e-speak

WebAccess Programmer's Guide

Developer Release 3.01
June 2000

HEWLETT® PACKARD

Contents

Chapter 1 WebAccess Introduction

Today's Internet technology falls short in fostering e-businesses and e-services to communicate and cooperate with each other in three areas: support for new business paradigms where e-services securely connect to other e-services, quick deployment/advertisement of new services, and dynamic service discovery and composition. The e-speak architecture addresses each of these challenges. First of all it is designed to let e-services and smart devices communicate and broker services with one another across the Internet. It speaks and understands XML, a de facto Internet standard, and is developed in Java to leverage its platform independence. Furthermore, it is designed with security in mind from ground up. For example it mediates all accesses to services and implements secure firewall traversal. The platform and transport independence along with the strong security allows disparate entities in the Internet to interact with and provide services for each other in a secure and manageable way.

Any e-speak enabled e-service can interact with any other regardless of their physical location, platform, system management policy, development environment, or device capabilities. E-speak provides a flexible resource model for service advertisement and a powerful query mechanism for dynamic service discovery. These abstractions enable e-services in the Internet to be composed and assembled dynamically

E-speak has a layered architecture where each layer provides a different level of essential abstractions for e-services development. At the bottom is the e-speak engine which provides the basic infrastructure services such as service registration/advertisement, discovery, security, and management. Above the bottom sits a number of programming libraries which support network object and document exchange models. WebAccess is the one that supports the document exchange model. On top of the libraries is business logic supported by a variety of XML-based

frameworks such as Microsoft's BizTalk Framework and CommerceNet's eCo. The e-speak service framework is set of protocol and interface specifications that make applications e-speak aware.

This guide describes the functionality of e-speak, a distributed infrastructure for e-services. In particular we focus on the abstractions and the functionality of WebAccess, a Web gateway to e-speak. WebAccess embraces XML over HTTP/TCP to interface with Web-based e-services.

E-speak builds all abstractions and system functionality on one single first-class entity - resource. A resource is a uniform representation of an entity created in or registered with the e-speak engine. E-speak does not distinguish active services such as name servers, printing services, and a car sales service, from passive resources such as files, data entries, and user profile information. Rather, it treats them uniformly by dealing only with their representations (i.e., resources). For their convenience users may distinguish active services from passive resources.

For example, suppose a user creates a file service and registers it with the e-speak engine. The corresponding file resource within the engine is nothing more than a description of the actual file such as name and size, and a specification such as access control policy. The e-speak engine does not access the file directly. Rather, it keeps a mailbox for the resource and routes and deposits requests to the file service into the mailbox. The file service defines a handler that listens to the mailbox for incoming requests and accesses the file upon request.

Service providers register a service with an e-speak engine by submitting service specification and descriptions. Description is about how the service is presented to users while specification specifies access information such as interfaces and access control policy. The dichotomy of the resource representation allows for a flexible yet secure service discovery framework. In response to a registration request, the engine creates a resource that abstracts the service. The descriptions may be advertised to other e-speak engines. e-speak allows service providers to unregister their services; however, e-speak does not attempt to maintain consistency among multiple engines.

Real life entities have different descriptions depending on their role, functionality, or environment. A fax-phone may have two descriptions, one for fax machine and the other for phone. An online bookstore is a seller to book lovers and a distributor to book publishers. To accommodate the requirement e-speak allows services to have multiple descriptions. A description consists of attribute name-value pairs.

Services dynamically enter and leave an e-speak community. Thus e-speak recommends users to discover a service before accessing it. An immediate implication is that every lookup request may return different results. e-speak provides a powerful querying mechanism for users to find most appropriate services. A query is specified over descriptions of a service.

The attribute-based description and lookup leads to a potential name collision problem. Suppose a 21-inch monitor as well as a 21-inch TV is registered. When a user tries to find a TV using a query "size=21" she may unexpectedly get the 21-inch monitor as well. To avoid name collision and to facilitate description and query validation, e-speak introduces a powerful abstraction called vocabulary and requires descriptions be specified in a specific vocabulary. A vocabulary defines a name space, i.e., a set of valid attributes and their types in the vocabulary. Attributes in a query are qualified with vocabulary references to resolve the name ambiguity. The engine validates descriptions and query constraints against vocabularies. Note that the name space is similar to a XML name space defined by a XML schema. On the other hand, a vocabulary defines a set of descriptions in the vocabulary as a type in programming languages defines a set of values in the type. Thus vocabularies naturally partition the search space of descriptions. This allows vocabularies to evolve over time independently of other vocabularies.

A vocabulary is a resource managed by the e-speak engine. Since it is a resource anybody can create and register a vocabulary. A vocabulary itself is described in other vocabularies. To end the recursion e-speak defines the base vocabulary with the Type and Name attributes which is unique across all e-speak engines. Since anyone can create vocabularies, two identical vocabularies may exist in an e-speak engine. The vocabulary conflict problem may be resolved by vocabulary unification. However, we envision that standard bodies such as RosettaNet, CommerceNet, and Microsoft BizTalk Framework, will create a few globally adopted market-specific vocabularies.

E-speak supports coarse-grain as well as fine-grain security mechanisms. First a user may share her services with specific users by creating her own vocabulary and making it visible only to the users. Since vocabularies partition the search space of descriptions, those who are not aware of the vocabulary may not find the services. If requested by a client, e-speak virtualizes names that identify services. With name virtualization neither service providers nor clients need to reveal their identity to interact with each other. The engine keeps mapping information which relates virtual names to actual services. Combined with dynamic discovery, the name virtualization may provide for dynamic fail-over, run-time upgrades, and service relocation without disrupting clients. Furthermore, it may be used to implement transparent load balancing with service replication. All requests to services through e-speak engines are mediated. The mediation is enforced using attribute certificates based on Simple Public Key Infrastructure. Different access rights to a service are granted depending on the attributes authenticated. Service access in e-speak is based on asynchronous messaging. Layered above this, e-speak libraries provide user friendly interaction models such as the network object model. The mediated yet uniform access is the design principle that allows the e-speak engine to accommodate any type of resources and service.

In summary, the e-speak engine provides the following infrastructure services:

- **Message routing.** Requests and replies are routed through e-speak engines. Reliable messaging may be supported between clients, if necessary.
- **Advertisement/registration.** Services may be described in multiple vocabularies. The descriptions may be advertised to other e-speak engines. Also services may advertise their descriptions into external advertising depots such as HP E-Services Village via the e-speak advertising service.
- **Dynamic discovery.** E-speak provides a powerful querying mechanism for clients to discover services dynamically from e-speak engines as well as from external advertising depots.
- **Security/mediation.** E-speak implements an attribute certificate based security mechanism using SPKI to mediate all accesses to services. If necessary, name translation is used to hide actual identities of service providers and clients.

- **User account and session management.** E-speak provides a centralized user management facility for storing preferences and application specific data. XML and browser clients must first login as a specific user before performing requests. That login results in a session which is transferable among client hosts.
- **Persistent message queue management.** E-speak provides asynchronous messaging which gives applications a polling mechanism to check for response messages. This provides flexibility for scenarios where e-services may not respond to a request for several days or even weeks and it would be unreasonable to block waiting for that response.

Chapter 2 A Sample Application

This chapter shows how WebAccess e-speak can be used for building WebAccess applications to dynamically access e-speak services in the Internet or how to create a new service that becomes a part of the e-speak community.

The first example for this tutorial is called Book Broker. The Book Broker service is an e-service that provides comparison book shopping. It discovers online bookstores dynamically and composes them to create its own service. The Book Broker example shows how WebAccess e-speak can be used for dynamic composition of existing services in the Internet.

Book Broker Example Overview

In this example, the Book Broker service and three online bookstores, Amazon.com, FatBrain.com, and Barnesandnoble.com, are registered as e-speak services. The Book Broker service and the three online bookstores are described in the broker and online bookstore vocabularies, respectively. Note that the three online bookstores are not e-speak enabled. We created proxy services to wrap them up and make them e-speak aware. The proxy services are registered with the engine and bridge the Book Broker service with XML/HTTP and the online bookstores with HTML/HTTP.

Clients discover the Book Broker service using the broker vocabulary. They are interested in neither how many online bookstores are currently available in the e-speak engine nor how the Book Broker service finds them. If multiple Book Broker services are registered with the engine, clients can choose one that they prefer. Upon request from a client, the Book Broker service discovers all the currently available online bookstores to which it forwards the book search query in parallel. The online bookstores process the query and return the results to the Book

Broker service. The Book Broker service collects all the results and returns a combined list of book offers to the client. Notice that the number and kind of bookstores found can vary over time since online bookstores can connect and disconnect dynamically.

The overall architecture of the example is shown in the following figure:

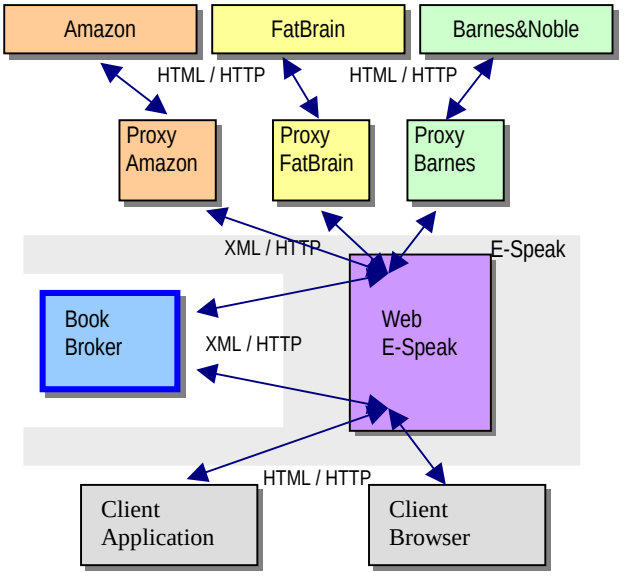


Figure 1 Architecture of the Book Broker service

Book Broker Schema and XML Examples

The Book Broker service uses XML documents to represent queries as well as the resulting book lists. The structure of the documents is defined in a simple Book Broker XML schema:

```
<?xml version="1.0"?>
```

```
<schema targetNameSpace="http://www.e-speak.net/Schema/BookBrokerbookquery.xsd"
    xmlns="http://www.w3c.org/1999/XMLSchema"
    xmlns:offer="http://www.e-speak.net/Schema/BookBroker/bookquery.xsd">
    <element name=”bookquery” type=”offer:queryType”/>
    <complexType name=”queryType”>
      <element name="author" type=”string” minOccurs="0"  maxOccurs=”1”/>
      <element name="title" type=”string” minOccurs="0" maxOccurs=”1”/>
      <element name="subject" type=”string” minOccurs="0" maxOccurs=”1”/>
      <element name="publisher" type=”string” minOccurs="0" maxOccurs=”1”/>
      <element name="isbn" type=”string” minOccurs="0" maxOccurs=”1”/>
    </complexType>
</schema>

<?xml version="1.0"?>
<schema targetNameSpace=
                  "http://www.e-speak.net/Schema/BookBroker/booklist.xsd"
                xmlns="http://www.w3c.org/1999/XMLSchema"
                xmlns:blist=
                 "http://www.e-speak.net/Schema/BookBroker/booklist.xsd">
    <element name=”booklist” type=”blist:bookListType” maxOccurs=”1”/>
    <complexType  name=”bookListType”>
       <element name=”bookoffer” type=”blist:bookOfferType”
                     maxOccurs=”unbound”/>
    </complexType>
    <complexType name=”bookOfferType”>
      <element name="author" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
      <element name="title" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
      <element name="publisher" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
      <element name="year" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
      <element name="offeredby" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
      <element name="price" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
      <element name="shipsin" type=”string”
                     minOccurs=”0”  maxOccurs="1"/>
      <element name="URLtoOrder" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
      <element name="comments" type=”string”
                     minOccurs=”0”  maxOccurs=”1”/>
    </complexType>
</schema>
```

The example query below finds books with the title “The old man and the sea” by E. Hemingway, which can be generated from a browser’s FORM POST request (see Figure 4) and transformed by Web e-speak.

```
<?xml version=”1.0”?>
```

```
<bookquery xmlns=
      "http://www.e-speak.net/Schema/BookBroker/bookquery.xsd">
    <author> Hemingway, Ernest </author>
    <title> The Old Man and the Sea </title>
</bookquery>
```

The following is a resulting book list from the Book Broker service.

```
<?xml version="1.0"?>
<boolist xmlns=
      "http://www.e-speak.net/Schema/BookBroker/booklist.xsd">
  <bookoffer>
      <author> Hemingway, Ernest </author>
      <title> The Old Man and the Sea </title>
      <publisher> Bubble Book Publishing </publisher>
      <year> 1996 </year>
      <offeredby> Amazon </offeredby>
      <price> $16.95 </price>
      <shipsin> 3 days </shipsin>
      <URLtoOrder> http://www.amazon.com/…/ </URLtoOrder>
      <comments> paper back </comments>
  </bookoffer>
  <bookoffer>
      <author> Hemingway, Ernest </author>
      <title> The Old Man and the Sea </title>
      <publisher> Sky Publishing Company </publisher>
      <year> 1999 </year>
      <offeredby> Barnes and Noble </offeredby>
      <price> $59.99 </price>
      <shipsin> 1-2 weeks </shipsin>
      <URLtoOrder> http://www.barnes.com/…/ </URLtoOrder>
      <comments> special edition </comments>
  </bookoffer>
</booklist>
```

The Book Broker service returns the resulting book list to clients in XML. Translation from XML to HTML/WML is done by Web e-speak. A sample response screen is shown in Figure 5. Clicking one of the URLs takes the user to the actual offer at the online bookstore's website (See the `URLtoOrder` element).

Book Broker Service Features

Even though the Book Broker service provides simple functionality, it has some notable features.

- Query results are obtained by accessing the live web sites, guaranteeing up-to-date results.

- All the necessary transformations and translations are handled by Web e-speak. Applications can use their own communication protocol and still interact with others without any friction. Document translation using XSL/XSLT between XML documents with compatible XML schemas will be supported in future releases.
- No prior knowledge about availability and contact location is required to access a service. This information is obtained on the fly with dynamic discovery. New online bookstores are instantly discovered as they join the community. Online bookstores leave the community transparently without affecting the Book Broker service.

Running the Book Broker Service

To run the BookBroker service, the environment as shown in Figure 1has to be set up with e-speak. The following steps need to be carried out by the service provider:

1. Create two Vocabularies:
   - the brokerVocab (with attributes: name, location, goods) for the registration of the BookBroker service,
   - the bookstoreVocab (with attributes: name, goods) for the registration of the Proxy services representing external online bookstores in e-speak;
2. The BookBroker service must be registered with the e-speak core using brokerVocab.
3. The Proxy services must be registered using the bookstoreVocab

In order to invoke the BookBroker service through WebAccess, the service has to be registered with its URL and needs to be available through HTTP. This can be achieved by running the service behind a web server or, simpler, implementing the service as a servlet.

The Bookbroker and proxy services are implemented as servlets and are installed on the same server as WebAccess. There are three proxy services for respective online book sellers in the Internet: ProxyAmazon (for amazon.com), ProxyBarnes (for barnesandnoble.com), and ProxyFatBrain (for fatbrain.com). The installation is described in the WebAccess Installation Guide.

A user can create the two vocabularies and register the Bookbroker and the proxy services through the browser interface provided by WebAccess as described in the next chapter. For convenience, all necessary registrations can be performed by running the BookBroker program from the command line. This feeds all registrations to the core's repository using the HTML interface provided by WebAccess. The BookBroker code has a section in it showing how it works. The effect is the same as having all resources registered manually through the Browser interface.

The following command line runs the registration function:

```
java net.espeak.webaccess.htdocs.servlets.BookBroker
```

It is assumed that the core and the Apache web server are running and needed services have been installed as servlets according to the instructions in the Installation and Configuration Guide: BookBroker, ProxyAmazon, ProxyBarnes, ProxyFatBrain. Also, the above command needs the CLASSPATH set as below:

```
CLASSPATH=.;%installDir%\lib;%installDir%\lib\esclient.jar;%installDir%\lib\escor
e.jar;
   %installDir%\extern\parser\xerces.jar;<JSDK>\jsdk20\lib\jsdk.jar;

%installDir%\extern\webmacro\webmacro.jar;%installDir%\extern\ldap\ldapjdk.jar;
   %installDir%\extern\oracle-lib\classes111.zip
```

After all vocabularies and services have been registered with the e-speak core, the BookBroker service is ready for use. Clients can use the BookBroker service through any of the provided interfaces, such as the Browser interface for human interaction or the XML interface provided for client programs. The Browser interface is presented here.

Follow the steps in the following pages.

1 You need a Login to WebAccess using any of the available user accounts.

Discovery of the brokerVocab in order to find the registered BookBroker afterwards:

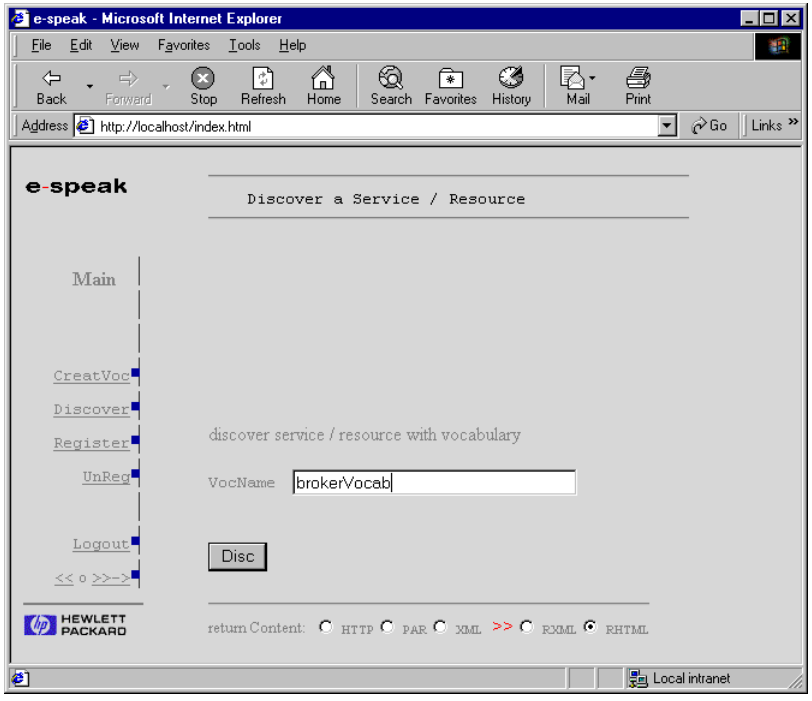


Figure 2 A screen shot of the service discovery

The following two screen shots show the query form used to find a broker service for “books” with the broker vocabulary. The screen below demonstrates how the result query looks when one broker service has been found for books. Since there is the only one service registered, it is the only service shown in the browser.

