency of update requests to web servers; update frequency is a crucial feature to make RSS effective and a central point in our application, which can guarantee a timely generation of RSS feeds and, even more important, an appropriate use of network bandwidth.

We introduced two different polling policies, which can be chosen and plugged in our application independently from each other. The first is called ”flat” polling policy, as it does not depend from update frequency, while the second is called ”smart”, as it tries to fit the update frequency of each web portal. *We compare our estimate with the ”real” publication time contained in the header, the so-called channel, of the feeds.* It is possible to reconfigure at run-time the Poller component of the application in order to switch policy at runtime.

With flat polling, Web resources are queried for updates at regular time intervals which can be modified. It is the simplest strategy and it well applies to regularly updated information.

8.1. Self-adaptive polling policy.

The smart polling policy adjusts the frequency of the requests for updates to the frequency with which web portals generate new information. As a result, we obtain timely RSS feeds, a lesser number of useless requests and a lesser use of network bandwidth. To compute the frequency of the requests of updated Web documents we first make an estimate of the frequency. Then this estimate is compared to the *real* frequency with which Web documents are updated or newly generated. Both the estimate and the *real* times are used to compute a new estimate. That is:

$$\tau_{n+1} = \alpha\tau_n + (1-\alpha)t_n \tag{1}$$

where τ_{n+1} is the estimate at the $(n+1)$-th iteration, τ_n the estimate at the n-th iteration, t_n the *real* frequency at the n-th iteration. The parameter α, whose value stands in the interval between 0 and 1, represents the relative weight of the previous estimate w.r.t. the *real* frequency. As τ_{n+1} takes into account the previous iterations, α represents the importance given to previous iterations. Some considerations about the parameter α. Its value, comprised between 0 and 1, influences the velocity with which the frequency of polling equals the frequency with which web portals publish new information. We found, on the other side, that its value does not influence the *c*onvergence of the frequency of polling to the frequency of publication, but only its velocity. Analogous results can be found in literature, even if in rather different situations. See, for example, the algorithm of processes scheduling known as *s*hortest job first [1], as well as the weighted mean frequently used in the iterative calculations typical of the Self-Consistent Integral Theory in Statistical Many-Body Thermodynamics [11], where it is also known as *mixed scheme.*

8.2. Fine-tuning of the α parameter. The main problem we faced in developing the polling polices consisted in finding the best value for the α parameter. It is a well-known fact of Single Exponential Smoothing theory that α must be chosen between 0 and 1, but there are no hints on what the optimal value would be. Moreover, it is not always easy, when looking at time series to factor out trends and seasonal aberrations. In order to gain some insight, we have considered historical data (time series) collected as follows. Two popular news portals, maintained by respected and well-known press agencies, have been tracked continuously over several days to determine the rate of update of their front page. Those Web sites were *www.cnn.com* and *www.ansa.it.* The latter is in Italian only and obviously generates less traffic. However, the dynamics of the news would in principle be the same.

Figures 4 and 5 show, over the time-scale of minutes, the distance between the publishing of two successive news. We found that historically such period is linear and relatively stable *around* the overall average. Hence, we decided to adopt the Single Exponential Smoothing estimation method[7].

Within this empirical framework, policies may differ on the choice of the α parameter that best estimates the time between two successive web page updates. Given the time series collected from the two news portals described above, we computed the mean square error (MSE), which measures the discrepancy between estimated and actually observed values. The value of MSE over a time series is given by the formula:

$$MSE = \frac{\sum(\tau_i - t_i)^2}{n} \tag{2}$$

[7]Single exponential smoothing method is appropriate for series that move randomly above and below a constant mean with no trend nor seasonal patterns [9].

