MidWire: Middleware for Web Information Reuse

Nader Mohamed and Jameela Al-Jaroodi
The Faculty of Information Technology, UAEU
Al Ain, P.O. Box 17551, UAE
nader.m@uaeu.ac.ae and jaljaroodi@gmail.com

***Abstract*—Nowadays the World Wide Web (Web) is the ultimate source of information. Everyone, regardless of their backgrounds or computer experience rely on it to get the information they need. However, most of the information needed may not be readily accessible in the huge row repository of data on the Web. This paper introduces the concept of middleware for retrieving web information. MidWire (MIDdleware for Web Information REuse) is developed to facilitate the use of public Web information by other user applications. The Web provides huge online and updated information related to different aspects of businesses and environments such as stock prices, currency exchange rates, interest rates, and weather information. All this information can be reused for other applications through the proposed middleware. MidWire deals with different types of Web information sources to provide the user applications' input. It also provides different advanced services such as caching and notification. This paper covers the challenges, the architecture, the services, and evaluation of MidWire.**

***Index Terms*—Middleware, WWW, Web Information Integration, Web scraping, Information Reuse, Information Retrieval**

I. INTRODUCTION

The Web carries a great wealth of information that has become important for almost everyone. Peoples need for Web information vary; some need static information which can be easily found and used. However, many rely on the real-time availability of dynamically and frequently changing data. Getting this information, filtering/analyzing it in real-time is very hard and time consuming. The Web is full of such live and dynamic information mostly provided as text such as news, multimedia such as Internet TV, or numbers such as stock, currency and weather information. This makes it viewer-friendly, yet it becomes hard to capture and manipulate that data automatically.

From the user perspective, the availability of live data may be crucial for the success of their applications. For example, the need to retrieve instant values in stock markets to decide on trade or investment options [1]. Other examples also include web information monitoring [2], for various needs such as online currency exchange information and web-based news updates. These applications cannot deliver the correct results without such information being made available in real-time. Much of this information is publicly available through dynamic HTML documents, XML documents, web services, and RSS feeds. A small portion of this information is made available through XML documents and web services. These can be easily integrated into other applications due to their structured format and available metadata. Various APIs are available to integrate information from XML or web services with the user applications. For example, a user interested in currency exchange rates can integrate any of the available web services like CurrencyConvertor [3] with his/her own application to get the latest rates all the time.

Huge information is mainly available in dynamic HTML documents. However, HTML documents are geared for the human viewing and lack the structure and the metadata that can help in automating their integration with other applications. Some efforts were made to facilitate using this type of information such as urlINFO [4], a Java class that can be used by Java applications to retrieve live information from HTML pages in real-time. However, this is a single user model that requires the user to have programming experience and cannot handle multiple users or requests at the same time. In addition, it does not provide advanced integration services such as caching, notification, and fault tolerance.

As more tools pop up to help retrieve Web information and help users integrate them into their own applications, many challenges also arise. These challenges involve the retrieval, organization and integration of the information efficiently and in real time. One of the contributions of this paper is to identify and discuss these challenges and then introduce the middleware framework we designed to facilitate Web information reuse. This middleware will help enhance the development and operations for utilizing the available public information for new applications. As we will discuss in the design, the framework, the features involved, focus mainly on creating tools and methods to extract the useful and required data and inform the users of the specified changes when the occur. However the overall architecture covers different types of data sources including XML files and web services. This makes the framework more generic and useful for any type of input.

In the rest of the paper, Section II provides background information and related work on Web information retrieval. Section III offers an overview of the Web information reused model and Section IV discusses the challenges of Web information reuse. Section V discusses the MidWire services while Section VI provides more details of the middleware architecture. A prototype implementation is provided in Section VII.

doi:10.4304/jsw.7.7.1451-1461

Section VIII provides some performance evaluations for the proposed middleware. Finally, Section IX concludes the paper and highlights some possible future work.

II. Background on Retreiving Web Information

The Web offers a multitude of public information in various document formats and there exist several tools and methods to access it. We can get the information from web services [5], XML files [6] or HTML documents [7]. Web services provide web APIs that can be accessed over the Web, and executed on a remote system hosting the requested services. These remote systems provide different services that provide reusable information about different aspects or products. For example, some stock markets may provide web services to supply current stock prices. Banks may provide web services to supply information about loans or currency exchange rates. If a specific piece of information is needed by an application, the user can easily define a variable and link it with the corresponding web service which provides the required information using the available API. The main problem with web services is that their availability is still limited and not all types of information are provided by them. Most of the useful information on the Web is still available in HTML format which cannot be easily retrieved and integrated with other applications in real-time. When XML was introduced, the use of self-defining structures promised an easy way to deal with and retrieve content. However, XML documents are not as common as, HTML documents and thus most of the information is still inaccessible this way. Unlike XML documents, HTML documents do not have any semantics for their content. Thus, obtaining specific data from a dynamic HTML document for reuse in other applications is a complex task. It is very difficult to identify the required parts of the data, retrieve it and then dynamically integrated into a user application.

Some work was done at different levels to try to benefit from the Web HTML documents. One example is developing an approach to link the large amount of data that are currently available in HTML documents to the Semantic Web ontology [8]. Another example is developing an approach that automatically captures the semantic hierarchies of HTML tables [9]. In addition much of the work on information retrieval has been applied on Web information to capture certain aspects from HTML documents. The main goals of information retrieval are efficiency, accuracy and ease of use [10] and there are many ways to retrieve specific information from HTML documents based on word, sentence or semantic matching. In [11] the authors produce a learning algorithm for retrieval based on previous experiences. Terrier [12] is another example that offers a platform to build and experiment with large-scale information retrieval methods. It offers a predesigned indexing architecture to help in the search and retrieval operations. In addition, in [13] the authors offer a learning model to pair usable wrappers with web pages to extract data from the HTML documents. In this case, the learning program is trained to identify page-independent characteristics that help in the extractions. Also some work was done for page summarization where important data is collected and a summary of the page is presented to the user. Temporal summarization [14] is an advanced model where data changes in the pages of interest are also kept in the summaries. Another approach is based on Web data warehousing [15] where the proposed method analyses websites and creates Web tables that keep track of the important data and changes along with the URL links for user reference. Most retrieval approaches however, require highly complex models that may work for some web pages and not others. In addition adapting different models will result in different monitoring systems and users will be more confused. Furthermore, many of the approaches we mentioned do not offer functionalities that could cater to multiple requests and/or multiple simultaneous users. In addition, these approaches do not incorporate user notification mechanisms that will help inform the users of events of interest when they occur. Thus users will still have to continuously monitor the results produced by these services for what they need.