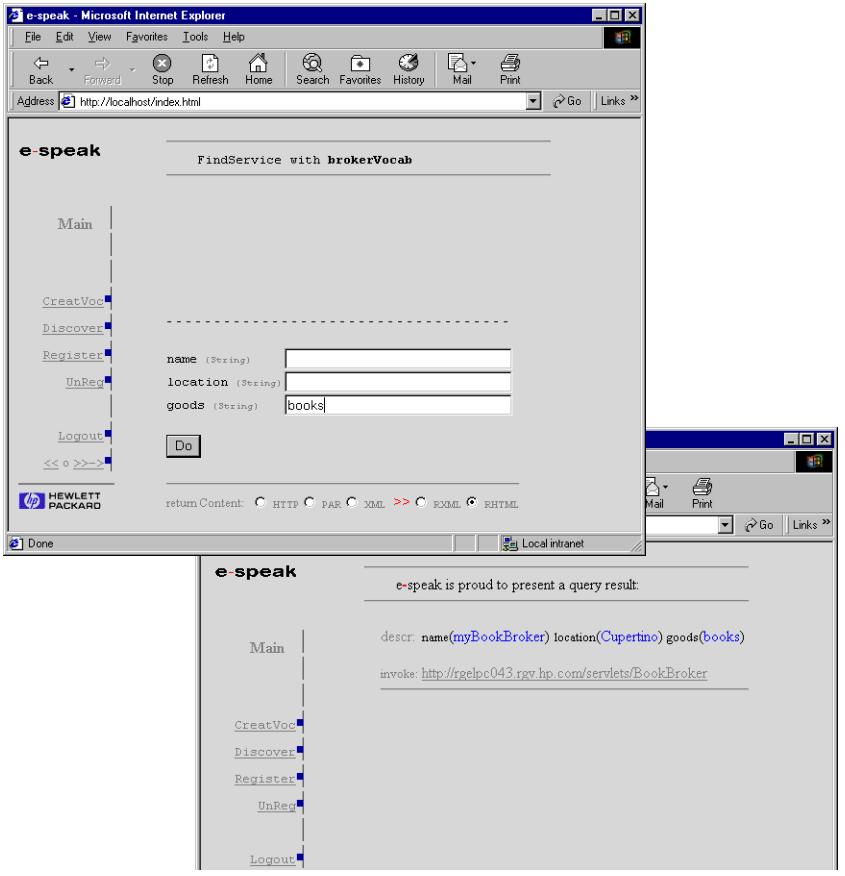


Figure 3 Screen shots of the service discovery with a given vocabulary

After clicking the "invoke" URL shown contained in the query result page, the BookBroker service receives a HTTP message sent by WebAccess's callout module. On receipt of this initial message, the BookBroker service returns its initial screen with the query form for books as shown below.

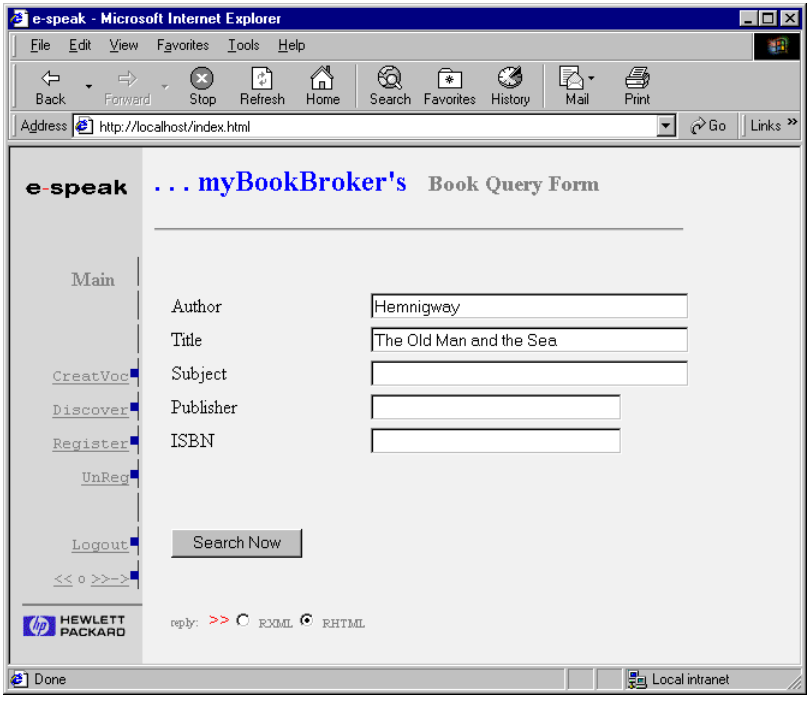


Figure 4 Screen shot of the book search form from the Book Broker

This interaction shows how a service is invoked through WebAccess. After the user has filled in some query information and has clicked on the <SearchNow> button, the BookBroker service processes the query, searches for registered proxy services of online book stores in the core, and sends an XML translated version of the book query to all found proxy services through HTTP. All proxy services process the query by transforming the XML version of the book query into the individual query formats of online bookstores and relaying them out to the respective websites. Returned results are translated back from HTML into XML according to the book-broker schema. These query results are returned to the BookBroker service which merges them into a final list of book offers, translates the list from XML into HTML, and then returns the list to the browser:

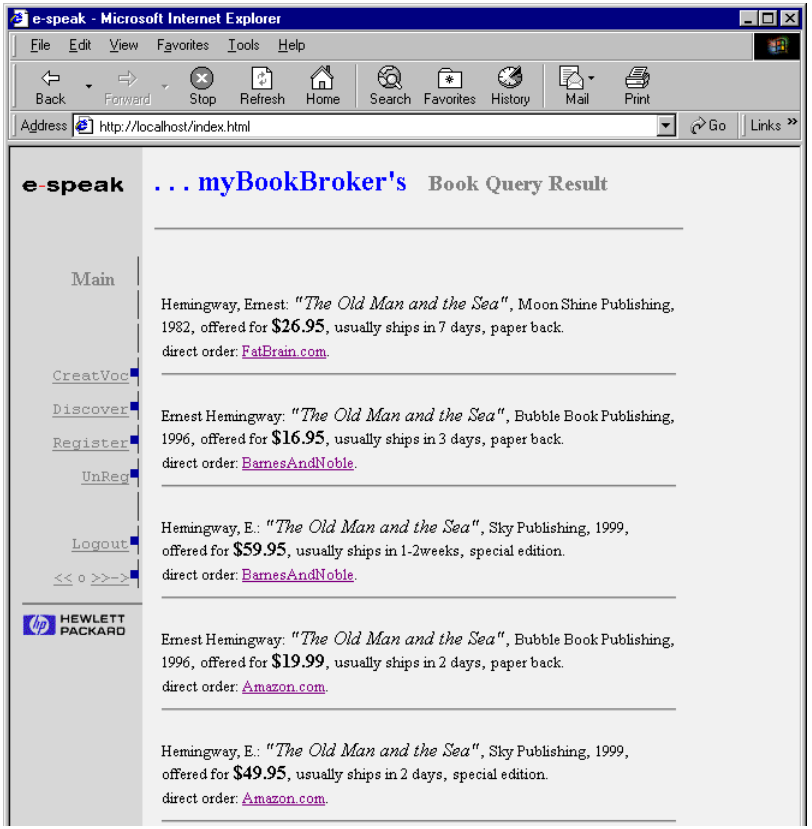


Figure 5 Screen shot of the response from the Book Broker

The next screen shows the result of the book query returned when the RXML field is checked. It returns the internal XML representation of the book list:

Source Code of the Book Broker Service

The source code of the complete BookBroker example is made available with the source distribution of e-speak. For the purpose of a brief overview, the source code of the BookBroker service is shown in Appendix B.

Chapter 3 Browser Interface—HTML

Browser access is provided through Form translation classes that accept a browser's form input and convert it into compliant documents. Form translation is done on the basis of the URL supplied in the GET HTTP request or parameters passed with a HTTP FORM POST.

WebAccess Browser -- Overview

Shown below are a series of WebAccess Browser pages that initially appear when a client is logging in, or conducting account management, related service setup activities and persistent queue management with e-speak.

The browser interface does not assume a XML-enabled browser. WebAccess performs needed XML to HTML transformation automatically.

The browser provides an interactive interface for users to explore the functionality of WebAcces, to provide a visible access to registered services and vocabularies in the e-speak core or to perform simple management tasks such as registering services, vocabularies or new users. However, it does not cover the full feature set of WebAccess or the core.

The browser interface has three main sections reflecting the structure of WebAccess as introduced in chapter 1. These sections are:

- **Main** core's functionality (create vocabularies, register and discover services),
- **ActMgr** -- Account Management (create and maintain user or application accounts),
- **PQMgr** -- Persistent Messaging (send messages, poll for and get messages).

The browser interface for these three sections is explained in detail below

Log in to WebAccess

After successfull installation of WebAccess, the login page appears in a browser pointing to a URL where the web server is running (`localhost/index.html` in the simplest case as shown in the figure below). To log into WebAccess e-speak, click the Login button.

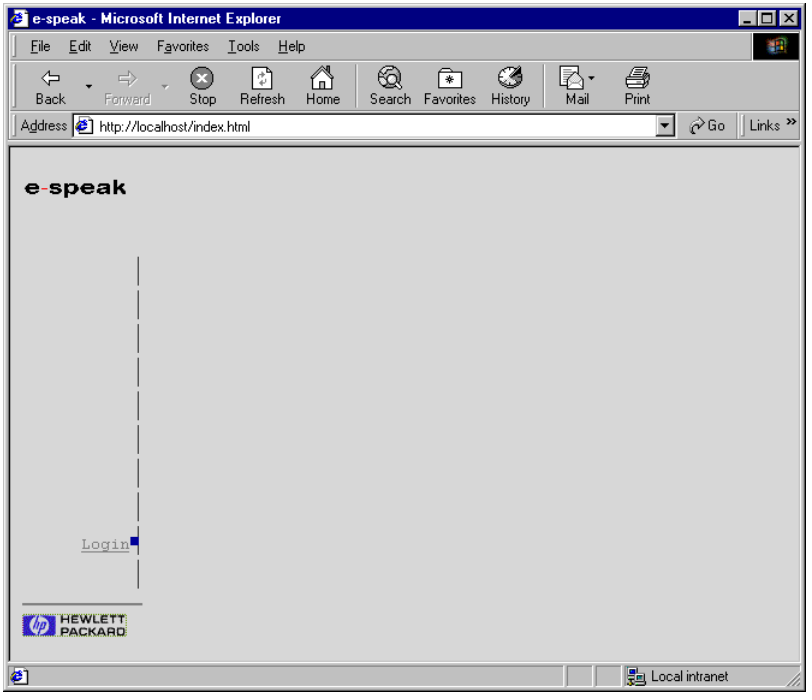


Figure 6 WebAccess Browser Login Screen

The Login user and password screen appears. Type in your user name and password and click the Login button to continue. A user **"esadmin"** with password **"esadmin"** has been pre-registered and can be used for the first login after the installation. The password should be changed after the first login as described in the section about the Account Manager in this chapter.

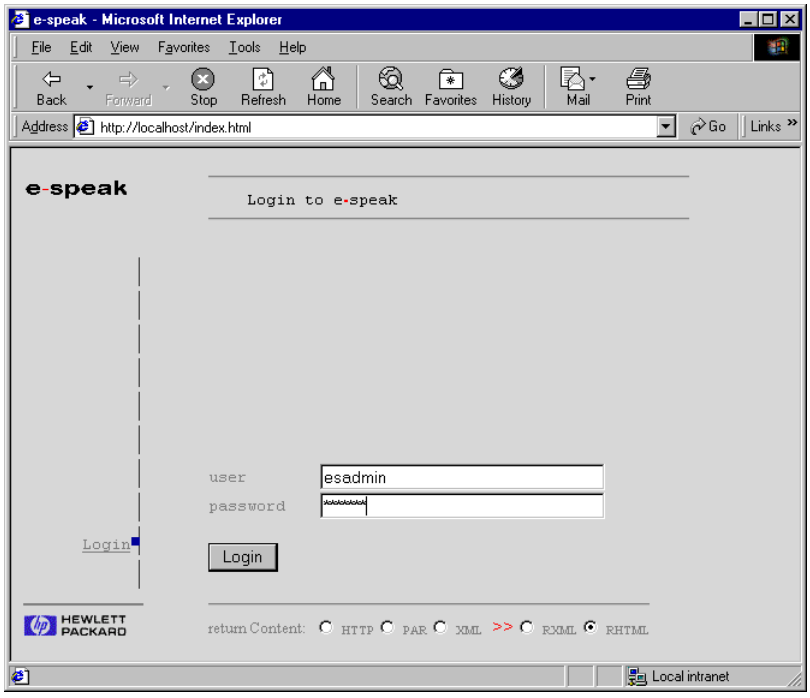


Figure 7 WebAccess Browser Login User and Password Screen.

Main -- WebAccess Browser Main Section

When logging in is completed successfully, the WebAccess Browser Main screen appears. It is shown below. From the main screen you can select from a group of activities, such as creating a vocabulary, registering or unregistering a service or discovering a service.

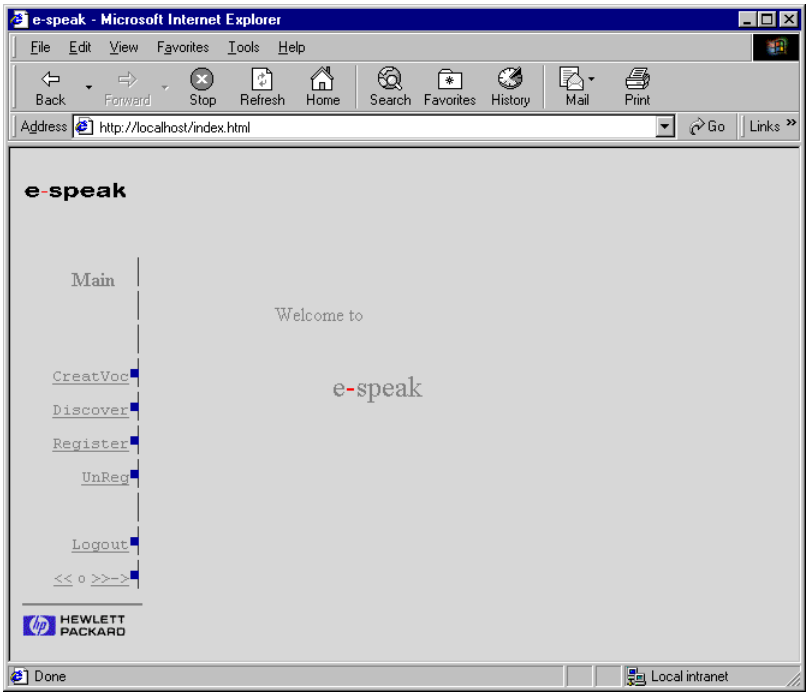


Figure 8 WebAccess Browser Main Screen

Create a Vocabulary

To create a vocabulary, click CreateVoc on the Main menu. The Create a Vocabulary screen appears. Type the vocabulary name in the VocName dialog box and the attributes for that vocabulary in the Attr(:Type) dialog box, following the instructions

on the screen. Select the form of your content return by clicking one of the radio buttons in the return Content section at the bottom of the screen. The completed screen looks like the example shown below.

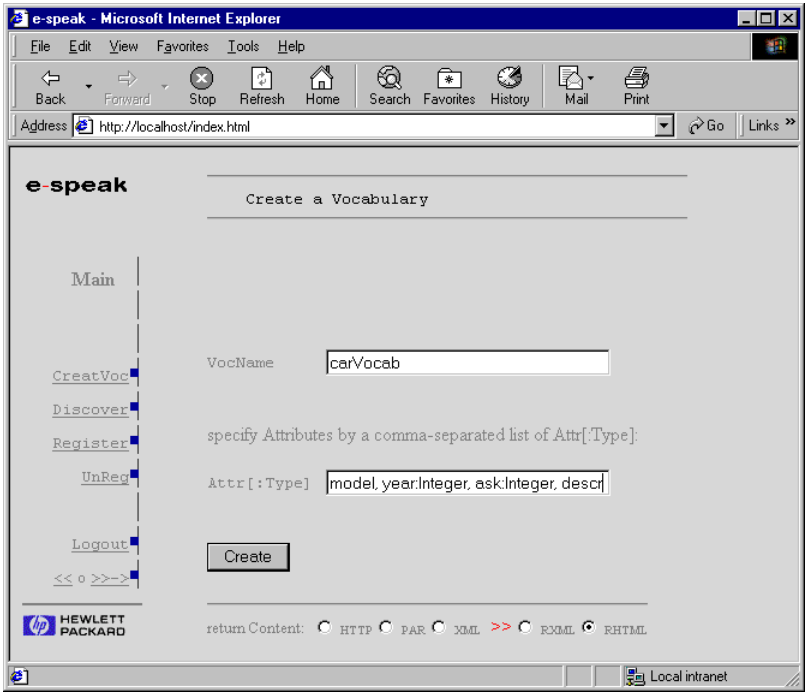


Figure 9 Create a Vocabulary.