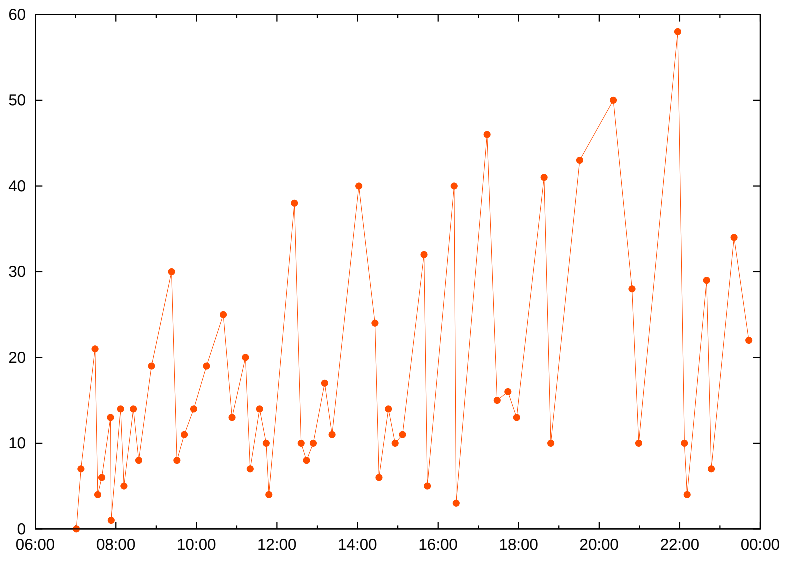


FIGURE 4. A Typical time series from ANSA news site

where t_i represents the observed data and τ_i represents the relative estimation. Since τ_i depends on α, MSE is effectively a function of α. To find the parameter values that minimize MSE we proceeded applying the golden ratio method [16], which consists in comparing the values of the function to be minimized over three points for each considered interval. We recall that with this method, equivalent to the bisection method for finding roots of a function, we successively narrow brackets around the minimum by upper bounds and lower bounds. The most efficient bracket ratios are in a golden ratio; from which this technique takes its name.

The value of α that minimizes MSE over the available time series was the best estimate of the time elapsing between two updates of the page. Figure 6 shows a typical example of the evolution of $MSE(\alpha)$.

8.3. Experimental validation. The self-adaptive approach described in the previous section has been experimented on the two very active news portals *CNN Most Recent* and *Ansa Top News*[8] .These two sites where chosen because i) they provide a somewhat equivalent service but do not conflict since they serve different linguistic domains and ii), more importantly, have a very high update rate. As an instance, on special events CNN Most Recent may be is posting more than one news per minute, and the average is always around 2-3 minutes. Ansa Top News, vice versa, is slower and on average publishes 3/4 news per hour.