In another approach researchers rely on collaborating agents to monitor changes in web pages and notify interested users of these changes [16]. More work on multi-agent data Web information gathering was done and several approaches were proposed such as the three-tier architecture [17] where the middleware brokers offer cooperation capabilities among multi user agents and multi resource agents. Another example is the autonomous methodology to obtain domain-specific information that can be integrated together to form a useful data repository matching the users' needs [18]. The Do-I-Care agent [19] also provides a tool to monitor page changes and classify them based on user information to be of interest or not. Some other research effort was focused on a different direction where attempts to transform HTML documents to another format are done to satisfy specific applications. One example of this transformation is from HTML Product Catalogues source code and images to RDF [20]. The multi-agent approach and the middleware approaches will help further enhance such systems by allowing for more modular design and help accommodate more complex requirements. Yet again many of these are still limited.

Further work also emerged in what is being known as scraping (data, screen and web scraping) [21]. This relies on creating techniques that will treat the data, screen or web page as a source that is scanned then the information registered is passed as input for other systems or services. One example of currently available is ScraperWiki [22], which uses programming models to help users write their own scraping models to get their required data. Other researchers also offered different approaches in Web scraping such as those in [23] and [24]. Most scraping techniques require the user to have programming experience to gain high level benefits, yet there are some efforts to offer simpler GUI models to help such as Mozends [25], which offers point and click interface to

select the web pages and data to extract, yet it cannot handle complex choices and high detailed requirements. Once more, the Web scraping approaches offer an excellent start for building services to obtain user-defined information from Web pages, yet again the available approaches handle the service on an individual user basis.

In general, there are various issues to handle when trying to extract Web data and integrate them with other applications [26]. For example, the sources are constantly changing, No clear semantics are available, varying naming conventions and difficulty in extracting HTML data. The approaches we discussed above try to solve some of these issues; however, none of them can provide a full usable solution. In our approach, MidWire, we try to offer a generic middleware framework that will support data extraction from multiple sources, accommodate for dynamic changes and also work with multiple users and requests efficiently.

III. WEB INFORMATION REUSE EARLY MODEL

As we discussed in Section II, Most useful information is available in HTML documents and these do not offer enough metadata to find and use the requires information. As a result many models appeared to monitor web pages and extract certain information as requested by the user. However, many cannot handle dynamic information and frequent changes. In addition, most provide the information in its raw form thus it cannot be directly used as part of other applications. Dealing with live data poses more problems and issues.

We recently developed a simple and efficient approach for retrieving live HTML-based Web information [4]. The main idea is based on finding fixed titles or headers that appear in browsers for HTML documents directly or semi-directly before the needed dynamic information. These fixed titles or headers are used as reference points to know the position of the required dynamic information. The proposed approach is developed as a Java class, urlINFO [4]. Multiple objects can be created from this class for different Web HTML documents that contain some of the required information. A number of techniques were developed to find this information in any HTML document. These techniques are implemented in a set of methods listed in Table 1.

All these techniques can be used to retrieve Web information for use in new applications. As soon as the fields are identified the *get* or *getWI* methods with the right arguments are used allowing the application to retrieve the required information. Users can target any HTML document on the Web to retrieve the information they need.

The different methods are designed to cover all possible access modes for the data. Using these methods, we can identify the location of the data and retrieve it as one or more variables. These variables are then made available for reuse in other applications. As a result, it will be possible to get the latest value available for the applications.

TABLE 1.
METHODS TO RETRIEVE HTML-BASED INFORMATION.

Method	Description
get(header)	To return the next field directly after the defined header. The search starts from the beginning of the page.
get(n,header)	To return the next field directly after the defined header appears *n* times. The search starts from the beginning of the page.
get(n,header, i)	To return the field after skipping i fields after the defined header appears *n* times. The search starts from the beginning of the page.
getWI()	To return the next field from the current read pointer position.
getWI(i)	To return the field after skipping *i* fields from the current read pointer position.
getWI(header)	To return the field located directly after the specified header from the current read pointer position.
getWI(n, header)	To return the field after the occurrence of the header *n* times from the current pointer position.
getWI(n, header, i)	To return the field after skipping *i* fields after the defined header appears *n* times from the current pointer position.

Here we rely on middleware to facilitate access, reuse and integration of available public information with application programs designed to meet the specific needs of users. Generally, different middleware platforms were created to add new values for different systems such as enterprise systems [27], cluster computing [28], wireless sensor networks [29], mobile ad hoc networks [30], and robotics [31]. The main research goals in middleware platforms are to develop simple mechanisms, approaches, and methodologies that add value to existing computer systems, networks, and distributed applications. This value can be in the form of scalability, reliability, availability, usability, extensibility, manageability, reusability, stability, efficiency, autonomicity and integrity. The mechanisms are usually based on the reuse of existing methods, protocols, software, and systems to add the needed values.

IV. WEB INFORMATION REUSE CHALLENGES

As the first model was designed we came across many challenges that need to be addressed to implement a complete middleware for Web information reuse. In this section, we discuss these challenges that make it hard to retrieve live Web information and reuse it as part of other applications. Efficient solutions are required to simplify the process of the integration and to smooth the communication among the different applications and the needed Web information. These challenges include:

- **Interoperability:** the Web mainly provides information in the form of HTML documents, XML, documents, and web services. On the other hand, most applications still use CORBA, RMI, and DCOM to facilitate integration. It is very difficult for example to allow a CORBA, RMI, or DCOM based application to reuse the Web Information provided in HTML, XML, or even some web services. Web information providers do not support CORBA, RMI, and DCOM interfaces through the Internet since these use special port numbers that are typically disabled by firewalls.