To create a vocabulary, click the Create button for the vocabulary that you have defined.

Content Buttons

A line of radio buttons is shown at the bottom, right hand side in the HTML pages. These buttons can be used to control the content format returned to the browser reflecting various processing stages within WebAccess. This function is useful for

debuging purposes, and it can also be used to explore WebAccess' internal content representations. Viewing the HTTP and XML WebAccess generated code is an excellent guide to developer's application service development.

Selecting the **<HTTP>** button causes a page showing the original HTTP request being returned to the browser. An example is shown in Figure 10.

The **<PAR>** button shows the parameters passed with the HTTP request which controls the processing of the request. See Figure 11 for an example.

If the **<XML>** button is selected, the XML representation of the request is returned. The specified operation is not executed. Only the first step of mapping a request into WebAccess XML is performed, and the result is shown in the browser. An example is shown in Figure 12 below. The transformation is controlled by a template available under the web server's

```
<DocumentRoot>/templates/exsd10_templ.wm
```

The <**RXML**> button returns the reply content in XML after executing the requested operation. This button executes the operation specified in the request.

Finally, the pre-selected <**RHTML**> button executes the operation, and an XML to HTML transformation is applied to generate HTML shown in the browser. This transformation is controlled by a template available under the web server's

```
<DocumentRoot>/templates/html10_templ.wm
```

A public domain package webmacro is used for internal transformations of content. The two templates mentioned above are used to generate XML from parameters passed with HTTP GET (URL encoding) or POST requests (used in FORMs) or for transforming XML back into HTML for non-XML capable browsers.

Templates might be changed for customization purposes.

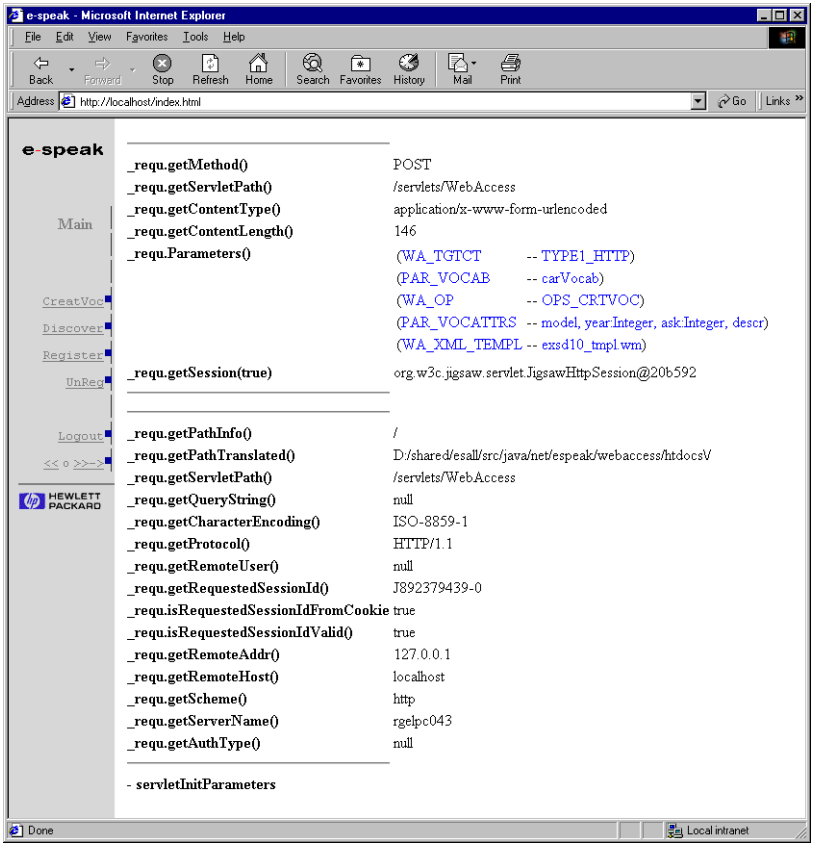


Figure 10 <HTTP> button selected -- return representation of HTTP request

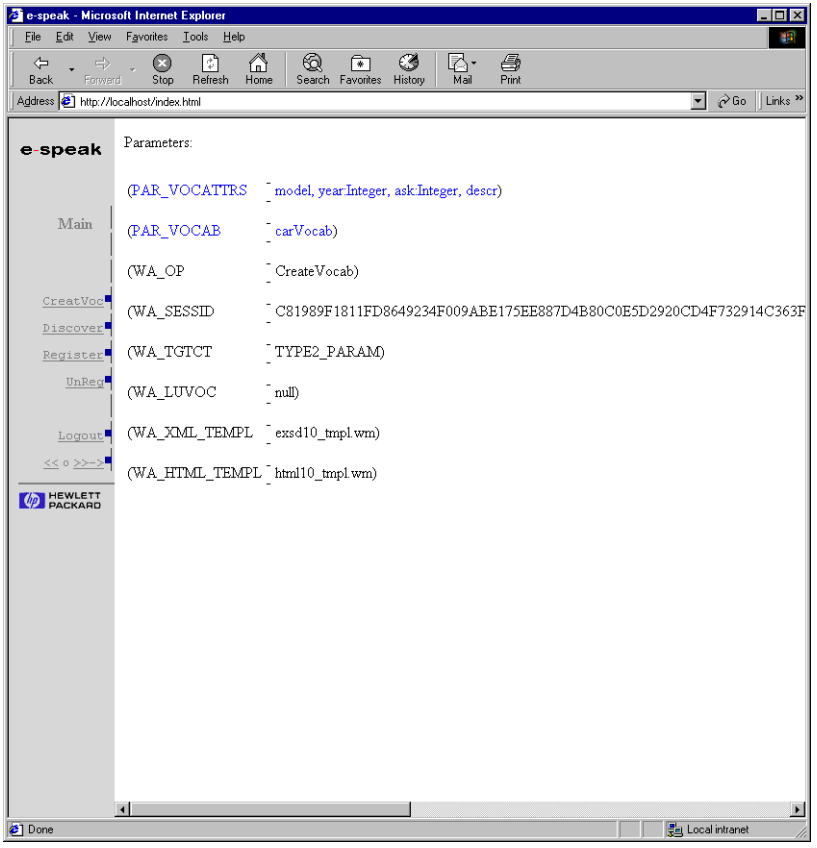


Figure 11 <PAR> button selected -- return parameter representation of HTTP request

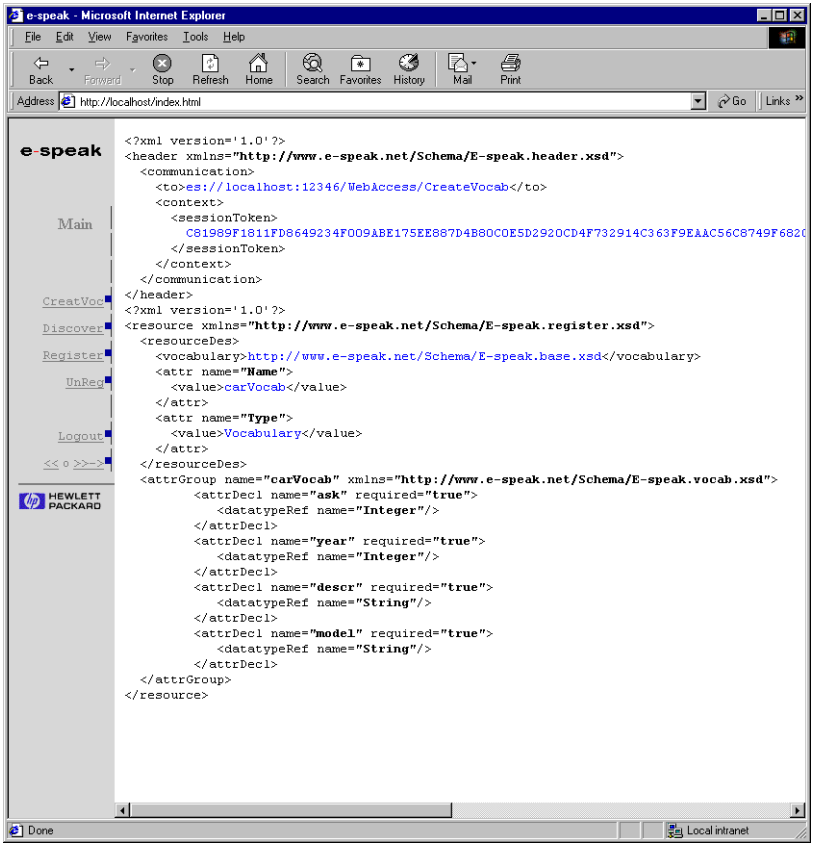


Figure 12 <XML> button selected -- return the internal XML representation of request

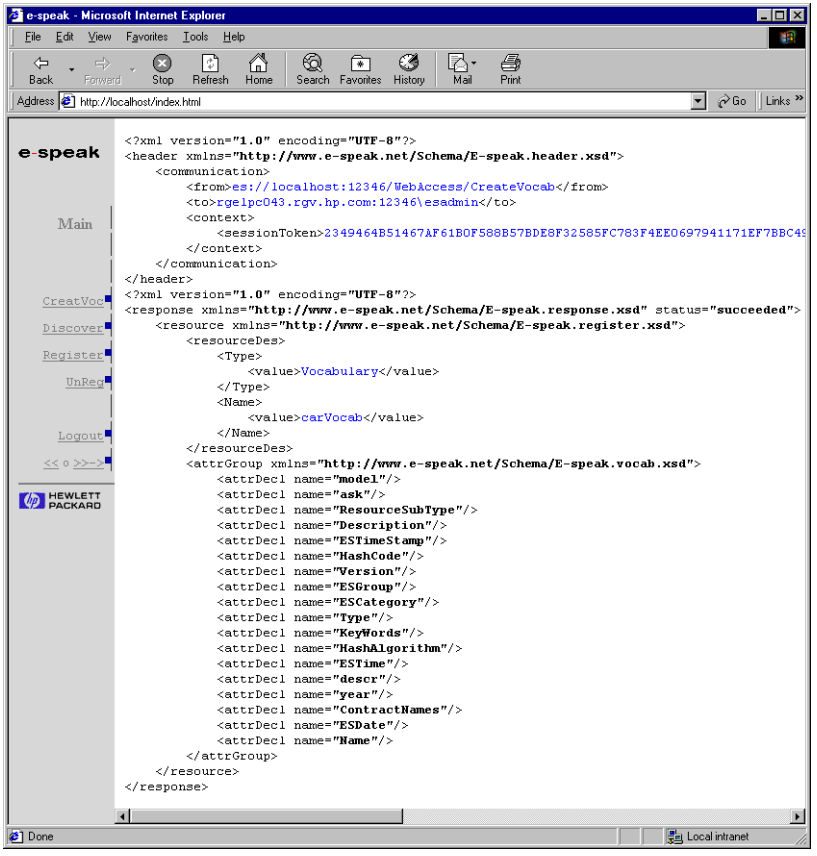


Figure 13 <RXML> button selected -- return XML representation of reply

If you do not select a return Content radio button, the HTTP request, the XML document request and the response is transparent to the user and the Main screen returns with the message succeeded, as shown in the figure below.

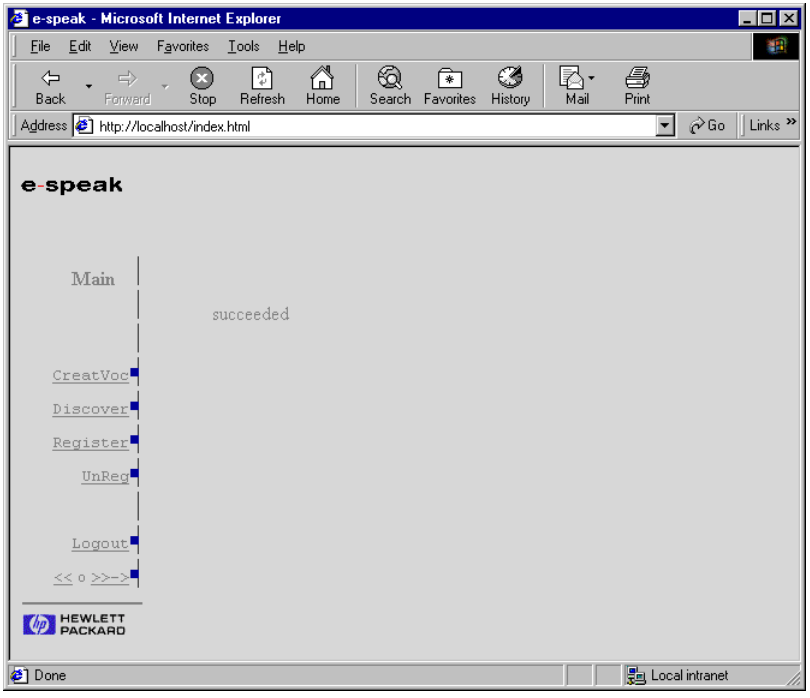


Figure 14 <RHTML> button selected -- return HTML version of reply content

Register a Service

To register a service with e-speak using the WebAccess Browser, select Register from the Main menu and complete the dialog box that appears in the Register a Service/ Resource screen, as shown below in the vocabulary service registration example.

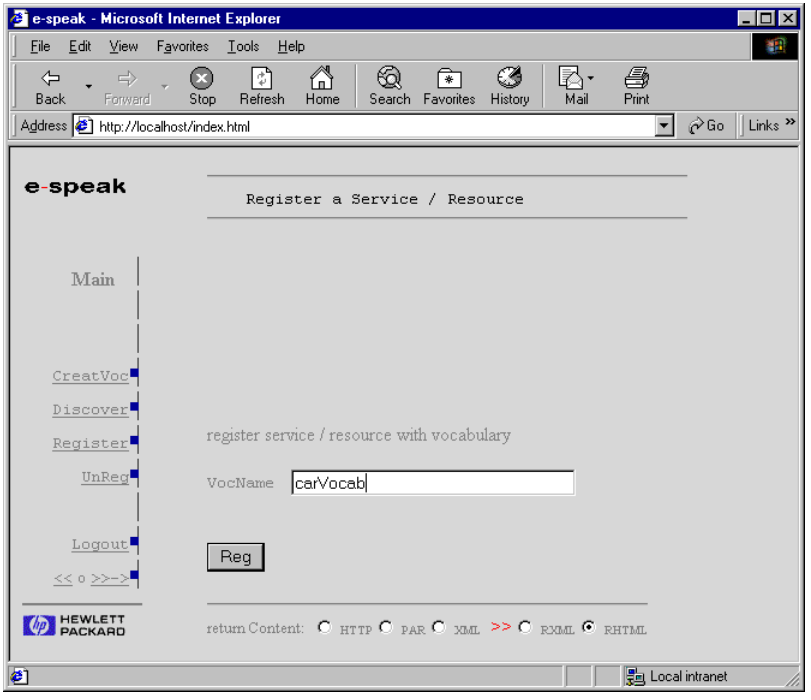


Figure 15 WebAccess Browser Register a Service/Resource Screen.

Click on the Reg button to open the Register screen and complete the vocabulary description by typing the values into the corresponding dialog boxes as shown below.

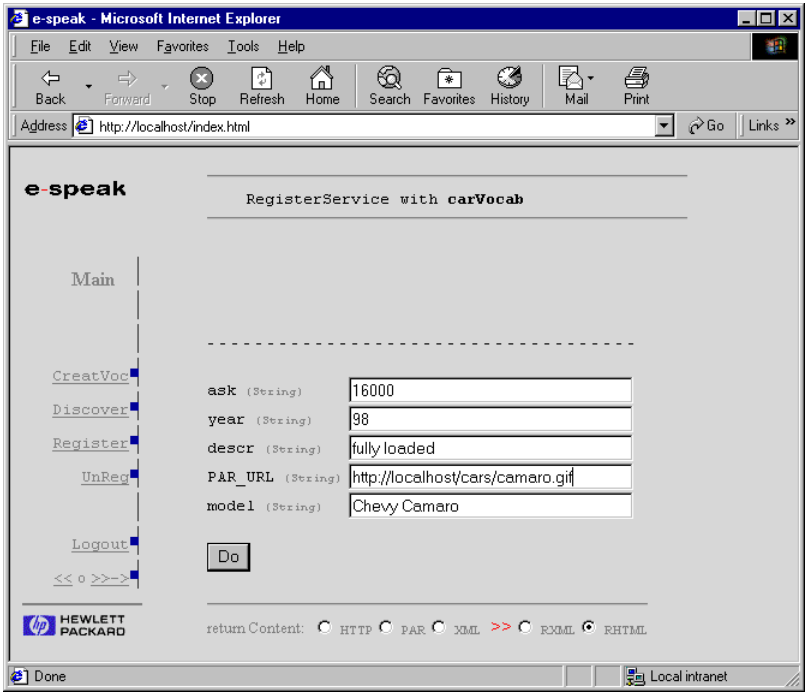


Figure 16 WebAccess Browser Register Service with carVocab attributes defined

The examples shown below show the XML document request and response generated during the service registration process. These screens only appear if one of the return Content radio button has been selected.