[8]*http://www.ansa.it/*

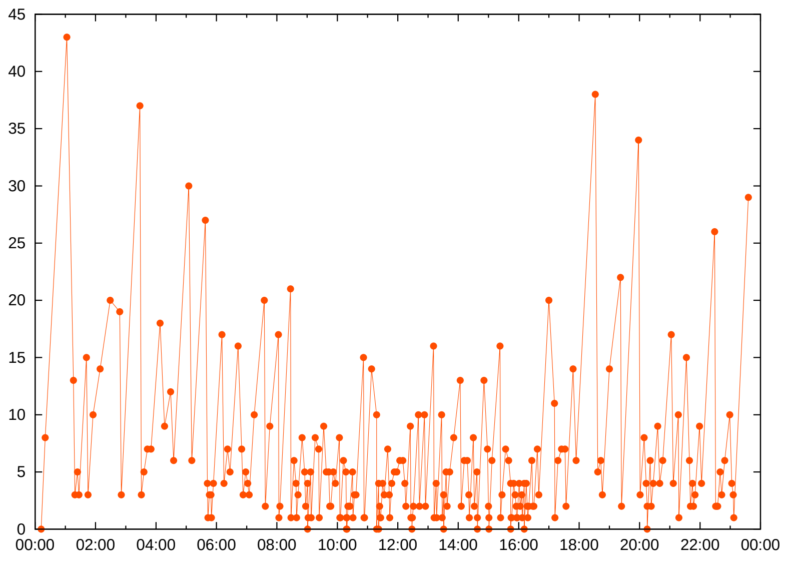


FIGURE 5. A Typical time series from CNN news site

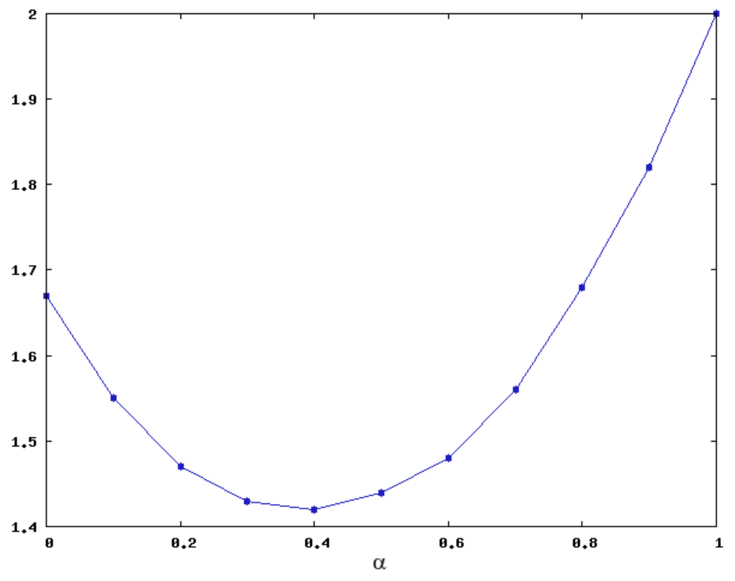


FIGURE 6. An example of $MSE(\alpha)$ for varying values of α

	Test no. 1	Test no. 2	Test no. 3	Test no. 4	Test no. 5	Test no. 6
	10:35	14:25	13:51	17:02	19:47	07:58
	10:36	14:47	13:52	17:16	19:56	08:15
	10:36	15:19	13:54	17:31	20:11	08:31
	10:47	15:29	13:56	17:45	20:32	09:24
	10:54	15:41	13:57	17:55	21:04	10:08
	10:56	15:44	14:13	18:25	21:34	10:50
best α	0.005	0.236	0.994	0.618	0.145	0.005
estim.	4 min	12 min	15 min	22 min	23 min	35 min
real time	2 min	14 min	9 min	11 min	51 min	32 min

TABLE 3. Some experimental values from CNN and ANSA

By analyzing two news portal with marked differences in update rate we believe we could make our results independent from each of them. In Table 3 we report the results of some experiments together with the obtained values of α and the subsequent estimated values of the publication times vs real publication times. It can be noted that best agreements with real publication times are obtained when time series are regular enough, whereas strongly irregular publication times cannot be followed by our algorithm.

9. Conclusions

In this article we have described a Web application that generates and manages RSS feeds extracted from HTML Web documents. The proposed Dynamo architecture is intended to be applicable to arbitrary Web sites, provided that the Web administrator agrees to add the proposed meta-tags to the commented part of each page.

By attaching a polling policy to individual sources, Dynamo makes it possible to specify and implement flexible gateways between the traditional Web, where information is marked-up simply by HTML formatting commands, to the Semantic Web, in the spirit of Gottlob et al. LiXto project [12].

The mixed estimate schema adopted here exploits only partially the information available on the historical frequency of the source. That is, only the duration of the last period between two updates is considered. In principle, historical data on update frequencies could reveal typical patterns of update frequency. We believe, however, that such frequency patterns are not easily applied at arbitrary time periods, and even less to different data sources.

Our solution requires minimal and totally transparent changes on their HTML pages. The data of interest is routinely *polled* from the actual sources by standard HTTP querying. Subsequently, the so-created Web service can be queried with REST-style sessions that extract the aggregated data. Clearly, users may want to define personalized polling polices. Since such policies normally (and logically) reside on the client-side, they are not considered in this work.

Indeed Dynamo polling policies are still limited limited in the sense that that the only permitted action is *polling*. Yet, they may be complex to define since they must be on forecasting theory and on empirical search for suitable parameters. Furthermore, to the best of our knowledge no automated Web data extraction tool is able to adapt polling frequencies automatically.