Applications that support web services can directly integrate themselves with the Internet to get the required information. Web services overcome the disabled ports problem by using the HTTP protocol for communication. HTTP usually uses port 80 which is generally enabled by most firewalls. Unfortunately, not all Web information is provided by web services. Furthermore, not all applications can support web services yet. In addition, XML provides some structure to the data made available; however, just as in web services, these are not very commonly used over the Web. To date, most of the information is still provided in HTML documents, which imposes a challenge on how to extract the required parts to be used by another application.

- **HTML Format Changes:** most information is still available on the Web in HTML format. Unlike XML documents, HTML documents are unstructured and have no semantics for the fields present in the document. As a result, integrating an HTML document that contains dynamic information with a local application can be a very difficult task. Somewhat, it is possible to implement a solution for the integration by extracting the required information from the relevant HTML documents based on the knowledge of the structure of the document and the position of the required information in the document. This allows an application to identify certain values based on their relative position to some fixed items in the document like labels or specific texts. However, the structure and the positions may change at any time, which makes the extraction method useless after the change. In addition, dealing with multiple HTML documents with different structures can be a very intricate task. Any changes in any used HTML document will require some changes in the local application that uses this document.
- **Distributed Information/Servers:** a local application may require some information that is distributed over multiple web servers located in different places. These servers may support different mechanisms to provide information such as through HTML documents, web services, or XML documents. At the same time, the response times for the requests from the client/local application asking for certain information from these servers may differ. Therefore, it becomes very difficult for the local application to deal with all the heterogeneity in the delivery mechanisms, in the response time, and in the number of the servers. This imposes a great challenge on the application developer to account for all these differences and ensure efficient operation of the information integration.
- **Highly Dynamic Information:** the required information provided by HTML documents, XML documents, or web services can be very dynamic. This causes the required information to rapidly change. For some applications, it is required to capture all changes that occur over time. One example is a stock price displayed in a dynamic HTML page. That price may change every two seconds. At the same time, some application may require registering all changes to that price to perform some calculations, analysis or make some decisions. Implementing the methods in the local application to get all changes in some fields in time and keep track of these changes continuously can be a very complicated task.
- **Fault Tolerance: s**ome web servers may be unavailable for some time due to different reasons including overloaded servers, network problems, and server maintenance. In addition, servers may have varying response times. A local application may not be able to function correctly due to the unavailability of a web server that provides some of the required information. However, the required information may be duplicated over multiple web servers under different contexts and possibly different formats as well. For example the current price of a specific stock can be available on multiple websites related to different organizations. If the local application uses one website to get the information and that site fails, it will not be possible to switch to a different site to get that same piece of information. However, it will be desirable to make the application capable of performing that switch when necessary. Yet, it is very difficult to utilize multiple web servers and implement a fault tolerance mechanism among them to provide the required information in real-time bases even with some faults.
- **Efficiency:** some Web information (especially those available on a single web page) may be required by multiple local applications at the same time. For example, all stock information in a single stock market is displayed in real-time on a single web page. Several applications may be requesting different stock prices from that same list at the same time. If the extraction is done within the application, each one will download and process the same page, while it may be much more efficient to download the page once, process it to extract multiple pieces of information then provide each application with its own required information. However, it is a challenging task to determine the duplication in requests among several independent applications and efficiently reduce the amount of processing required to extract the needed information for each one of the applications.

There are several possible solutions for the challenges mentioned above. These solutions can be implemented as part of the local applications that need to use the Web information. However, this approach is inefficient and needs huge development and testing efforts and a lot of time. This effort may be duplicated for different applications that need to reuse Web information. It will be more efficient to have some well developed and appropriately tested independent services that can be efficiently used to obtain the required information by any application. Thus middleware offers a suitable platform for this type of services. This way, it will be easier to

reuse and integrate the required services with different applications.

V. MIDWIRE SERVICES

In this section we discuss our middleware solution to solve most of the challenges mentioned in Section IV. The middleware connects the Web as an information source with the local applications that need to use the information (see Figure 1). This middleware provides some services that can be used by users to configure the required information needed by their local applications. The configuration defines the location of the required information. As it supports information reuse we call it MidWire (Middleware for Web Information Reuse).

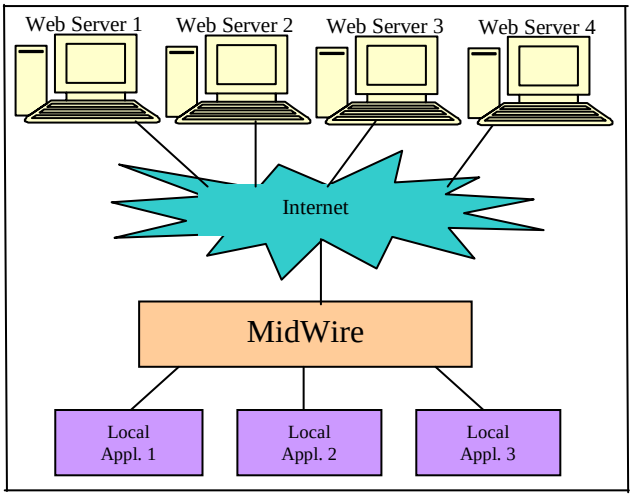

Figure 1. Three local applications reuse Web information available in four web servers through a middleware located locally.

The design of MidWire provides the following functions:

1. Establish connections with web servers and web services.
2. Download and extract the required information from HTML and XML documents.
3. Present all the required Web information in a uniform way such that it can be easily reused by or integrated with the local applications.
4. Capture changes in highly dynamic Web information.
5. Provide a fault tolerance mechanism by utilizing the duplications of the required information over the Web.
6. Provide APIs to allow the local applications developers to easily use the middleware services.