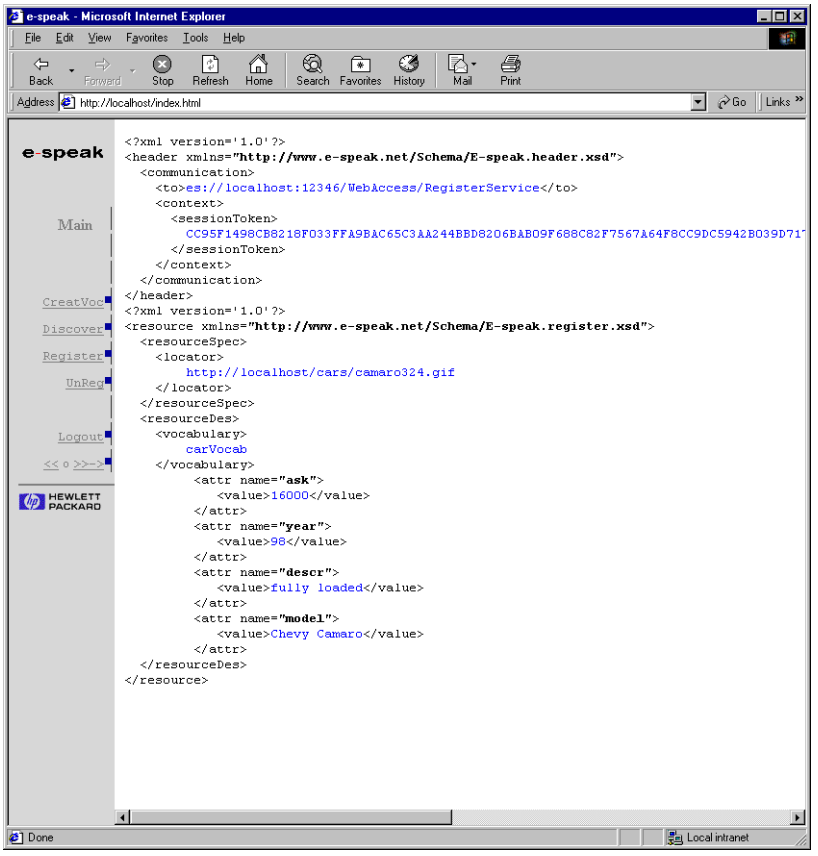


Figure 17 WebAccess Browser Generated XML document request for registration

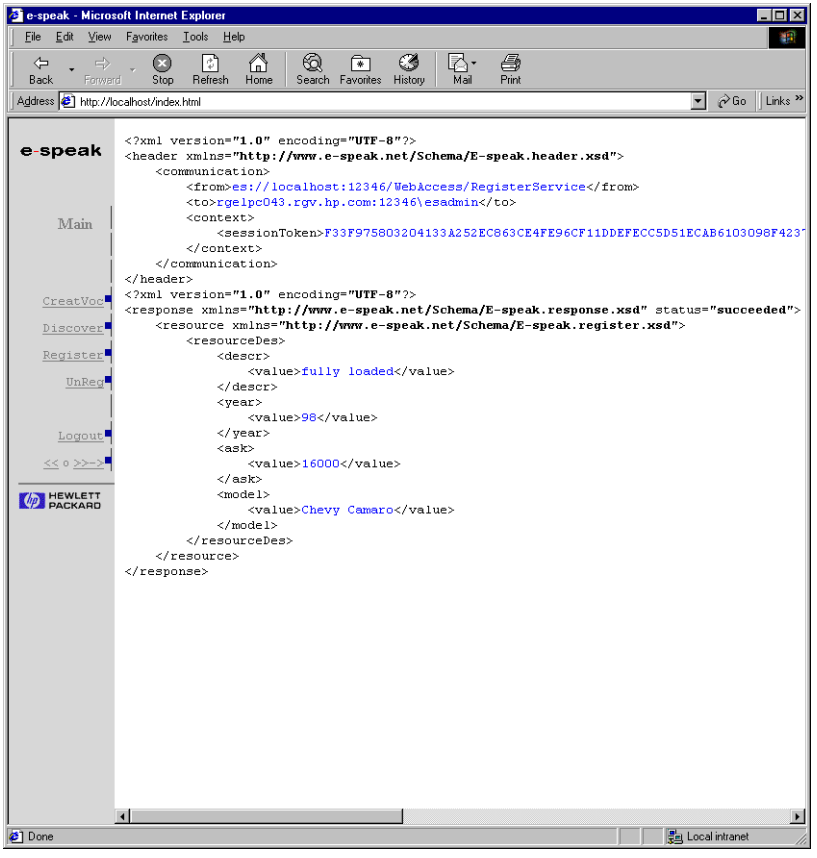


Figure 18 E-speak generated XML response document for carVocab registration

If none of the return Content radio buttons were selected, the WebAccess Browser returns a succeeded screen for a completed registration, as shown in the example below.

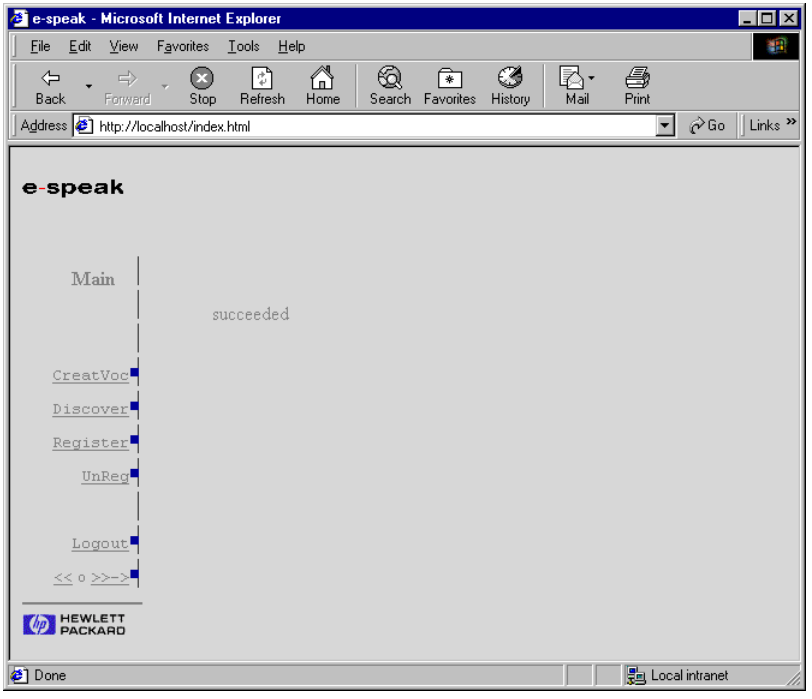


Figure 19 WebAccess Browser Main Screen -- Successful Message Returned

Discover a Service or Resource

To find a service or resource select Discover from the Main screen. Complete the dialog box and select a radio button in the return Content, then click on Disc, as depicted in the example shown below.

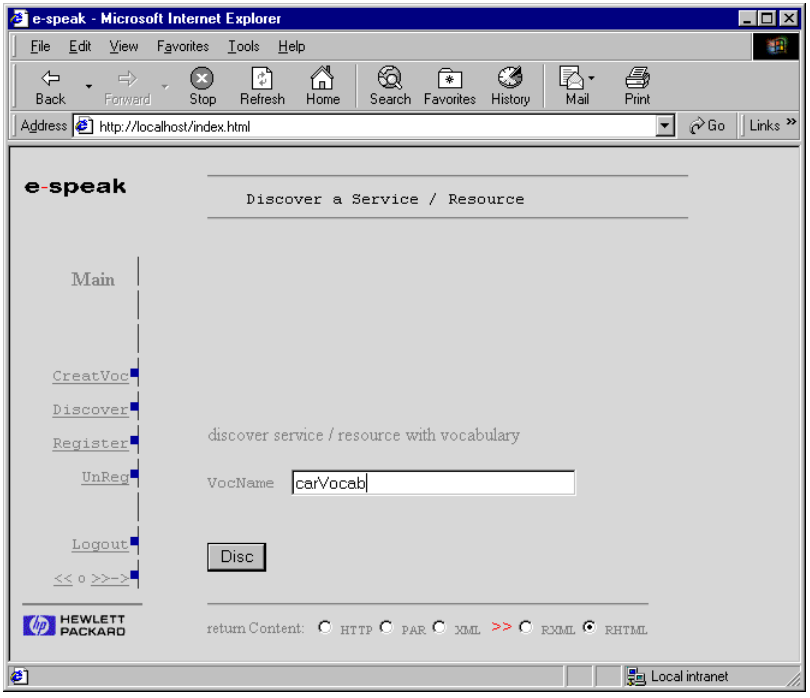


Figure 20 WebAccess Browser Discover a Service/Resource

The WebAccess Browser returns a FindService screen. In the example below, it is for the carVocab shown in the earlier creation and registration examples.

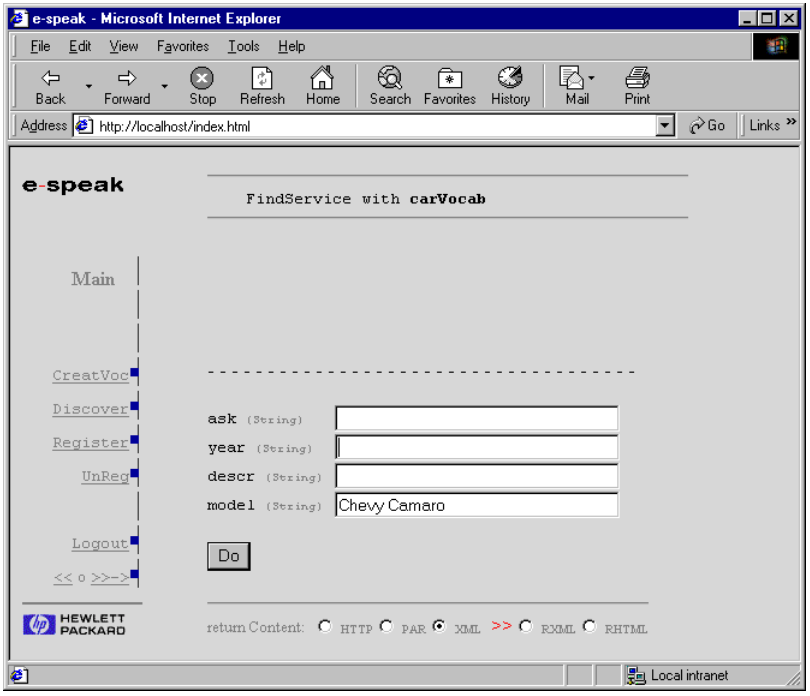


Figure 21 WebAccess Browser FindService

Define the attributes of the service search by completing the dialog boxes and selecting the Do button. Selecting the return Content XML radio button results in the screens shown below.

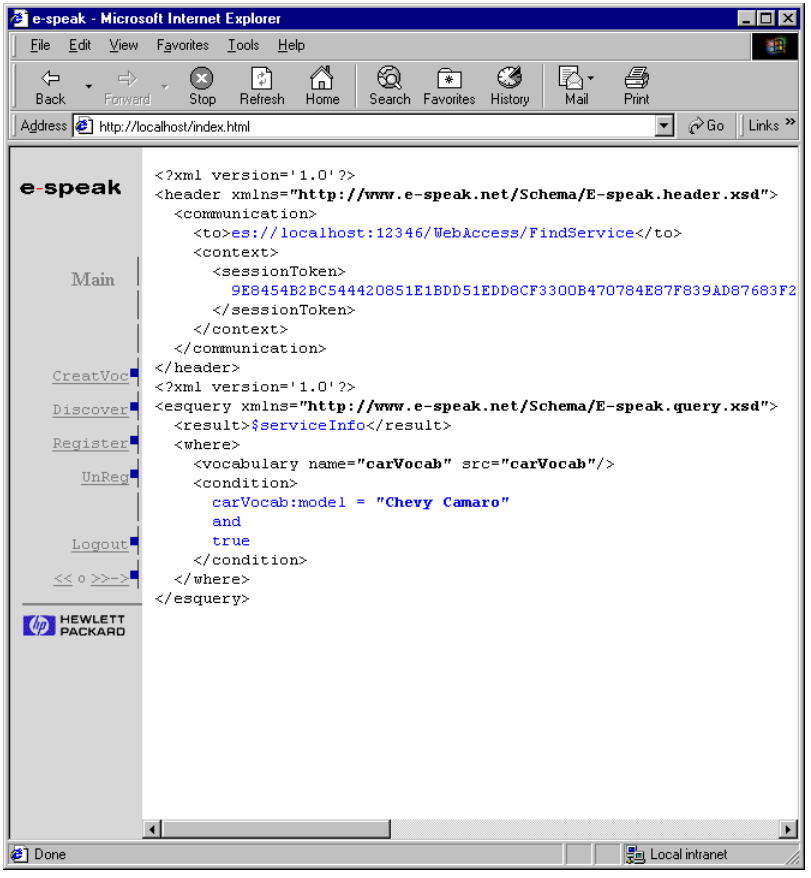


Figure 22 WebAccess Browser Discover a Service XML e-speak Query

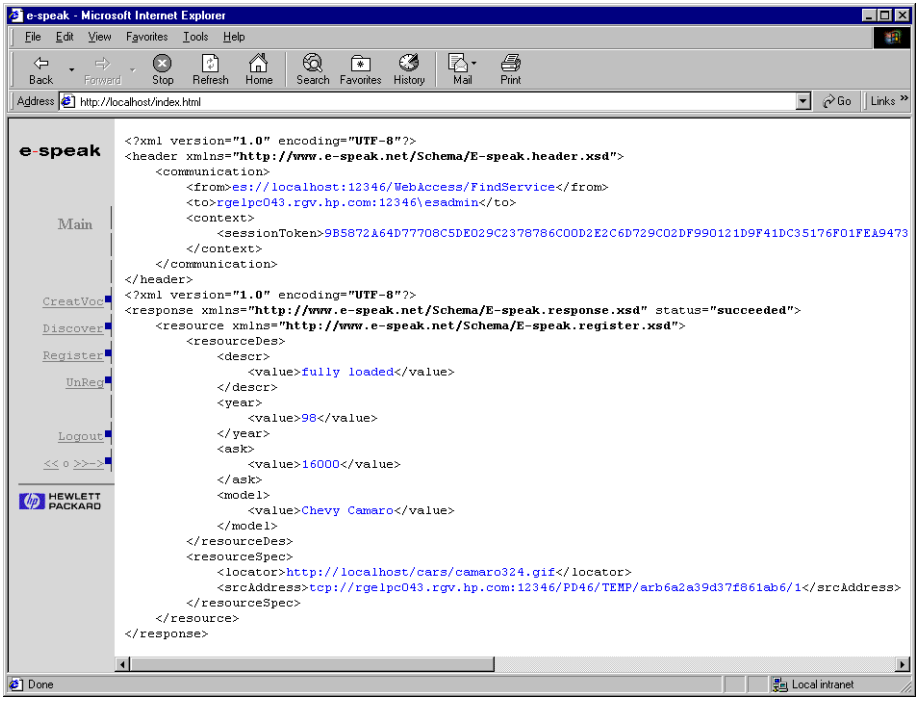


Figure 23 WebAccess Browser XML e-speak Response

The final result of the WebAccess Browser Discover a Service/Resource process is a presentation of the e-speak query results, as shown in the example below.

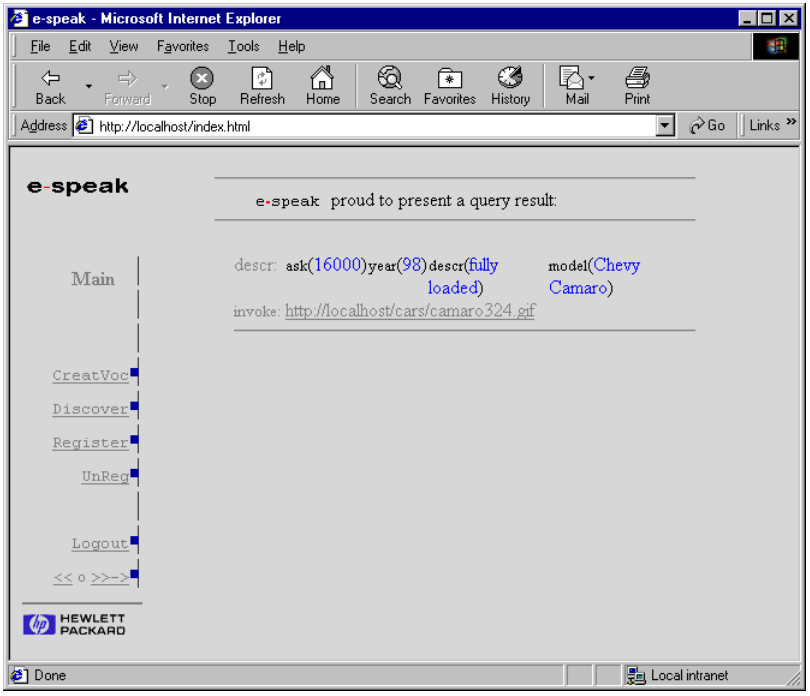


Figure 24 WebAccess Browser e-speak Query Result

If the make of the car had not been specified in this example, the query result looks like the example shown below. All registered resources are returned as shown in the figure below.