One factor preventing the spread of the Semantic Web is —among other things— the complexity of extracting, from existing, heterogeneous HTML documents machine-readable information [14]. Although our project addresses only a fraction of the Semantic Web vision, our management of HTML documents needs some technique to locate and extract some valuable and meaningful content.

Acknowledgments

S. Bossa designed and implemented the first version of Dynamo; C. Giacobbe and G. Trimarchi also contributed within their graduation dissertation to the experiments that determined the optimal polling policies described in this article. Thanks to L.A. Ripamonti and the staff of RCM, the *Milan Civic Network* for working with us in the deployment of Dynamo on their community portal. One of us (GF) wishes to thank Prof. P. V. Giaquinta for helpful discussions about mixed schemata.

References

[1] Greg Gagne Abraham Silberschatz, Peter Galvin. *Operating System Concepts VI Edition*. John Wiley & Sons, 2002.

[2] John Allsopp. *Microformats: Empowering Your Markup for Web 2.0.* friends of ED, March 2007.

[3] Elisa Bertino, Alessandra Mileo, and Alessandro Provetti. Pdl with preferences. In *Proc. of POLICY 2005 Conference, IEEE Press*, pages 213–222, 2005.

[4] Sergio Bossa. *Towards the Semantic Web: a platform for dynamic generation, query and archival of RSS contents (In Italian)*. Graduation Project in Computer Science, Univ. of Messina, *http://informatica.unime.it/*, 2005.

[5] Sergio Bossa, Giacomo Fiumara, and Alessandro Provetti. An architecture for policy-based rss polling. In *Proc. of Workshop from Objects to Agents (WOA06)*, 2006.

[6] Ronald Bourret. Xml and databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[7] C. Chang, M. Kayed, M.R. Girgis, and K. Shaalan. A survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18:1411–1428, 2006.

[8] F. DeCindio, G. Fiumara, M. Marchi, A. Provetti, L. Ripamonti, and L. Sonnante. Aggregating information and enforcing awareness across communities with the dynamo rss feeds creation engine: preliminary report. In *Proc. of COMINF06, an OTM 2006 Workshop*, volume 1, pages 227–236. Springer LNCS 4277, 2006.

[9] Francis Diebold. *Elements of Forecasting, 4th edition*. South-Western College Publishing, 2006.

[10] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation, Univ. of California-Irvine, 2000. Chair-Richard N. Taylor.

[11] Paolo V. Giaquinta and Santi Prestipino. Density-functional theory of a lattice-gas model with vapour, liquid, and solid phases. *Phys J.*, 15:3931–3956, 2003.

[12] G Gottlob, R. Baumgartner, and S. Flesca. Visual web information extraction with lixto. *Proc. of VLDB Conference*, 2001.

[13] G Gottlob and et Al. The lixto data extraction project – back and forth between theory and practice. *Proc. of PODS, Principles of Database Systems*, 2004.

[14] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM*, 51, 2004.

[15] Massimo Marchi, Alessandra Mileo, and Alessandro Provetti. Specification and execution of policies for grid service selection. In *Proc. of the European Conference on Web Services 2004 (ECOWS04), Springer Verlag-LNCS 3250*, pages 102–115, 2004.

[16] W.H. Press, S.A. Teukolsky, W. T. Vetterling, and B.P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing, second edition*. Cambridge University Press, Cambridge, 1999.

[17] James M. Snell. Resource-oriented vs. activity-oriented web services. http://www.ibm.com/developerworks/xml/library/ws-restvsoap/, 2004.
[18] W3C. Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/xslt, 11 1999.
[19] W3C. Soap version 1.2 part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, 06 2003.
[20] W3C. Xml schema part 0: Primer version 2.0. http://www.w3.org/TR/xmlschema-0, 10 2004.

Giacomo Fiumara, Massimo Marchi, Alessandro Provetti
Università degli Studi di Messina
Dipartimento di Fisica
Salita Sperone 31. Contrada Papardo
I-98166 Messina, Italy
* **E-mail**: gfiumara@unime.it

Presented: November 29, 2007
Published online: December 16, 2008