MidWire provides the required Web information for local applications in three delivery techniques:

1. **Polling**: in this technique, the required information will only be downloaded and provided when a local application requests it. The local applications are provided with APIs to make the requests. Responses for these types of requests usually take few seconds since the middleware will need to connect to Web servers, download and process the web page and deliver the requested information with each request.
2. **Caching**: in this technique the middleware frequently downloads and extract the required information and keeps them in a local cache. The information cached will be based on the history of requests made by local applications. The cache will contain the latest downloaded information that may be soon needed by the local applications. Local applications can directly read the required information from the cache using the available APIs. This type of read will not take much time from the local applications to get recent information.
3. **Notification**: in this technique, the local applications can ask the middleware to send notifications to them when a certain value over the Web has changed. The middleware will monitor that value and will only send the notification when the value has changed from the time the request was made. For example, an application is interested to be notified as soon as the current Google Stock price changed. In this case, the middleware will monitor the Google stock current price from one of the web pages or one of the web services providing this information and will only notify the application when the price changes. This type of communication request is useful for applications that do not need a frequent access to the Web information. It transfers the overhead of frequent Web accesses from the application to the middleware.

The suitability of the above communication techniques depends on the application scenarios. Table 2 lists some of the common application scenarios and their most suitable communication techniques.

MidWire addresses several of the challenges discussed in Section IV. Interoperability is addressed by providing a middleware framework that may be implemented in several ways such as using Java modules which can operate across different platforms. In addition, the design of MidWire is flexible enough to allow for the incorporation of different components. It is capable of handling highly dynamic and changing HTML content. MidWire also allows for incorporating multiple sources and servers to be used.

TABLE 2.
DELIVERY SCENARIOS

Application Scenario	Communication Method
Single application accesses different Web information from time to time.	Polling
Single or multiple applications need a continuous stream of information from one or a few web sources.	Caching
Multiple applications need the same information from the Web.	Caching
Single or multiple applications need to capture changes in some Web information.	Notification

VI. MIDWIRE ARCHITECTURE AND IMPLEMENTATION

MidWire is designed and implemented in three layers. The layers are: The Web Information Retrieval Layer, The Cache Layer, and the Delivery Layer (see Figure 2). In this section, the functions of each layer are discussed.

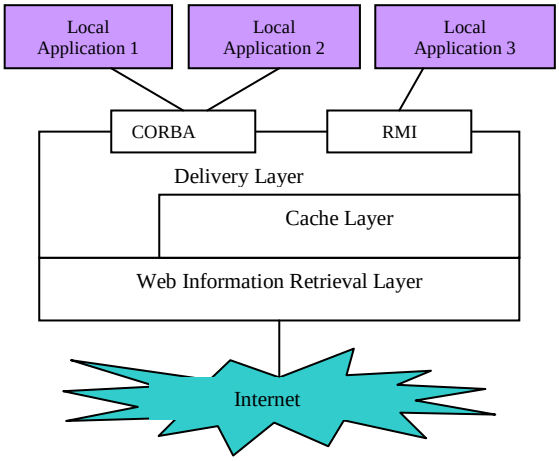

Figure 2. The Middleware Architecture.

A. Web Information Retrieval Layer

One of the main layers of this middleware is the Web Information Retrieval Layer. This layer directly deals with the web servers. The main function of this layer is to retrieve the required Web information. The required Web information can be obtained from the Web using web services or using any library that provides mechanisms to obtain the required information from dynamic HTML pages as discussed in Section III. The information is obtained from the Web in a series of individual requests to be used for serving the polling type deliveries which we discussed in the previous section or as streams to update the cache layer. This layer uses multiple threads parallelize the retrieval of highly dynamic information from the Web. This layer retrieves the required Web information based on user configurations.

One of the advanced functions that can be provided in this layer is to discover changes in the structure of the defined dynamic HTML documents. This layer can implement an automatic validation mechanism to allow the middleware to make sure that the formats of the defined HTML documents were not changed before attempting to extract the required Web information. One possible solution for this problem is to automatically capture and store the format patterns of the HTML documents. These patterns can be used by the system to discover any future changes in the downloaded documents. In case there are changes, the system notifies the middleware administrator to configure new parameters for the middleware.

Another function that can be provided by this layer is to enhance the performance and reliability of the retrieving process. If a piece of the required information is provided by multiple sites or web services, this layer can discover which site or web service can provide faster access. This can be discovered automatically by the layer using some experimental testing. MidWire can test and record response times of available sources periodically to select the best one to provide the required information. In addition, this layer can switch from a faulty source or unreliable source to a working source without affecting the user applications. This function requires that the middleware administrator defines all websites or web services that provide the same information.

B. Cache Layer

The main function of this layer is to provide memory for updated information obtained from the Web. This layer will be accessed by the Web Information Retrieval Layer to update the cache with new information and by the Delivery Layer to obtain updates on required Web information.

One of the advanced functions that can be implemented by this layer is to capture the access patterns of the information in the cache by the applications. Capturing the access patterns can be used to adjust the required speed of retrieving the required information from the Web by the Web Information Retrieval Layer. For example, the applications access a certain value in the cache once every 20 seconds. In this case there is no sense in retrieving that value every 5 seconds. Therefore, the retrieval layer can be informed about that fact to adjust its download accordantly.

C. Delivery Layer

This layer will be accessed by the application to receive Web information either from the cache or from the Web directly using the Web Information Retrieval Layer. This layer will also wrap the required Web information to a format that can be accessed by the applications. For example, this layer can provide different access methods such as RMI, CORBA, web services, and DCOM. These access methods can be either implemented by the users or using tools to help them in automatically generating servers that use both the cache layer and the Web Information Retrieval Layer to get the required information for the applications. The implementation of code generation techniques can be similar to [32].

This layer can combine information collected from different web pages and web services to be delivered as a reply for an application request. The advantage of this function is that instead of making the applications deal with multiple web services and web pages to collect a set of needed information, this layer can provide all required information in one record and reply. This layer also can implement the notification services mentioned in the previous section. This layer notifies the interested applications about any changes in required values.