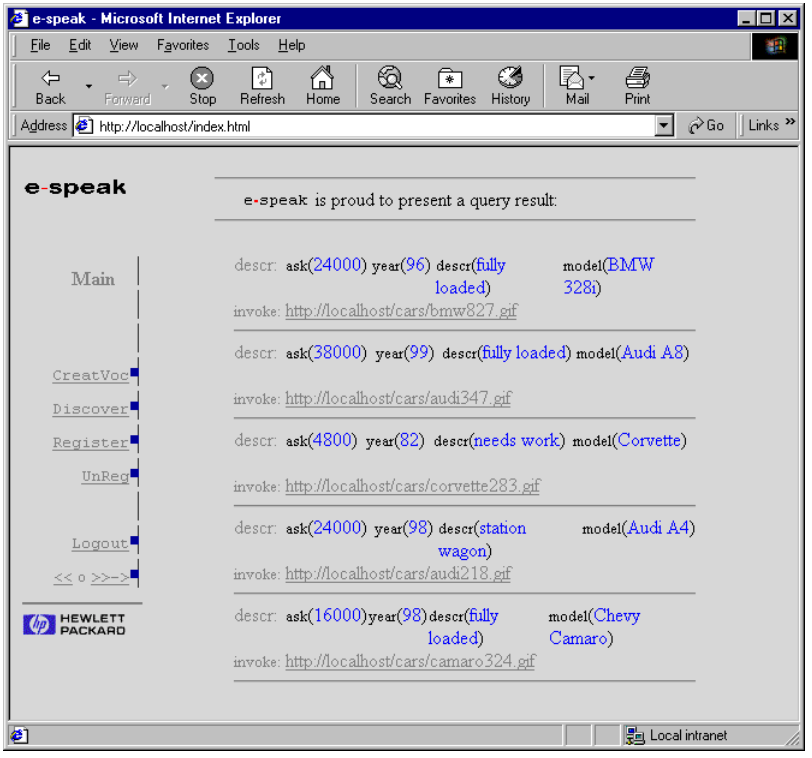


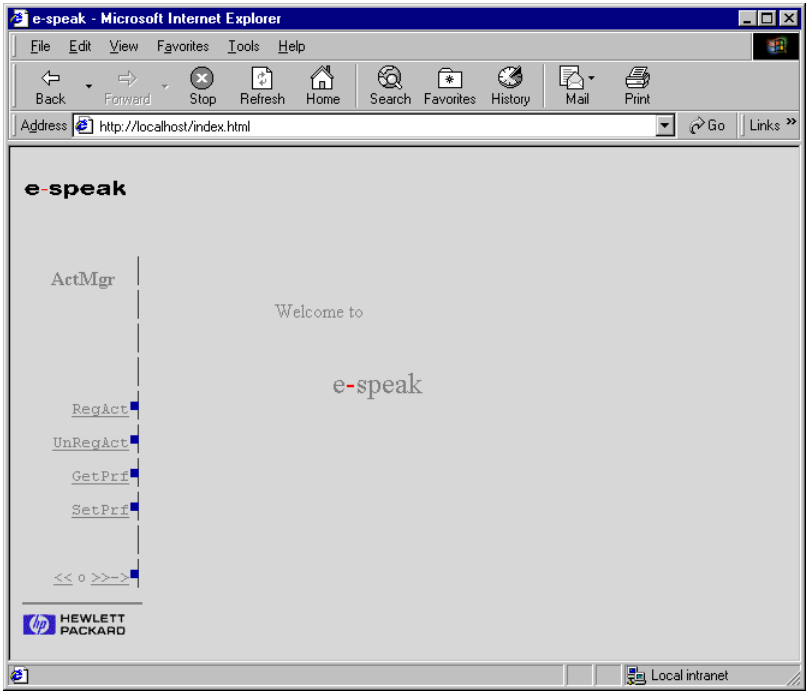
Figure 25 WebAccess Browser e-speak Query Results

ActMgr -- Account Management

Account management consists of registering and unregistering an account, creating and modifying a user profile and obtaining user profile information. In this section, registering an account is the only example for Account Management, using the WebAccess Browser.

In order to go to the Account Manager's menu bar, the **"<<o>>->"** navigation tabs should be used. They are at the bottom of the left side menu bar. Subsequent clicking any of the tabs scrolls through three menus appearing at the left side bar: `Main`, `ActMgr` and `PGMgr` referring to the respective sections.

The main Account Manager screen is shown below.



Register an Account

To register an account you select RegAct from the ActMgr menu. The Register an Account with e-speak screen appears as shown below. Complete the dialog boxes for the user name, password, then click on RegAct..

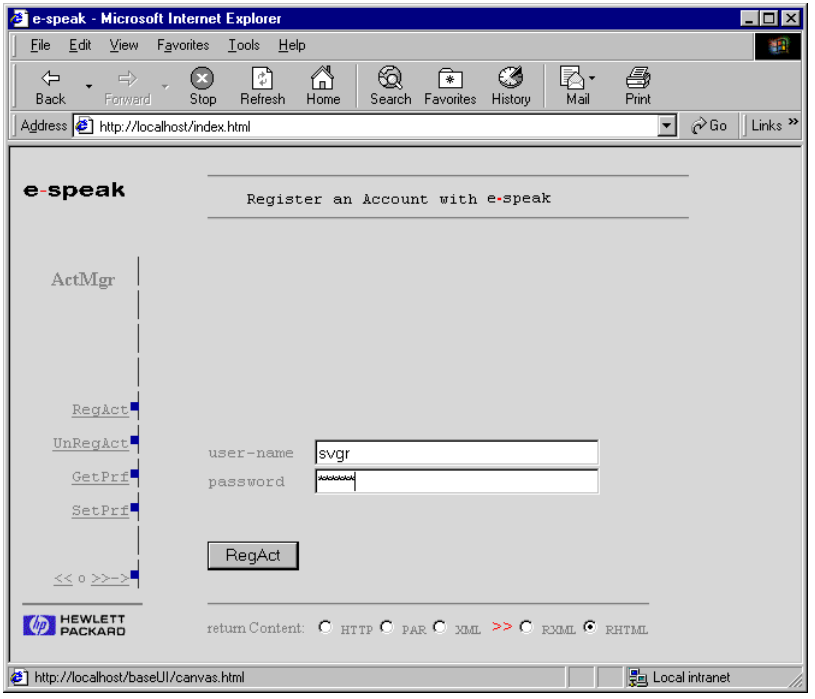


Figure 26 WebAccess Browser Register and Account with e-speak screen

Permissions for registering new users are obtained based on the current session the user is in.

By selecting the XML radio button on the return Content line of the Register an Account with e-speak screen, the following XML request and XML response screen appear.

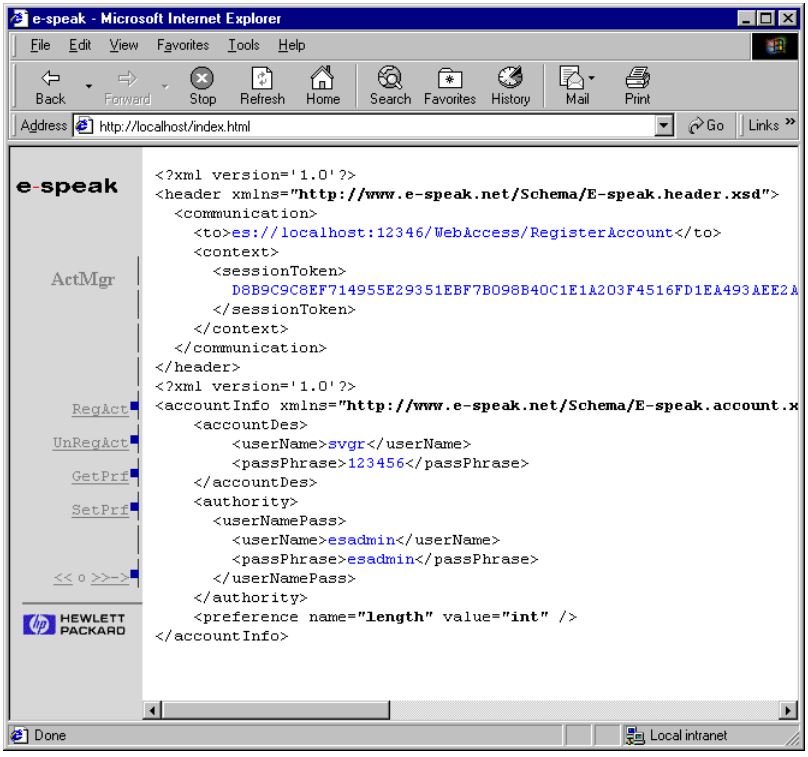


Figure 27 WebAccess Browser Generated ActMgr Registration XML Request

E-speak returns the screen shown below.

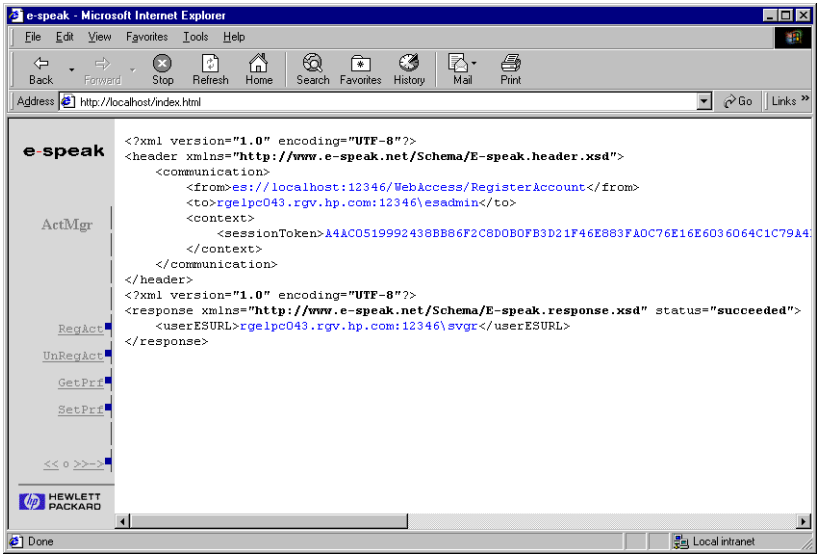


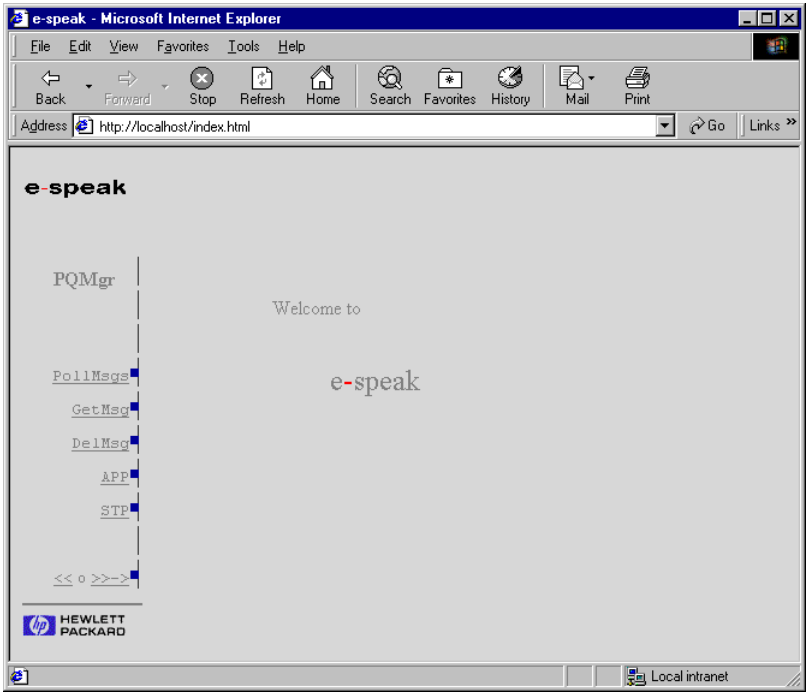
Figure 28 WebAccess e-speak Generated XML Account Registration Response

The different screens shown in this section also demonstrate how the content buttons can be used to inspect results. It may also be used to identify how the appropriate XML or HTTP parameters have to look like for buildingclient applications.

Account and Session Management is described in detail in chapter 8 of this document.

PQMgr -- Persistent Queue Management

Persistent queue management provides the interfaces and infrastructure for storing and retrieving email-like messages persistently stored in the WebAccess data base. The WebAccess PQMgr Browser screen enables you to poll messages, get messages, delete messages and configure messages. The menu for the PQMgr is shown below. It is reachable through the **"<<o>>->"** navigation tabs.



Pesistent Messages are described in detail in chapter 9 of this document.

APP -- Asynchronous Persistent Poll Messages

Selecting APP opens the Asynchronous Persistent Polling Msg (APP) screen as shown below.

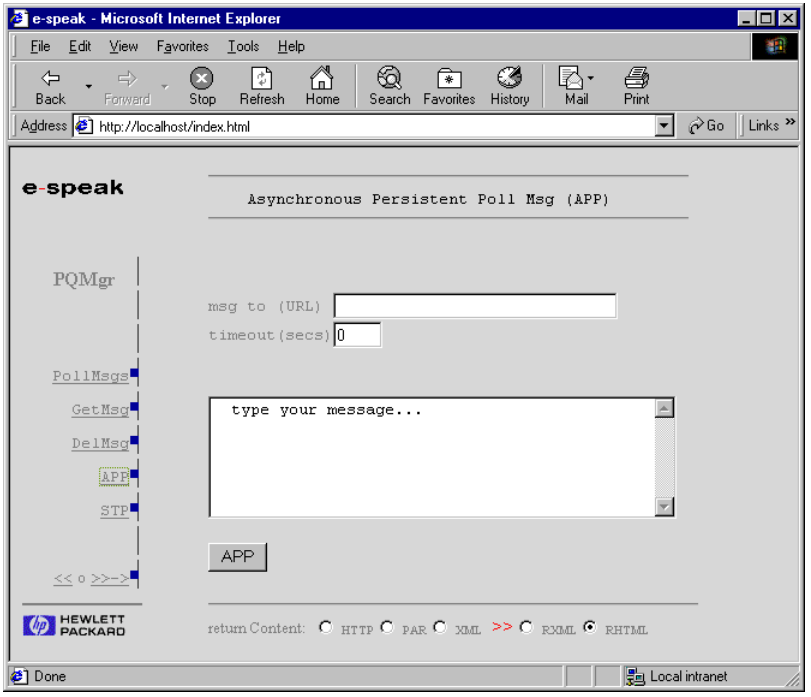


Figure 29 Asynchronous Persistent Polling Msg (APP) screen

APP messages are stored in a data base (Oracle or mySQL) for specified users who later might poll for these messages.

Complete the msg to (URL) and timeout (SECS) and message content dialog boxes. Define the return contents by selecting a return Content radio button and click on the APP button. The example shown below depicts a completed APP screen.

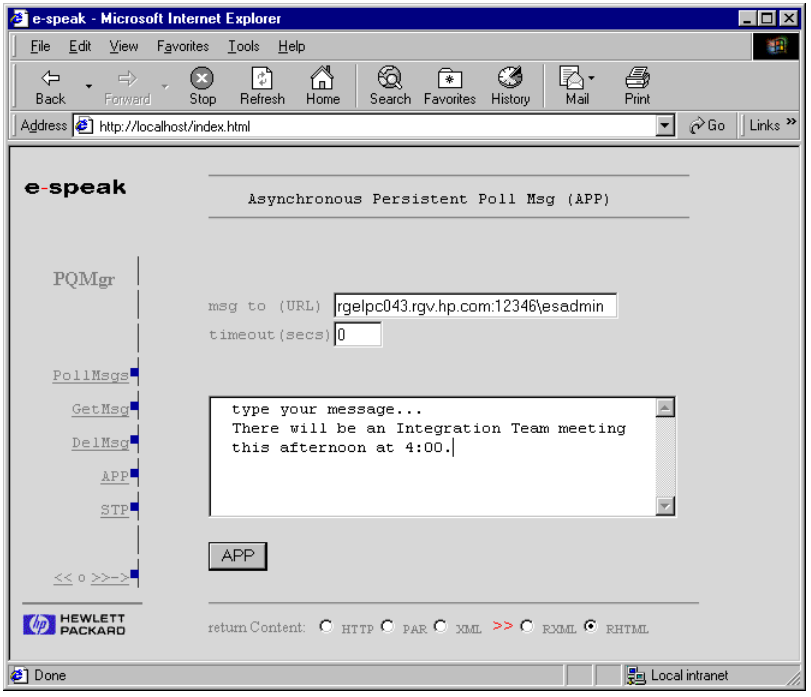


Figure 30 WebAccess Browser PQMgr APP completed dialog boxes

If the return contents were defined as XML, the following screen appears.

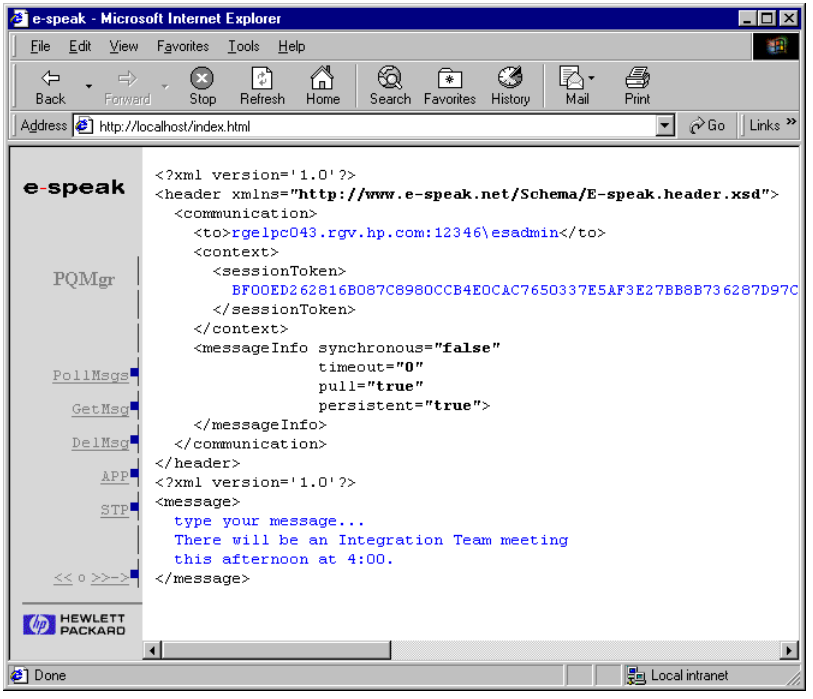


Figure 31 WebAccess Browser APP Message XML Request

An example of the response from e-speak is shown below. It also shows how the XML looks like applications may send to WebAccess to store APP messages.