VII. PROTOTYPE IMPLEMENTATION

We have implemented a prototype of MidWire. In this prototype we did not cover all the features that MidWire can provide; however, we covered some essential features. We used Java to implement the prototype, yet MidWire can be implemented using any other language. The middleware admin can build a configuration file using a set of Java classes, which define the operations of MidWire. We use Java classes for configuration since it

provides flexibility in dealing with different methods for retrieving Web information. This method is used for defining some configurations in other middleware systems such as JOR [33], which is a middleware for Java object content-base routing among distributed Java applications.

In the admin configuration file, each Java class represents a data structure that will be retrieved from the Web. Each class can be defined for a set of related pieces of information from a single page or from a single web service. For example, a class can be defined for US stock quote pages from Yahoo. The class will have a number of fields that define the required structure of a stock in one of the US markets. It can have for example, *StockCode*, *LastTrade*, *Bid*, *Ask*, and *Volume*. One example of this class is *US_Stock* class shown in Figure 3. The class also has a constructor to define the initial setup process for the download. Objects from these classes will be used internally within the MidWire components. These objects represent updated related Web information called Web Information Object (WIO). Each class must have a method called *update()*. This update method is to retrieve the updated information for the WIO. The *update()* method should contain some code to update the defined fields in the class. Within this class the user can use urlINFO or any web service to retrieve related web information. The parameter in the constructor is used to pass information about the retrieve request. It can be for example "WMT". This is to retrieve quote for Wal-Mart Stores. WIOs are then transferred from the Web Information Retrieval Layer to the cache layer or to the Delivery Layer.

The configuration file also has a class called *setup*, which extends the class *MidWireConfiguration*. This class is to define which of the WIOs should be cached and which of the cached WIOs should be monitored for notification services. The definition is done in a static method called *MidWireSetup*. *MidWireConfiguration* contains some methods that implement and provide support for both the cache and notification services. For each cached information, WIO and download interval are defined as *cache(WIO, interval)*. The *interval* defines the frequency of updates in seconds. If the *interval* is 2, then MidWire will download a new update every 2 seconds. Examples of these cached WIOs for Wal-Mart Stores and Intel Cooperation stock information are shown in Figure 3.

In addition, in the configuration file the user can define which of the cached WIOs is to be monitored for providing notification services. The middleware admin can use the method *notify(WIO, field)* where *WIO* is the cached web information object while *field* is the attribute to be monitored for change. If it changes, then this method will notify the interested applications about the changes.

Before we go to the Delivery Layer, let us discuss the implementation of the Web Information Retrieval Layer and the Cache Layer. The Web Information Retrieval Layer was implemented as a set of Java classes. One of the main functions of this layer is to update the Cache Layer with new updated WIOs. In this layer, a thread is started for each WIO to download updated Web information at each defined interval. With each cached object, a timestamp is recorded for the thread start time. Updated WIOs will be sent to the cache layer for storage. The Web Information Retrieval Layer provides a static method to retrieve an update for any WIO. This method can be used by the Delivery Layer to retrieve updated WIO directly from the Web. The Web Information Retrieval Layer uses the Java virtual machine class loader and the reflection API [34] to deal with and invoke the defined methods in the user configuration file.

```
import java.io.*;
class setup extends MidWireConfiguration
{
  public static void MidWireSetup()
  { // Cache Wal-Mart Stock Information
    US_Stock wmt = new US_Stock("WMT");
    cache(wmt,2);
    // Cache Intel Stock Information
    US_Stock intc = new US_Stock("INTC");
    cache(intc,2);
    // Notify when Wal-Mart LastTrade changed
    notify(wmt,"LastTrade");
  }
}

class US_Stock
{
  public String StockCode;
  public String LastTrade;
  public String Bid;
  public String Ask;
  public String Volume;
  private urlINFO stock;
  public US_Stock(String dPar)
  {   StockCode = dPar;
      Stock=new
urlINFO("HTTP://finance.yahoo.com/q?s="+dPar+"&ql=1");
  }

  public void update()
  {   stock.download();
      LastTrade = stock.get("Last Trade:");
      Bid = stock.get("Bid:");
      Ask = stock.get("Ask:");
      Volume = stock.get("Volume:");
  }
}
```

Figure 3. MidWire Configuration.

The fault tolerance in retrieving Web information is handled by the Web Information Retrieval Layer by allowing the middleware admin to define two update methods, *update1()* and *update2()* in the Web information class. Both update methods will retrieve the same required information but from different websites or web services. Invoking any of the two update methods can retrieve the recent Web information and update the attribute of the WIO. The Web Information Retrieval Layer can rely on one of the update methods for obtaining updated information and switch to the other if the first one fails.

The Cache Layer is also implemented as a set of Java classes. It has a data structure to store updated WIOs. Only recent WIOs sent by the Web Information Retrieval Layer are considered. WIOs timestamps are used to

implement that mechanism. The cache layer provides a static method that allows retrieving any cached WIO. This method can be used by the Delivery Layer to retrieve updated WIO from the cache.

The Delivery Layer is responsible for delivering updated Web information to different applications. This layer can be implemented as an RMI server, CORBA server, or any other type of servers. This layer uses the static methods provided by both the Web Information Retrieval Layer and the Cache Layer to obtain updated WIOs. The layer maps WIOs to different formats and communication types used by the applications. We coded this layer manually. However, it is possible to automate the coding by developing tools to map WIOs to RMI server or CORBA server as examples. This tool can implement RMI server by considering both RMI server interface class and one or more WIOs. If common field names are used, then the RMI server code can be automatically generated.

The notification service is provided by the notification manager which is one of the components of the Delivery Layer. Any application interested in monitoring some Web Information should subscribe by sending a subscription message to the notification manager. The notification manager maintains a table for the information about all subscriptions. Whenever, the Cache Layer receives a WIO that is monitored for notification and it has some changes, it will notify the notification manager about that change. The notification manger checks its subscription table to find subscribers with matching change definitions and send the notifications to all applications registered to receive these notifications.