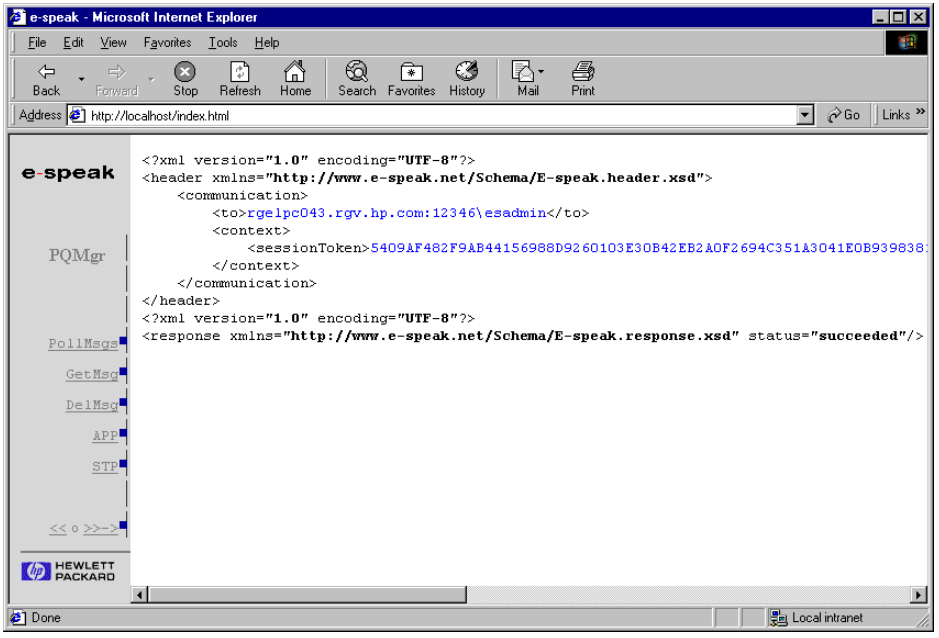


Figure 32 WebAccess Browser PQMgr APP e-speak XML Response

PollMsgs -- Poll for messages arrived for a user

To poll for messages, select PollMsgs from the PQMgr menu. The List/Poll for Persistent Messages appears as shown below.

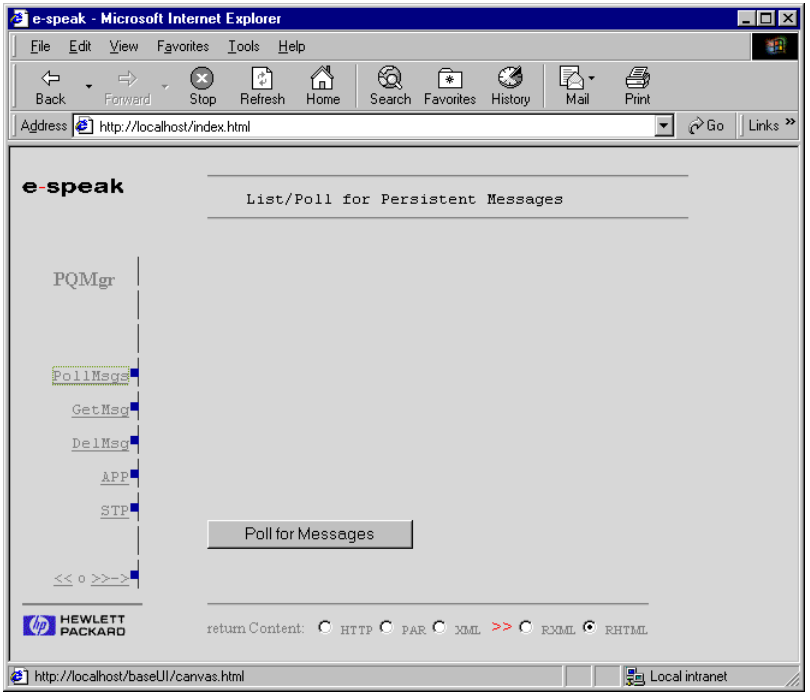


Figure 33 WebAccess Browser List/Poll for Persistent Messages screen

Click on Poll for Messages and the screen below appears. All persistent messages are displayed by Message ID and sender.

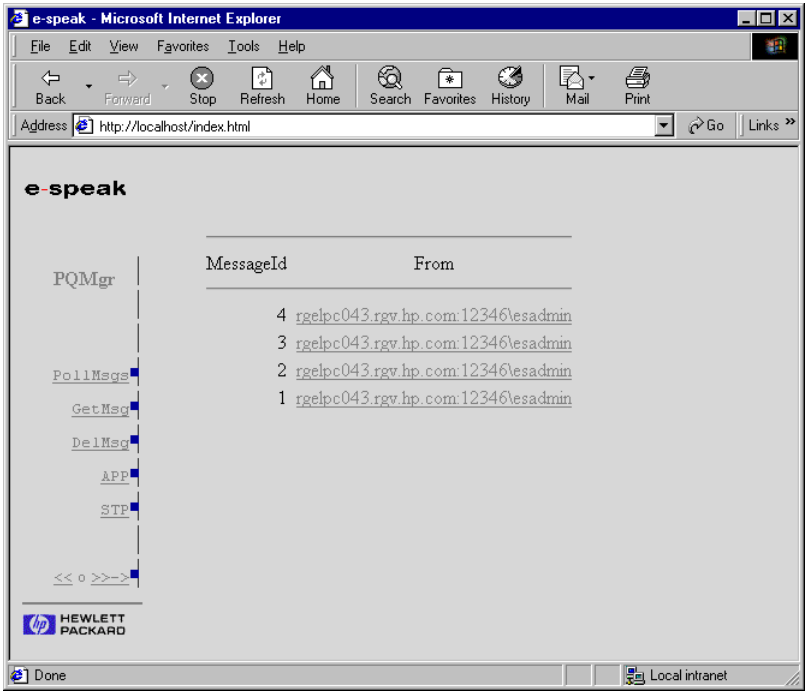


Figure 34 WebAccess Browser Message Listing screen

To get a particular message, click on the pathname below From. A PQMgr screen appears that displays the details of the message. Details include the Message ID, sender, message properties and message contents.

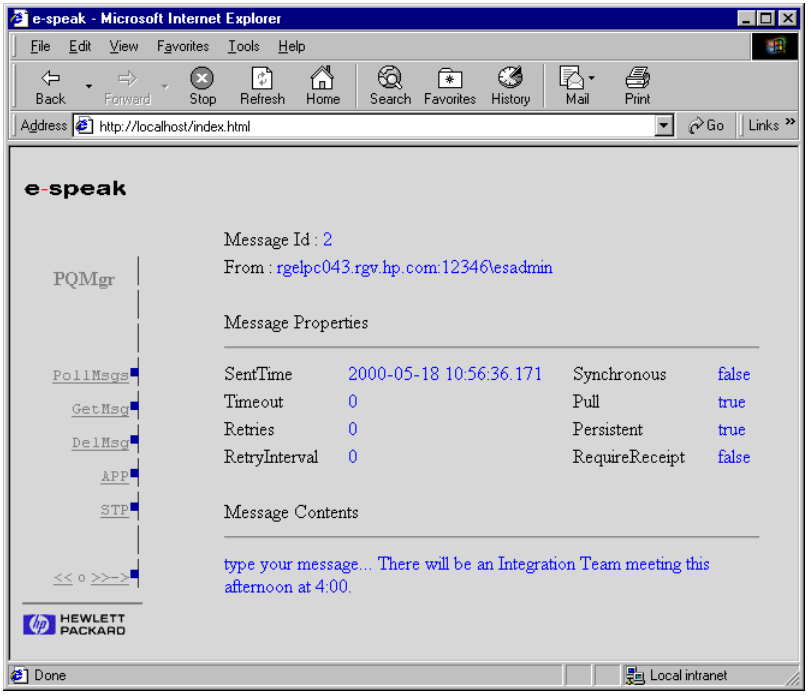


Figure 35 WebAccess Browser PQMgr Message Properties Screen

GetMsg -- Get a particular message

If you know the message ID, you can select GetMsg which displays the screen shown below. By typing the message ID number in the dialog box and clicking on GetMsg you can also obtain a specific message, with its properties, from the queue.

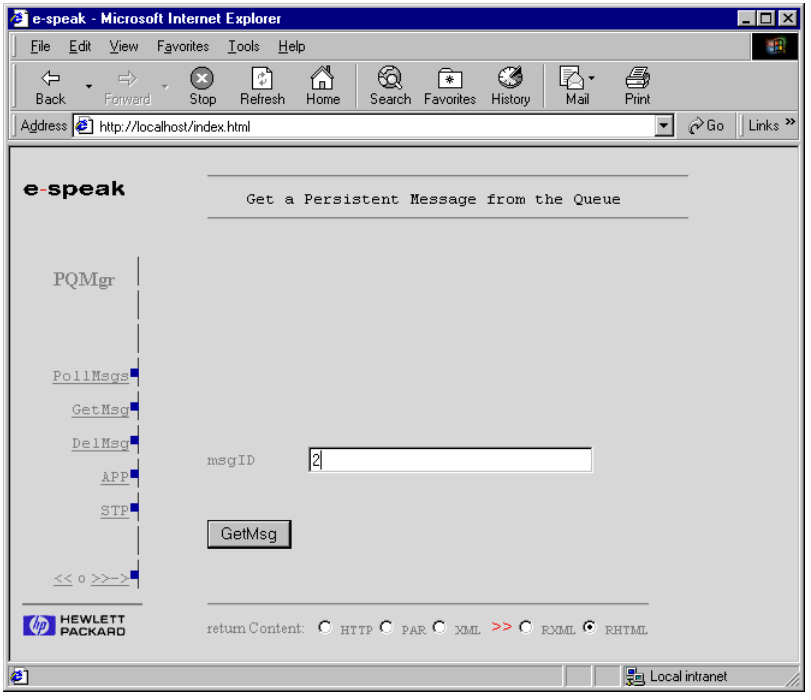


Figure 36 WebAccess Browser PQMgr GetMsg Screen

If the return Content is defined as XML, the screen shown below appears.

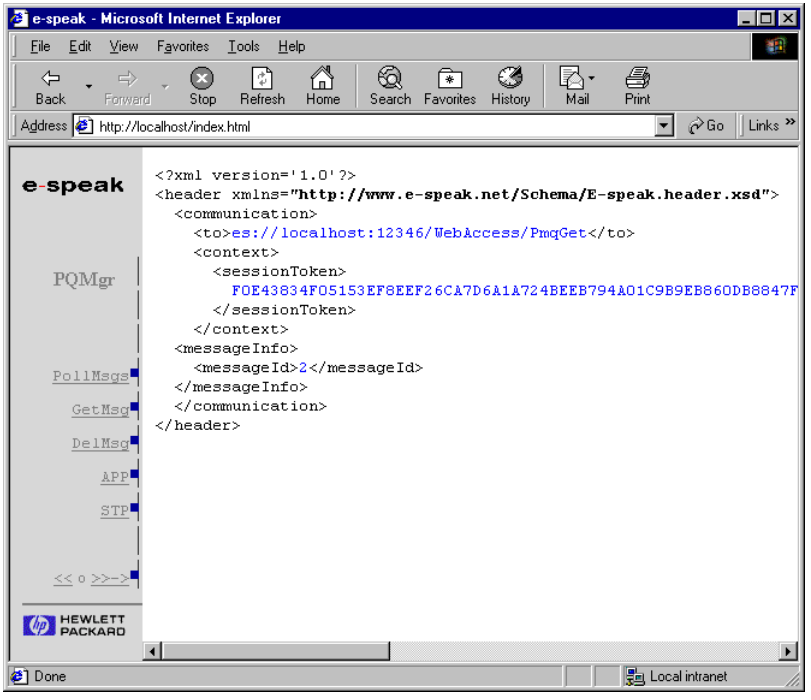


Figure 37 WebAccess Browser PQMgr Get Message XML Request

Session Expiration

WebAccess uses sessions as described in detail in chapter 8 of this document. Sessions are based on accounts through which users or client applications log in. Sessions are closed at logout. Sessions expire after a certain period of inactivity as configured in webaccess.xml.

When a session expires, the browser reports this by the following screen..

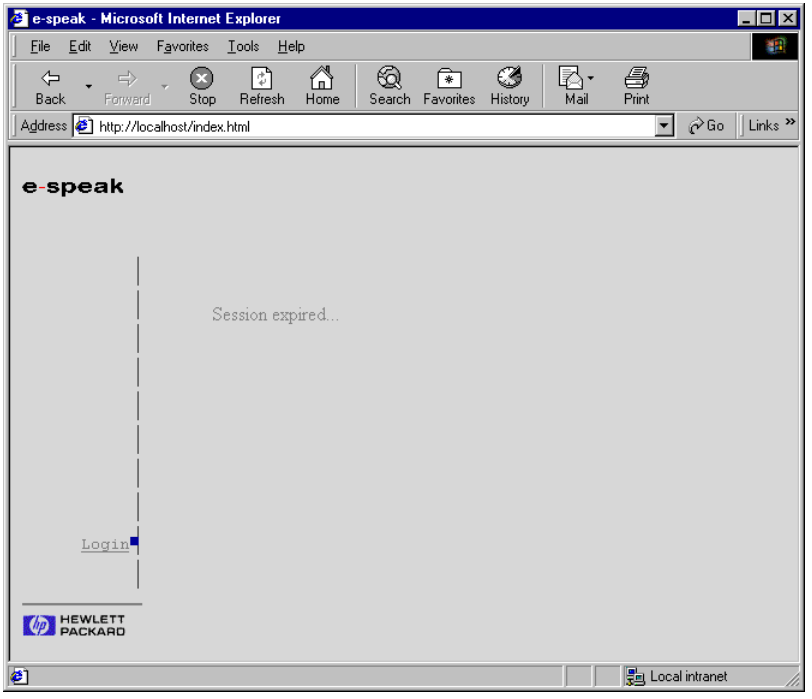


Figure 38 WebAccess Browser Session Expired screen

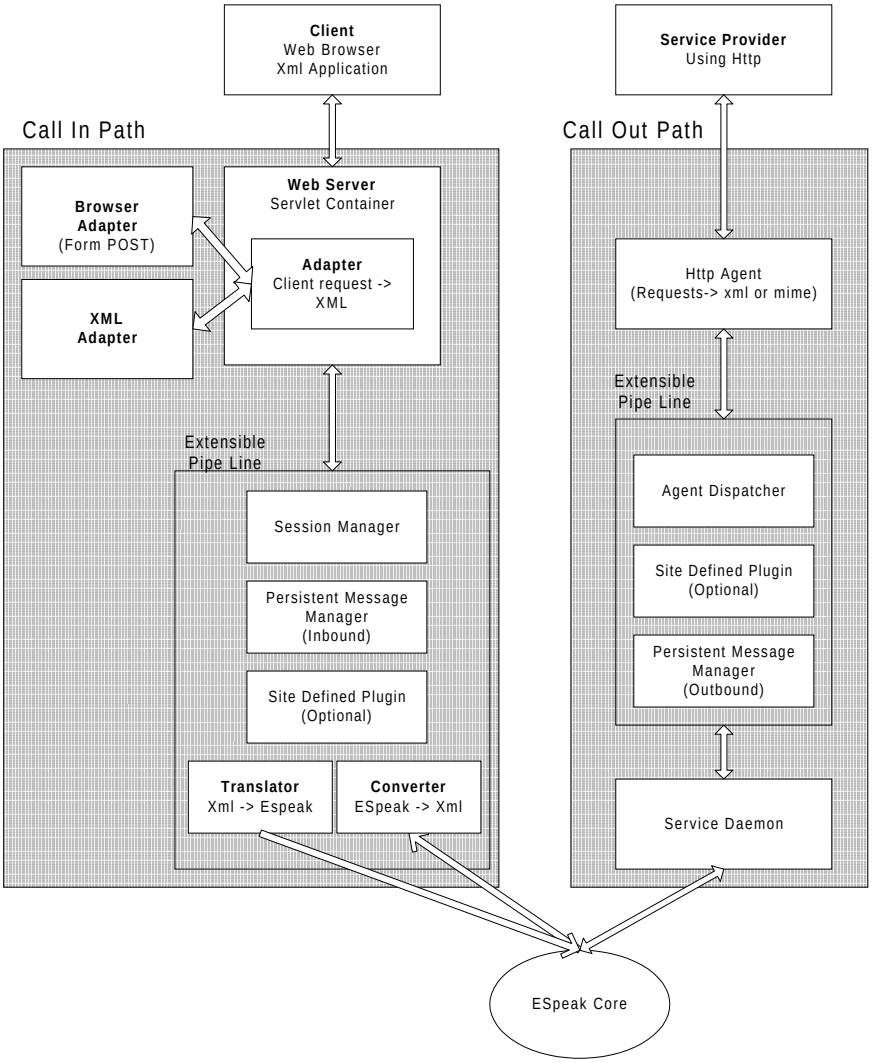
Chapter 4 WebAccess Overview

WebAccess: The E-speak Gateway

WebAccess consists of a stack of modules. Requests are pushed down the stack until they are delivered to the e-speak engine. The topmost module is the adapter module that generates XML DOM trees from a request. The adapter module is also responsible for transforming FORM POST requests from a web browser to XML documents and XML replies to HTML/WML documents. The generated DOM trees are passed to the session manager where login and logout requests are handled.

Only authenticated clients may send messages through WebAccess. The next module is the persistent message queue manager which handles poll requests. For example it allows mobile clients to have intermittent connections to WebAccess. Finally the requests are sent to the e-speak engine where they are routed to target services or handled by the engine itself depending on the requested operation.

WebAccess Architectural Overview



External Components

Client – Typically a custom built application that communicates with the WebAccess system using the HTTP protocol and text/xml or multipart MIME format. A web browser can be used for simple interactions and for trouble shooting because WebAccess comes with a simple HTML application.

Service Providers – An application that can process client requests. It must be registered by a client before other clients can use it, and it must log in as a client to process any requests that are being held by the Inbound Persistent message manager.

Internal Components

- **Web Server** – The web server can be any that supports java servlets version 2.0. Web Access has been tested with Apache version 1.3 and Jserv version 1.1 on Windows NT 4.0 and Linux version 6.0. Webaccess can be used with SSL enabled web servers.
- **Adapter Dispatcher** – Converts HTTP POST requests into XML using the Browser and XML Adapters. This component also handles the conversion of XML responses to HTML. It uses a series of template files to aid in the creation of HTML and xml documents. These templates can be customized by a site to change the look of the HTML responses. A client that uses URL encoded form post typically receives HTML responses. Clients that use text/xml or mine multipart usually receive a response in the same encoding. When a service is invoked, the response is defined by the service provider.
- **Browser Adapter** – Converts HTTP POST requests in URL encoded form to XML. Special HTML form parameter names are used to build the XML request. These parameters are described in Chapter 3. This adapter also processes multipart MIME requests. Use of multipart MIME is preferred since it allows binary data to be included in a service invocation request. In addition, requests for e-speak core services require the use of more than one xml document and multipart MIME is preferred for combining several XML documents into one request.
- **XML Adapter** – Converts HTTP POST that use the content type “text/xml” to XML requests. The format of the XML requests is described in appendix A.

- **Extensible pipeline** – There are two pipe lines that can be extended by a site to process inbound and outbound requests. Extensions are specified webaccess.xml document that is stored in <E-SPEAK-HOME>/extern/webmacro.
- **Session Manager** – This pipe line component manages the session token for each client. It prevents access to the system if the request contains an invalid session token. Session tokens are designed so that they can be used in a cluster of webaccess servers that communicate with the same e-speak core.
- **Persistent Message Manager (Inbound)** – This pipe line component stores messages that clients can pick up later. This is similar to an email inbox except that messages have expiration dates and are automatically removed if they are not read.
- **Site Defined Plug-in** – This is a place holder for any plug-ins that a site may wish to add to the pipe line to do extra processing. A site defined plug-in can be placed anywhere in the Call In or Call Out pipe lines.
- **Translator** – Converts XML requests to API calls to the e-speak core.
- **Converter** – Converts e-speak responses to XML.
- **Espeak core** – This module manages services, resources and users, etc.
- **Service Daemon** – This module processes requests for service invocations from the e-speak core and sends them to the call out pipe line. It returns service specific responses to the requesting client.
- **Persistent Message Manager (Outbound)** – Stores messages for service providers that are temporarily disconnected from the system. It retries sending the messages for a specific time period.
- **Agent Dispatcher** – Picks an agent to deliver a request to a service provider. A site can extend the list of agents. An agent typically handles a specific protocol for communicating with multiple service providers. For example, an agent can be created to communicate with service providers that are accessible via a custom protocol.
- **HTTP Agent** – This built-in agent sends requests to service providers that can accept HTTP POSTs in multipart MIME and/or “text/xml” format.

Chapter 5 WebAccess Examples

This chapter shows examples of the requests through which users can invoke e-speak internal services, such as create vocabulary, register service, find service, and account management. The responses are also XML documents. The syntax for requests and responses is defined in e-speak XML schemas. (See the Appendix.) All operations exception *Login* must be in valid sessions.

The request to an e-speak internal service consists of two parts:

- E-speak header document, which specifies the ESURL of the internal service, the requestor and the related routing information. (ESURLs are case sensitive).
- Request body document which provides the actual parameters for the service.

Login

The user submits his user name and passphrase to e-speak. E-speak authenticates the identity. If the user name was registered with e-speak and the passphrase matches the user's passphrase, e-speak responds with the user's ESURL and establishes a session. Otherwise, e-speak returns an appropriate error message.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
                <to>es://localhost:12346/WebAccess/Login</to>
                <from></from>
```

```
       </communication>
</header>

Login Document
<?xml version="1.0"?>
<userNamePass
xmlns="http://www.e-speak.net/Schema/E-speak.account.xsd">
   <userName>WAUser1</userName>
   <passPhrase>flyESpeak</passPhrase>
</userNamePass>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
   <communication>
   <to> </to>
      <from>es://localhost:12346/WebAccess/Login</from>
      <context>
         <sessionToken>1234567890</sessionToken>
      </context>
   </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
   <userESURL>es://localhost:12346\WAUser1<\userESURL>
</response>
```

Failure:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
    <to></to>
        <from>es://localhost:12346/WebAccess/Login</from>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Username/password not correct</responseInfo>
</response>
```

Logout

The logout operation closes the session for the user. Only the header is needed.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
                <to>es://localhost:12346/WebAccess/Logout</to>
                <from> es://localhost:12346\WAUser1</from>
               <context>
                    <sessionToken>1234567890</sessionToken>
              </context>
      </communication>
</header>
```

Response

Success:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd">
    <communication>
        <!-- This is the user's esurl -->
        <to>es://localhost:12346\WAUser1</to>
        <from>es://localhost:12346/WebAccess/Logout</from>
        <context>
           <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version="1.0"?>
<response status="succeeded">
</response>
```

RegisterAccount

The *RegisterAccount* operation creates an account in e-speak. The request has two parts:

- account description, the user name and pass phrase of the account
- authority, the authority of this operation. It can be the user himself or superuser (esadmin).

If the operation is succeeded, it replies the account's ESURL; otherwise, error messages appear.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>

<to>es://localhost:12346/WebAccess/RegisterAccount</to>
                 <from>es://localhost:12346\esadmin</from>
                 <context>
                    <sessionToken>1234567890</sessionToken>
                 </context>
       </communication>
</header>
```

Account Document

```
<?xml version="1.0"?>
<accountInfo
xmlns="http://www.e-speak.net/Schema/E-speak.account.xsd">
   <accountDes>
   <UserName>WAUser1</UserName>
      <PassPhrase>flyESpeak</PassPhrase>
   </accountDes>

   <authority>
      <userNamePass>
      <userName>esadmin</userName>
          <passPhrase>esadmin</passPhrase>
      </userNamePass>
   </authority>
</accountInfo>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
<to>es://localhost:12346\esadmin</to>
        <from>es://localhost:12346/WebAccess/RegisterAccount</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
    <userESURL>es://localhost:12346\WAUser1<\userESURL>
</response>
```

Failure:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <!-- This is the admin’s esurl -->
        <to>es://localhost:12346\esadmin</to>
        <from>es://localhost:12346/WebAccess/RegisterAccount</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
```

```
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>User already exists</responseInfo>
</response>
```

UnregisterAccount

The *UnregisterAccount* operation removes an account in e-speak. The request has two parts:

- account description, the user name and pass phrase of the account
- authority, the authority of this operation. It can be the user himself or superuser (esadmin).

If the operation is successful, it does not reply; otherwise, it provides an error message.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>

<to>es://localhost:12346/WebAccess/UnregisterAccount</to>
                <from>es://localhost:12346\esadmin</from>
                <context>
                   <sessionToken>1234567890</sessionToken>
                </context>
       </communication>
</header>
```

Account Document

```
<?xml version="1.0"?>
<accountInfo
xmlns="http://www.e-speak.net/Schema/E-speak.account.xsd">
   <accountDes>
   <UserName>WAUser1</UserName>
   </accountDes>

   <authority>
      <userNamePass>
      <userName>esadmin</userName>
          <passPhrase>esadmin</passPhrase>
      </userNamePass>
   </authority>
</accountInfo>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <!-- This is the admin’s esurl -->
        <to>es://localhost:12346\esadmin</to>

<from>es://localhost:12346/WebAccess/UnregisterAccount</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
</response>
```

Failure:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <!-- This is the admin’s esurl -->
        <to>es://localhost:12346\esadmin</to>

<from>es://localhost:12346/WebAccess/UnregisterAccount</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Could not unregister user</responseInfo>
</response>
```

CreateVocabulary

The following example creates a vocabulary named *car-dealer-simple*, describing it in the base vocabulary (http://www.e-speak.net/Schema/E-speak.base.xsd) and registering it with e-speak. The attribute properties of the vocabulary are described in <attrGoup>, each of them is described by <attrDecl>.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
               <to>es://localhost:12346/WebAccess/CreateVocab</to>
                <from>es://localhost:12346\WAUser1</from>
                <context>
                    <sessionToken>1234567890</sessionToken>
                </context>
   </communication>
</header>
```

Creating the Vocabulary Document

```
<?xml version="1.0"?>
<resource
xmlns="http://www.e-speak.net/Schema/E-speak.register.xsd" >
   <resourceDes>

<vocabulary>http://www.e-speak.net/Schema/E-speak.base.xsd</vocabu
lary>
        <attr name=”Name”>
            <value>car-dealer-simple</value>
        </attr>
        <attr name=”Type”>
             <value>Vocabulary</value>
        </attr>
    </resourceDes>
```

```
    <attrGroup name="Simple Car dealer vocabulary”

xmlns=”http://www.e-speak.net/Schema/E-speak.vocab.xsd”>
         <attrDecl name="Manufacturer" required="true">
               <datatypeRef name="string"/>
         </attrDecl>
         <attrDecl name="Model" required="true">
               <datatypeRef name="string"/>
         </attrDecl>
         <attrDecl name="Price" required="false">
               <datatypeRef name="float">
                  <default>0.0</default>
                  <minInclusive>0.0</minInclusive>
                  <maxInclusive>100000.0</maxInclusive>
               </datatypeRef>
         </attrDecl>
   </attrGroup>
</resource>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <!-- This is the admin’s esurl -->
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/CreateVocab</from>
        <context>
              <sessionToken>1234567890</sessionToken>
         </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
    <attrGroup name="Simple Car dealer vocabulary”

xmlns=”http://www.e-speak.net/Schema/E-speak.vocab.xsd”>
         <attrDecl name="Manufacturer" required="true">
               <datatypeRef name="string"/>
         </attrDecl>
         <attrDecl name="Model" required="true">
               <datatypeRef name="string"/>
         </attrDecl>
         <attrDecl name="Price" required="false">
               <datatypeRef name="float">
                  <default>0.0</default>
                  <minInclusive>0.0</minInclusive>
                  <maxInclusive>100000.0</maxInclusive>
               </datatypeRef>
         </attrDecl>
    </attrGroup>
</response>
```

Failure:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <!-- This is the admin’s esurl -->
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/CreateVocab</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Could not create vocabulary</responseInfo>
</response>
```

RegisterService

The *RegisterService* operation adds the specification and description of a service in e-speak so that it can be found and used by others. The following example registers a service named “carDealer”, whose URL is http://www.car-dealer.com. It is also described in the vocabulary “car-dealer-simple” with corresponding attributes.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
          <to>es://localhost:12346/WebAccess/RegisterService</to>
          <from>es://localhost:12346\WAUser1</from>
          <context>
                <sessionToken>1234567890</sessionToken>
            </context>
        </communication>
</header>
```

Description and specification of the Service

```
<?xml version="1.0"?>
<resource xmlns="http://www.e-speak.net/Schema/E-speak.register.xsd
">
    <resourceSpec>
         <locator>http://www.car-dealer.com</locator>
    </resourceSpec>
    <resourceDes name=”carDealer”>
```

```
          <vocabulary>car-dealer-simple</vocabulary>
         <attr  name="Manufacturer">
               <value>Honda</value>
         </attr>
         <attr name="Model">
               <value>Accord</value>
        </attr>
    </resourceDes>
</resource>
```

Response

Success:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd">
    <communication>
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/RegisterService</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version="1.0"?>
<response status="succeeded">
    <resource
xmlns="http://www.e-speak.net/Schema/E-speak.register.xsd">
        <resourceSpec>

<srcAddress>es://localhost:12346/CarDealerService</srcAddress>
            <locator>http://www.cardealer.com</locator>
        </resourceSpec>
```

```
        <resourceDes>
            <attr name="Manufacturer">
                <value>Honda</value>
            </attr>
            <attr name="Model">
                <value>Accord</value>
            </attr>
            <attr name="Price">
                <value>25000.00</value>
            </attr>
        </resourceDes>
    </resource>
</response>
```

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd">
    <communication>
        <!-- This is the admin's esurl -->
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/RegisterService</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version="1.0"?>
<response status="failed">
    <responseInfo>Could not register service</responseInfo>
</response>
```

FindVocabulary

Vocabulary is a kind of resource, so finding vocabulary is the same as finding a service in WebAccess. Users can specify the reply contents of the lookup through <result>:

- $vocabulary -- reply includes the resource description and vocabulary attribute properties.
- $user -- reply includes the account description.
- $serviceinfo -- the resource description of the service.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
               <to>es://localhost:12346/WebAccess/FindService</to>
                <from>es://localhost:12346\WAUser1</from>
                <context>
                       <sessionToken>1234567890</sessionToken>
                </context>
       </communication>
</header>
```

Query for the Vocabulary

```
<?xml version="1.0"?>
<esquery xmlns=" http://www.e-speak.net/Schema/E-speak.query.xsd" >
    <from src="es://locahost:12346"/>
    <result> $vocabulary </result>
    <where>
         <!--  use the e-speak default vocabulary  -->
         <condition>Name="car-dealer-simple" and Type="Vocabulary"
         </condition>
    </where>
</esquery>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <!-- This is the admin’s esurl -->
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/FindVocabulary</from>

        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
    <attrGroup name="Simple Car dealer vocabulary”

xmlns=”http://www.e-speak.net/Schema/E-speak.vocab.xsd”>
         <attrDecl name="Manufacturer" required="true">
               <datatypeRef name="string"/>
         </attrDecl>
         <attrDecl name="Model" required="true">
               <datatypeRef name="string"/>
         </attrDecl>
         <attrDecl name="Price" required="false">
               <datatypeRef name="float">
                  <default>0.0</default>
                  <minInclusive>0.0</minInclusive>
                  <maxInclusive>100000.0</maxInclusive>
               </datatypeRef>
         </attrDecl>
```

```
    </attrGroup>
</response>
```

Failure:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <!-- This is the admin’s esurl -->
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/FindVocabulary</from>

        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Could not find vocabulary</responseInfo>
</response>
```

FindService

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
               <to>es://localhost:12346/WebAccess/FindService</to>
```

```
                <from>es://localhost:12346\WAUser1</from>
                <context>
                    <sessionToken>1234567890</sessionToken>
                </context>
        </communication>
</header>
```

Query of the Services

```
<?xml version="1.0"?>
<esquery xmlns=" http://www.e-speak.net/Schema/E-speak.query.xsd" >
    <from src="es://locahost:12346"/>
    <vocabulary name="cd" src="car-dealer-simple"/>
    <result> $serviceInfo </result>
    <where>
         <!--  find all dealer who has car made by Honda and price
less than 3000 -->
         <condition>cd:Manufactor = &apos;Honda&apos; and cd:Price
&lt; 3000.00
         </condition>
    </where>
    <preference>
       <!--  sort the results by their price, the cheaper the better
-->
         <operator>min</operator>
         <expr> cd:Price </expr>
    </preference>
    <arbitration>
         <!--  find the 3 cheapest -->
         <cardinality>3</cardinality>
    </arbitration>
</esquery>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/FindService</from>
    <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
   <!-- two services are found -->
    <resource
xmlns=”http://www.e-speak.net/Schema/E-speak.register.xsd”>
        <resourceSpec>

<srcAddress>es://localhost:12346/AutoDealerService</srcAddress>
            <locator>http://www.autodealer.com</locator>
        </resourceSpec>
        <resourceDes>
            <attr name=”Manufacturer”>
                <value>Toyota</value>
            </attr>
            <attr name=”Model”>
                <value>Camry</value>
            </attr>
            <attr name=”Price”>
                <value>24500.00</value>
            </attr>
```

```
        </resourceDes>
    </resource>
    <resource
xmlns="http://www.e-speak.net/Schema/E-speak.register.xsd">
        <resourceSpec>

<srcAddress>es://localhost:12346/CarDealerService</srcAddress>
            <locator>http://www.cardealer.com</locator>
        </resourceSpec>
        <resourceDes>
            <attr name="Manufacturer">
                <value>Honda</value>
            </attr>
            <attr name="Model">
                <value>Accord</value>
            </attr>
            <attr name="Price">
                <value>25000.00</value>
            </attr>
        </resourceDes>
    </resource>
</response>
```

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd">
    <communication>
        <to>es://localhost:12346/WebAccess\WAUser1</to>
        <from>es://localhost:12346/WebAccess/FindService</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Could not find service</responseInfo>
</response>
```

GetAccountProfile

Get the account profile:

- UserName, UserInfo, UserType, UserESURL
- Perferences(optional) .

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>

<to>es://localhost:12346/WebAccess/GetAccountProfile</to>
                <from>es://localhost:12346\WAUser1</from>
                <context>
                    <sessionToken>1234567890</sessionToken>
                </context>
       </communication>
</header>
```

Account Document

```
<?xml version="1.0"?>
<accountInfo
xmlns="http://www.e-speak.net/Schema/E-speak.account.xsd">
   <accountDes>
   <userName>WAUser1</userName>
   </accountDes>
```

```
   <authority>
      <userNamePass>
      <userName>esadmin</userName>
          <passPhrase>esadmin</passPhrase>
      </userNamePass>
   </authority>
</accountInfo>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <to>es://localhost:12346/WebAccess\WAUser1</to>

<from>es://localhost:12346/WebAccess/GetAccountProfile</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
    <accountInfo
xmlns=”http://www.e-speak.net/Schema/E-speak.account.xsd>
       <UserName>WAUser1</UserName>
       <userType>regular</userType>
       <userInfo />
       <userESURL>
es://localhost:12346/WebAccess\WAUser1<\UserESURL>
        <preference name=”length” value=”int”/>
```

```
        <preference name="color" value="red"/>
        <preference name="font" value="bold"/>
    </accountInfo>
</response>
```

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd">
    <communication>
        <to>es://localhost:12346/WebAccess\WAUser1</to>