VIII. The Performance Measurements

A number of experiments were conducted on the implemented prototype of MidWire to measure the performance gain and overhead. In the following subsections we provide details of these experiments. First, we measure the overhead introduced by using MidWire when a single value is requested using the polling method. Then we evaluate the performance gain of using MidWire when multiple client applications are requesting a stream of related dynamic values from the Web. In this experiment, we use the caching option. We also evaluate the performance gain of a client application using MidWire compared to using direct Web retrieval methods implemented and embedded in the client applications themselves. In this experiment, we use the notification option. To evaluate MidWire, four multi-core machines were used with the specifications listed in Table 3. These machines were connected by a dedicated LAN using Dell 2324 Fast/Gigabit Ethernet switch. In addition, the machines have wireless network interface cards.

A. Measuring the Overhead

An experiment was conducted to measure the overhead of using MidWire to retrieve some web information using the polling option. The retrieve process was repeated multiple times to retrieve some information though two methods. The first method uses a direct connection between the Web and a prototype client application that implements some code to directly retrieve the required information. This was done using machine M2 through a wireless connection for the prototype client application. The second method is uses MidWire to retrieve the required information. In this method MidWire is connected to the Web and the prototype client application is connected to MidWire. This was done using machine M5 for MidWire and machine M2 for the prototype client applications. Machine M5 was connected to the Internet through wireless connection while both machines M2 and M5 where connected through the wired switched network. The wireless connection of the client machine M2 was disabled. In each case, the retrieve process is repeated 100 times and the average response time is taken. The response time is measured from the prototype client application. The experiment was conducted for some information obtained from four different web pages and the results are shown in Figure 4. On average around 22 milliseconds is added as overhead by using MidWire. This is a very small overhead and represents a very small percentage of the average time needed to download a webpage. In addition, using MidWire relieved the client from the burden of implementing the method to get the information and it also allows the client to reuse the method through MidWire in any other application.

TABLE 3. MACHINE SPECIFICATIONS.

Machine	Specifications
M1	Desktop, Microsoft Windows XP Professional, Intel® Core 2 CPU 6400 @ 2.13GHz, 3.00 GB of Ram, Gigabit Ethernet NIC
M2	Laptop, Microsoft Windows XP Professional, Intel® Core 2 Duo T7300 @ 2.00GHz, 0.99 GB of RAM, Gigabit Ethernet NIC
M3	Laptop, Microsoft Windows 7, Intel® Core i5 CPU M430 @ 2.37 GHz, 4.00 GB of RAM, Fast Ethernet NIC
M4	Laptop, Microsoft Windows 7, Intel® Core i5 CPU M450 @ 2.4 GHz, 4.00 GB of RAM, Fast Ethernet NIC
M5	Laptop, Microsoft Windows 7, Intel® Core i7CPU Q740 @ 1.73GHz, 6.00 GB of RAM, Fast Ethernet NIC

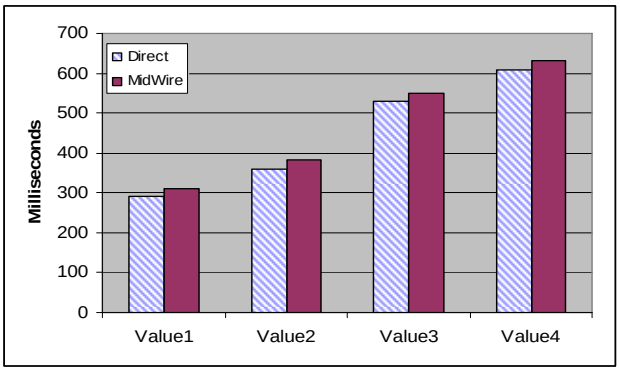

Figure 4. Overhead of MidWire.

B. Performance Gain Using the Caching Option

The caching option is useful when there are multiple client applications that need the same or related

information or information located on the same webpage. To evaluate this option, we used two experiments with multiple applications that directly retrieve related information and then retrieve them through MidWire. All application computers are connected through a limited bandwidth ISDN Internet connection, 1 Mbps through a wireless switch/router device. The experiments were conducted using 2, 4, 6, and 8 client applications in both configurations: direct retrieval and using MidWire. MidWire was installed on M5 while clients where installed on machines M1, M2, M3, and M4. The machines were used for a maximum of two client applications. Only the wired network was used to connect the five machines together. In addition, machine M5 was also connected to the Internet via the wireless network.

The results for both configurations are shown in Figure 5. The time measured for MidWire includes the one-time page download time to MidWire which then uses that page for all applications. The response time significantly increases with the increase of the number of requesting applications in the direct retrieval since the page will be downloaded several times. While the response time only slightly increases with the increasing number of client applications using MidWire. The high increase in the direct retrieval is mainly due to the contention on the limited Internet bandwidth as well as the web server. This is due to multiple and frequent downloads for the same web pages. However, using MidWire, a more efficient download process for the required pages is used. In this case, the same and related information is retrieved once and used by multiple client applications.

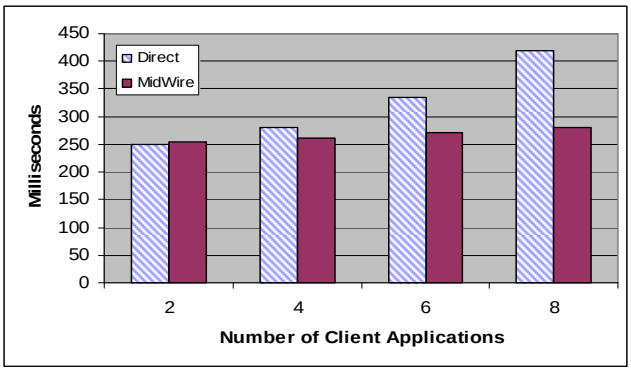

Figure 5. Performance of Caching in MidWire.

C. Performance Gain Using the Notification Option

The notification option in MidWire relives client applications from intensive information download and retrieval processing. MidWire can be configured to notify interested client applications about any change in a value whenever it happens. An experiment was conducted using four multi-core computers to demonstrate that the notification option can provide a good solution for monitoring Web information with efficiently. The four machines M1, M2, M3, and M4 were used to concurrently execute two applications. The first application was parallel matrix multiplication, which fully utilizes the computation powers of the four computers as well the network among them to produce the result. The parallel application was implemented using JOPI and distributed agents supporting runtime environment [28]. The second application replicated on all computers is a prototype of an application that needs to retrieve some value changes from the Web. Two configuration cases were attempted. The first was by implementing the retrieve process as part of the prototype application. In this case the retrieve part will continuously monitor the required web values. In the second configuration, MidWire was used to notify the prototype client application about any value changes. MidWire was installed on machine M5 with a wireless connection to the Internet. This experiment was designed to evaluate the impact of consuming the system recourses available on the four computers. As both applications execute on the machines, we notice that the parallel matrix multiplication execution time is 22 seconds when using the direct retrieval and it only reaches 18.5 seconds when MidWire is used. The main reason for this difference is that the direct retrieval consumes more resources on the machines thus affecting the other application using these resources. MidWire, on the other hand, does not require that much resources since it will separately retrieve and sort out the required results for all instances of the application. As a result the load on the machines is less.

D. Discussion

Based on the results provided by this section, although MidWire adds some overhead for some scenarios for retrieving Web information, it provides efficient solutions for retrieving multiple and stream Web information for multiple client applications. The overhead is relatively low and is compensated for in the savings gained using the caching and notification methods. These solutions can be provided using the caching and notification communication options allow for more efficient downloading of Web pages and better handling of the required data. Furthermore, MidWire reduces the overall load on the client applications resources and the network infrastructure in use. In addition to the performance advantages, MidWire saves significant efforts and time in designing, developing and testing, information retrieval modules as part of different client applications.

Using MidWire, new client applications can be easily linked to utilize the available web information. One example of these applications is the Personalized Stock Investors Alert System [1]. This system provides personalized monitoring for stock information over the Web. The user defines a set of conditions to get notifications about specific stock information. This helps stock investors to define their business rules for stock buying and selling to get notifications about the occurrence of the defined events. Currently, this application is implemented using direct connection to the Web. However, MidWire can be used to relive the stock alert application from retrieving and processing the required web information.

IX. CONCLUSION

In this paper we discussed the design issues of middleware services to help retrieve, integrate and reuse dynamic web-based information from the Web with local applications. To do that, we went through the different methods used to access Web information and how middleware solutions may be useful to enable and optimize these methods. We discussed the different challenges to be addressed when considering the middleware design. Some of these challenges are interoperability, HTML format changes, distributed information, highly dynamic information, fault tolerance, efficiency and software engineering issues. Then we discussed the design of the proposed middleware which provides three delivery techniques: polling, caching and notification. Finally we described the architecture of the proposed middleware which comprise of three layers: the Web Information Retrieval Layer, the Cache Layer and the Delivery Layer. This paper provided the design aspects of the middleware with a prototype implementation. Some experiments were conducted to demonstrate the advantages of using MidWire.

One drawback of the current design of MidWire is that it works on a single machine. This makes MidWire a single-point-of-failure. In addition, it is not scalable for very high load. As future work, we plan to investigate the use of multiple machines for MidWire. This will make MidWire more reliable and scalable. In addition, we plan to investigate models and algorithms for validating the formats of the defined HTML documents and detecting changes in them before attempting to extract the required Web information. Furthermore, we plan to develop some techniques to enhance the retrieval process. We plan to investigate adding some advanced features to MidWire such as the dynamic discovery of existing sources, their ranking, as well as using web information modeling to capture the dynamic changes and validate the retrieval formally.

ACKNOWLEDGEMENTS

This work was partially supported by a UAEU research grants #01-04-9-11/09 and #01-03-9-11/08. A primary version of this paper was presented at NBiS 2009.

REFERENCES

[1] J. Al-Jaroodi and N. Mohamed, “A Personalized Stock Investors Alert System,” in The Journal of Software, Finland, Vol. 4, No. 8, pp. 875-882, October 2009.

[2] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, “A Generic Notification System for Internet Information,” In Proc. IEEE International Conference on Information Reuse and Integration (IRI-08), Las Vegas, USA, pp. 166-171, 2008.

[3] Web Service: CurrencyConvertor, http://www.webservicex.net/ws/wsdetails.aspx?wsid=10, viewed May 23, 2011.

[4] N. Mohamed and J. Al-Jaroodi, “A Simple and Efficient Approach for Retrieving Live HTML-based Internet Information,” in System and Information Sciences Notes, Vol. 1, No. 3, pp. 221-224, July 2007.

[5] F. Curbera et al. “Unraveling the Web Services: an Introduction to SOAP, WSDL, and UDDI,” in IEEE Internet Computing, Mar/Apr 2002.

[6] Extensible Markup Language (XML) 1.0 (Fourth Edition), http://www.w3.org/TR/REC-xml/, September 2006.

[7] RFC 1866, Hypertext Markup Language – 2.0, November 1995.

[8] R. Burget, “Hierarchies in HTML Documents: Linking Text to Concepts,” in Proc. 15th International Workshop on Database and Expert Systems Applications (DEXA'04), pp. 186-190, 2004.

[9] S. Li, Z. Peng, and M. Liu, “Extraction and Integration Information in HTML Tables,” in Proc. 4th International Conference on Computer and Information Technology (CIT'04), pp. 315-320, 2004.

[10] H. Li, “Web Information Retrieval - Through Introduction to Semantic Matching Project,” presentation, MSRA WSM Lecture Series, Feb. 25, 2011.

[11] J. Xu and H. Li, “A Boosting Algorithm for Information Retrieval,” in proc. 30th Annual International ACM SIGIR Conference On Research And Development in Information Retrieval, ACM, USA, 2007.

[12] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma, “Terrier: A High Performance and Scalable Information Retrieval Platform,” in Proc. SIGIR Open Source Workshop, Seattle, Washington, USA, 2006.

[13] W.W. Cohen and W. Fan, “Learning Page-independent Heuristics for Extracting Data from Web Pages,” in Computer Networks, Vol. 31, No. 11-16, pp. 1641-1652, May 1999.

[14] A. Jatowt and M. Ishizuka, “Temporal Multi-page Summarization,” in the Journal of Web Intelligence and Agent Systems, IOS Press, Vol. 4, No. 2, pp. 163-180, 2006.

[15] E. P. LIM, W. K. Ng, S. S. Bhowmick, F. Qin, and X. Ye, “A Data Warehousing System for Web Information,” in Proc. 1st Asia Digital Library Workshop, Research Collection School of Information systems, Paper 1025, 1998.

[16] N. Glance, J. L. Meunier, P. Bernard, and D. Arregui, “Collaborative Document Monitoring,” in Proc. International ACM SIGGROUP Conference on Supporting Group Work, ACM, NY, USA, 2001.

[17] E. Shakshuki, H. Ghenniwa, and M. Kamel, “A Multi-agent System Architecture for Information Gathering,” in Proc. 11th International Workshop on Database and Expert Systems Applications, London, UK, pp. 732-736, 2000.

[18] S. Rahimi and N. Carver, “A Multi-Agent Architecture for Distributed Domain-Specific Information Integration,” in Proc. 38th Annual Hawaii International Conference on System Sciences, 2005.

[19] M. S. Ackerman, B. Starr, and M. Pazzani, “The Do-I-Care Agent: Effective Social Discovery and Filtering on the Web,” in proc. 5th International Conference Computer-Assisted Information Retrieval (RIAO 1997), Montreal, Canada, 1997.

[20] M. Labský, V. Svátek, O. Šváb, P. Praks, M. Krátký, and V. Snášel, “Information Extraction from HTML Product Catalogues: From Source Code and Images to RDF,” in Proc. IEEE/WIC/ACM International Conference on Web Intelligence (WI'05), pp. 401-404, 2005.

[21] Web Scraping, Wikipedia, http://en.wikipedia.org/wiki/Data_scraping, viewed October 2011.

[22] ScraperWiki, https://scraperwiki.com/, viewed October 2011.

[23] M. Beno, J. Misek, and F. Zavoral, "AgentMat: Framework for Data Scraping and Semantization," in Proc. 3rd International Conference on Research Challenges in Information Science (RCIS), pp. 225-236, April 2009.

[24] A. Pan, J. Raposo, M. Alvarez, J. Hidalgo, and A. Fina, "Semi-automatic Wrapper Generation for Commercial Web Sources," book chapter in Engineering information Systems in the Internet Context, Editors: C. Rolland, S. Brinkkemper and M. Saeki, 2002.

[25] Mozenda, http://www.mozenda.com/data-scraping-software, viewed October 2011.

[26] A. Y. Levy, "The Information Manifold Approach to Data Integration," in IEEE Intelligent Systems, Vol. 13, No. 12-16, 1998.

[27] C. Britton, "IT Architecture and Middleware: Strategies for Building Large, Integrated Systems for Building Large, Integrated Systems," Addison-Wesley, 2001.

[28] J. Al-Jaroodi, N. Mohamed, H. Jiang, and D. Swanson, "Middleware Infrastructure for Parallel and Distributed Programming Models on Heterogeneous Systems," in IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 11, pp. 1100-1111, November 2003.

[29] S. Hadim, N. Mohamed, "Middleware Challenges and Approaches for Wireless Sensor Networks," in IEEE Distributed Systems, Vol. 7, No. 3, March 2006.

[30] S. Hadim, J. Al-Jaroodi, N. Mohamed, "Trends in Middleware for Mobile Ad Hoc Networks," in the Journal of Communications, Vol. 1, No. 4, pp. 11-21, July 2006.

[31] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Middleware for Robotics: A Survey," in Proc. IEEE International Conference on Robotics, Automation and Mechatronics (RAM 2008), Chengduo, China, September 2008.

[32] N. Mohamed, A. Romaithi, E. Samahi, M. D. Kendi, and E. Jabrie, "Generating Web Monitors," in Proc. International Conference on Internet Computing (ICOMP 2008), Las Vegas, USA, pp. 111-116, July 2008.

[33] N. Mohamed, X. Liu, A. Davis, and B. Ramamurthy, "JOR: A Content-Based Object Router," The Journal of Computer Communications, Special Issue on Activated & Programmable Internet: The Converging Technologies for the Internet based Active and Programmable Systems, Elsevier, Vol. 28, No. 6, pp. 654-663, April 2005.

[34] Java API Specification Website: http://java.sun.com/reference/api/index.html, Sun Microsystems, 2011.

Nader Mohamed is an associate professor at The Faculty of Information Technology, United Arab Emirates University, Al-Ain, UAE. He obtained his Ph.D. in Computer Science from University of Nebraska-Lincoln, Nebraska, USA in 2004. He was an assistant professor of Computer Engineering at Stevens Institute of Technology in New Jersey, USA. His current professional interest focuses on middleware, Internet computing, sensor networks, and cluster, Grid, and Cloud computing. He published more than 90 refereed articles in these fields.

Jameela Al-Jaroodi received her Doctorate of Philosophy degree in Computer Science from The University of Nebraska-Lincoln, USA in 2004. Since August 2006, she has been with the Faculty of Information Technology, at The United Arab Emirates University, UAE as an assistant professor. Prior to joining UAEU, Dr. Al-Jaroodi was a research assistant professor at Stevens Institute of Technology in New Jersey, USA. Currently, her research interests involve middleware, distributed collaborative systems, security, and mobile and pervasive computing. While at Stevens, Dr. Al-Jaroodi received the Research Excellence Grant from Sun Microsystems, Inc. In addition, several areas of her research were also supported by the United States National Science Foundation (NSF), Nebraska Foundation, the National Center for Information Technology in Education (NCITE) and UAEU research grants.