<from>es://localhost:12346/WebAccess/GetAccountProfile</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version="1.0"?>
<response status="failed">
    <responseInfo>Could not get the user profile</responseInfo>
</response>
```

SetAccountProfile

Set the profile for an account:

- Account description (UserName, UserInfo, UserType, UserESURL, User-Phrase). (all optional)
- Perferences (optional)

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>

<to>es://localhost:12346/WebAccess/SetAccountProfile</to>
                 <from>es://localhost:12346\WAUser1</from>
                 <context>
                      <sessionToken>1234567890</sessionToken>
                </context>
       </communication>
</header>
```

Account Document

```
<?xml version="1.0"?>
<accountInfo
xmlns="http://www.e-speak.net/Schema/E-speak.account.xsd">
   <accountDes>
   <UserName>WAUser1</UserName>
   <UserType>regular</UserType>
</accountDes>

   <authority>
      <userNamePass>
      <userName>esadmin</userName>
          <passPhrase>esadmin</passPhrase>
```

```
      </userNamePass>
   </authority>
<preference name="length" value="int" />
   <preference name="color" value="red" />
   <preference name="font" value="bold" />
</accountInfo>
```

Response

Success:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <to>es://localhost:12346/WebAccess\WAUser1</to>

<from>es://localhost:12346/WebAccess/SetAccountProfile</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
</response>
```

Failure:

Header Document

```
<?xml version=”1.0”?>
<header xmlns=”http://www.e-speak.net/Schema/E-speak.header.xsd”>
    <communication>
        <to>es://localhost:12346/WebAccess\WAUser1</to>
```

```
<from>es://localhost:12346/WebAccess/SetAccountProfile</from>
        <context>
            <sessionToken>1234567890</sessionToken>
        </context>
    </communication>
</header>
```

Response Document

```
<?xml version="1.0"?>
<response status="failed">
    <responseInfo>Could not set the account profile</responseInfo>
</response>
```

Poll Messages

In the following example, *WAUser1* (specified) by the <from> in the Header Document polls the E-speak persistent message queue to find out if there are messages for him.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346/WebAccess/PmqPoll</to>
             <from>es://localhost:12346\WAUser1</from>
             <context>
                  <sessionToken>1234567890</sessionToken>
             </context>
       </communication>
</header>
```

Response

Success:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346\WAUser1</to>
             <from>es://localhost:12346/ WebAccess/PmqPoll</from>
             <context>
                  <sessionToken>1234567890</sessionToken>
             </context>
       </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
   <messageList>
        <message>
            <messageId>123</messageId>
            <from>es://localhost:12346\WAUser1</from>
        </message>
        <message>
            <messageId>136</messageId>
            <from> es://localhost:12346\WAUser2</from>
        </message>
    <messageList>
</response>
```

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
```

```
            <to>es://localhost:12346\WAUser1</to>
            <from>es://localhost:12346/ WebAccess/PmqPoll</from>
            <context>
                 <sessionToken>1234567890</sessionToken>
            </context>
       </communication>
</header>
```

Response Document for Zero Messages

```
<?xml version="1.0"?>
<response status="succeeded">
    <responseInfo>There are no messages</responseInfo>
      </responseInfo>
```

Response Document for Failure

```
<?xml version="1.0"?>
<response status="failed">
    <responseInfo>Could not get messages.</responseInfo>
</response>
```

Get Message

In the following example, *WAUser1* (specified) by the <from> in the Header Document tries to get messages from the E-speak persistent message queue.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
            <to>es://localhost:12346/WebAccess/PmqGet</to>
            <from>es://localhost:12346\WAUser1</from>
            <context>
                 <sessionToken>1234567890</sessionToken>
```

```
            </context>
            <messageInfo>
                <messageId>123</messageId>
            </messageInfo>
       </communication>
</header>
```

Response

Success:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
            <to>es://localhost:12346\WAUser1</to>
            <from>es://localhost:12346/ WebAccess/PmqGet</from>
            <context>
                 <sessionToken>1234567890</sessionToken>
            </context>
       </communication>
</header>
```

Response Document

The response document is the actual message sent.

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
            <to>es://localhost:12346\WAUser1</to>
            <from>es://localhost:12346/ WebAccess/PmqGet</from>
            <context>
                 <sessionToken>1234567890</sessionToken>
```

```
            </context>
       </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Could not get message.</responseInfo>
</response>
```

Delete Messages

In the following example, *WAUser1* (specified) by the <from> in the header document deletes messages for him in the E-speak persistent message queue.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
            <to>es://localhost:12346/WebAccess/PmqDelete</to>
            <from>es://localhost:12346\WAUser1</from>
            <context>
                 <sessionToken>1234567890</sessionToken>
            </context>
            <messageInfo>
                <messageId>123</messageId>
           </messageInfo>
       </communication>
</header>
```

Response

Success:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346\WAUser1</to>
             <from>es://localhost:12346/ WebAccess/PmqDelete</from>
             <context>
                  <sessionToken>1234567890</sessionToken>
             </context>
       </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
</response>
```

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346\WAUser1</to>
             <from>es://localhost:12346/ WebAccess/PmqDelete</from>
             <context>
                  <sessionToken>1234567890</sessionToken>
             </context>
       </communication>
</header>
```

Response Document

```
<?xml version="1.0"?>
<response status="failed">
    <responseInfo>Could not delete message.</responseInfo>
</response>
```

Asynchronous Messages

In the following example, WAUser1 sends an asynchronous message to WAUser2.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346\WAUser2</to>
             <from>es://localhost:12346\WAUser1</from>
             <context>
                  <sessionToken>1234567890</sessionToken>
             </context>
             <messageInfo synchronous=”false”
                                  timeout=”600”
                                  pull=”true”
                                  persistent=”true”>
           </messageInfo>
       </communication>
</header>
```

Response

Success:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346\WAUser1</to>
             <from></from>
             <context>
                  <sessionToken>1234567890</sessionToken>
```

```
            </context>
       </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
</response>
```

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
            <to>es://localhost:12346\WAUser1</to>
            <from></from>
            <context>
                 <sessionToken>1234567890</sessionToken>
            </context>
       </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Could not queue the message.</responseInfo>
</response>
```

Synchronous Messages

In the following example, WAUser1 sends a synchronous message to WAUser2.

Request

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346\WAUser2</to>
             <from>es://localhost:12346\WAUser1</from>
             <context>
                  <sessionToken>1234567890</sessionToken>
             </context>
             <messageInfo retries=”3”
                                retryInterval=”300”
                                synchronous=”true”
                                pull=”false”
                                persistent=”false”>
           </messageInfo>
       </communication>
</header>
```

Response

The response is mainly concerned with the message delivery to the client/service

Success:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
             <to>es://localhost:12346\WAUser1</to>
```

```
            <from> es://localhost:12346\WAUser2</from>
            <context>
                 <sessionToken>1234567890</sessionToken>
            </context>
            <messageInfo>
                  <messageId>123</messageId>
            </messageInfo>
       </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”succeeded”>
</response>
```

Failure:

Header Document

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd" >
   <communication>
            <to>es://localhost:12346\WAUser2</to>
            <from> es://localhost:12346\WAUser1</from>
            <context>
                 <sessionToken>1234567890</sessionToken>
            </context>
            <messageInfo retries=”3”
                                retryInterval=”300”
                                synchronous=”true”
                                pull=”false”
                                persistent=”false”>
                <messageId>123</messageId>
                <sentTime>2000-04-21 14:30:00</sentTime>
            </messageInfo>
       </communication>
</header>
```

Response Document

```
<?xml version=”1.0”?>
<response status=”failed”>
    <responseInfo>Could not send the message.</responseInfo>
</response>
```

Chapter 6 WebAccess Clients and Services

WebAccess Clients and Services are different from e-speak JESI clients and services. WebAccess clients and services use HTTP to connect to WebAccess or to each other.

WebAccess Service

Any WebAccess Service is a servlet. This section describes writing some basic servlets that can communicate with WebAccess.

The sample service is a servlet that extends the HttpServlet class. The request to the service (servlet) can consist of any of the following content types: text/xml or multipart/form-data. The request will always come to the service as a FORM POST. When the service receives the request, it reads the data from its input stream, then parses it to get a vector of XML documents. Then the vector can be used to get at various documents sent to the service. The service then writes to the output stream of the response like any other servlet. Again, the response content type could be text/XML or multipart/form-data.

The following code is a sample service that, when invoked, determines the kind of request that was sent to it. Then, it responds in the same content type as the request.

```
import net.espeak.webaccess.contransform.ContentException;
import net.espeak.webaccess.Util;
import net.espeak.webaccess.WAConstants;
import net.espeak.webaccess.util.infra.AccessXml;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class XmlService extends HttpServlet {
```

```
AccessXml hdrXml = null;

    /**
     * Initializes variables at the servlet level and starts the ServiceDaemon
     * @param  config  the configuration to be used for initialization
     */
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        _config = config;
    }

    /**
     * HTTP post request come to this method.
     *
     * @param      req    the request in the form of HTTP
     * @param      res    the response in the form of HTTP
     * @exception  IOException       when an input/output error occurs
     * @exception  ServletException  when an error occurs with the servlet
     */
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        try {
            boolean bMime = false;
            Vector domVec = null;
            try {
                if(req.getContentType().equals("text/xml")) {
                    bMime = false;
                }
                else {
                    bMime = true;
                }
                ServletInputStream sis = req.getInputStream();
                byte[] data = new byte[sis.available()];
                sis.read(data);
                domVec = Util.splitDocument(new String(data));
                hdrXml = new AccessXml(Util.getESpeakHeader(domVec));
            } catch(IOException e) {
                Util.printStackTrace(e);
               throw new ContentException("exception while reading XML from HTTP
request");
            }
            if(bMime) {
                sendMimeResponse(res, domVec);
            }
            else {
                sendXmlResponse(res, domVec);
            }
        } catch (Exception e) {
            res.setContentType("text/html");
            PrintWriter out = res.getWriter();

            out.println("<h3>" + e + "</h3>");
            out.println("</BODY></HTML>");
```

```
    }
}

/**
 * HTTP get method comes to this method. This method can be used to check
 * whether the servlet is properly configured with the webserver
 *
 * @param      req    the request in the form of HTTP
 * @param      res    the response in the form of HTTP
 * @exception  IOException       when an input/output error occurs
 * @exception  ServletException  when an error occurs with the servlet
 */
public void doGet(HttpServletRequest req,
    HttpServletResponse res) throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<HTML>");
    out.println("<HEAD><TITLE>Xml Service</TITLE></HEAD>");
    out.println("<BODY bgcolor=\"EEFBEE\">");
    out.println("<CENTER> XmlService Is Alive </CENTER>");
    out.println("<CENTER>" + (new Date()) + "</CENTER>");
    out.println("</BODY></HTML>");
}

/**
 * Gets information about the servlet.
 *
 * @return String     the servlet information as a string
 */
public String getServletInfo() {
    return "WebAccess Service";
}

/*
 * Sends a multipart/form-data type response
 *
 * @param res - The HttpResponse object to which to send output
 * @param domVec - the vector of DOMs sent in the request
 * @exception Exception - Throws an exception if anything goes wrong
 */
private void sendMimeResponse(HttpServletResponse res, Vector domVec)
    throws Exception {
    String boundary = Util.createMimeBoundary();
    res.setContentType("multipart/form-data; boundary=" + boundary);
    boundary = "--" + boundary;
    // Create a response header with the names of the attachments
    AccessXml respHdrXml = (AccessXml)Util.createHeader();
    respHdrXml.setValue(WAConstants.HDR_COMM_FROM,
                        hdrXml.getValue(WAConstants.HDR_COMM_TO));
  respHdrXml.setValue(WAConstants.HDR_COMM_CONTEXT_TOKEN,
                      hdrXml.getValue(WAConstants.HDR_COMM_CONTEXT_TOKEN));
    int idx = 0;
```

```
       respHdrXml.setAttribute("header/attachment[" + idx++ + "]/name", "header");
        respHdrXml.setAttribute("header/attachment[" + idx++ + "]/name",
"response");
        respHdrXml.setAttribute("header/attachment[" + idx++ + "]/name",
"myResponse");
        PrintWriter out = res.getWriter();
        out.println(boundary);
        out.println("Content-Disposition: name=header");
        out.println("Content-Type: text/xml");
        out.println();
        // Send a ESpeak header
        out.println(respHdrXml.getText());
        out.println(boundary);
        out.println("Content-Disposition: name=response");
        out.println("Content-Type: text/xml");
        out.println();
        // Send a ESpeak response.
        writeESpeakResponse(out);
        out.println(boundary);
        out.println("Content-Disposition: name=myResponse");
        out.println("Content-Type: text/xml");
        out.println();
        // Send any other response
        writeResponse(out, domVec);
        out.println(boundary);
    }

    /*
     * Sends a text/xml type response
     *
     * @param res - The HttpResponse object to which to send output
     * @param domVec - the vector of DOMs sent in the request
     * @exception Exception - Throws an exception if anything goes wrong
     */
    private void sendXmlResponse(HttpServletResponse res, Vector domVec)
        throws Exception {
        res.setContentType("text/xml");
        PrintWriter out = res.getWriter();
        // Send a ESpeak header
        writeESpeakHeader(out);
        // Send a ESpeak response.
        writeESpeakResponse(out);
        // Send any other response
        writeResponse(out, domVec);
    }

    /*
     * This method writes a ESpeak header to the output stream
     */
    private void writeESpeakHeader(PrintWriter out)
        throws Exception {
        out.println("<?xml version=\"1.0\"?>");
```

```
        out.println("<header xmlns=\"http://www.e-speak.net/Schema/E-
speak.header.xsd\">");
        out.println("<communication>");
        out.print("<to>");
        out.print(hdrXml.getValue(WAConstants.HDR_COMM_FROM));
        out.println("</to>");
        out.println("<from>http://localhost/servlets/XmlService</from>");
        out.println("<context>");
        out.println("<sessionToken>");
        out.println(hdrXml.getValue(WAConstants.HDR_COMM_CONTEXT_TOKEN));
        out.println("</sessionToken>");
        out.println("</context>");
        out.println("</communication>");
        out.println("</header>");
        out.println();
    }

    /*
     * This method writes a ESpeak response to the output stream
     */
    private void writeESpeakResponse(PrintWriter out)
        throws Exception {
        out.println("<?xml version=\"1.0\"?>");
        out.println("<response status=\"succeeded\" “ +
               “xmlns=\"http://www.e-speak.net/Schema/E-speak.response.xsd\">");
        out.println("<responseInfo>");
        out.print("Hello ");
        out.println(hdrXml.getValue(WAConstants.HDR_COMM_FROM));
        out.println("</responseInfo>");
        out.println("</response>");
        out.println();
    }

    /*
     * This method writes any other response to the output stream
     */
    private void writeResponse(PrintWriter out, Vector domVec)
        throws Exception {
        out.println("<?xml version=\"1.0\"?>");
        out.println("<myResponse xmlns=\"http://www.mycompany.com/schemas/
my.xsd\">");
        out.println("<info>");
        out.print("Hello ");
        out.println(hdrXml.getValue(WAConstants.HDR_COMM_FROM));
        out.println("</info>");
        out.println("</myResponse>");
        out.println();
    }
}
```

This servlet can be configured with a webserver as any other servlet.

WebAccess XML Client

There are two types of clients that can access WebAccess and any service connected to WebAccess. These are: the browser and the XML client. The browser is described in another chapter. This section describes only the XML client. Both of the clients send XML documents conforming to the E-Speak schemas (See Appendix). The only difference between the two is that the browser client provides a simple presentation layer to the user and hides the complexity of creating XML requests.

The XML client opens a socket connection to the WebServer running the WebAccess servlet, then sends requests on that socket. The first request any client should send is login. After login, any request can be sent as long as it has an E-Speak header. The login response returns a "Session Token" that is very important for any future requests. This token carries the timeout information and if this token is not sent then the client may be timed out.

```
import net.espeak.webaccess.Util;
import net.espeak.webaccess.WAConstants;
import net.espeak.webaccess.util.infra.AccessXml;
import java.net.MalformedURLException;
import java.net.Socket;
import java.net.SocketException;
import java.net.URL;
import java.net.UnknownHostException;
import java.io.BufferedReader;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.StringReader;
import java.util.Enumeration;
import java.util.Vector;
import org.w3c.dom.Document;

public class XmlClient {

    private boolean bDebug = false;
    private boolean bMime = false;
    private String xmlDir = "";
    private String hostname = "localhost";
    private int port = 80;
    private boolean bHtmlResp = false;
    private boolean bMimeresp = false;
```

```
/**
 * This method sets the directory where to find Xml message files
 * @param dir - directory containing Xml files
 */
public void setXmlDir(String dir) {
    xmlDir = dir;
}

/**
 * This method returns the Xml directory
 * @return the Xml directory
 */
public String getXmlDir() {
    return xmlDir;
}

/**
 * This method sets the debugging flag.
 * @param debug - the value for the debugging flag
 */
public void setDebugOn(boolean debug) {
    bDebug = debug;
}

/**
 * This method returns true if debugging is turned on or false otherwise.
 * @return the value of the debugging flag
 */
public boolean isDebugOn() {
    return bDebug;
}

/**
 * This method sets the flag to use MIME format for requests.
 * @param mimeFlag - the calue of the mime flag
 */
public void setMime(boolean mimeFlag) {
    bMime = mimeFlag;
}

/**
 * This method returns whether the mime formatting is turned on.
 * @return true if MIME is turned on, false otherwise
 */
public boolean isMimeOn() {
    return bMime;
}

/**
 * This method setst he hostname to use to make connection.
 * @param host - the hostname where the WebServer/WebAccess is running.
 */
public void setHostname(String host) {
```

```
    hostname = host;
}

/**
 * This method returns the hostname of the WebServer/WebAccess.
 * @return hostname
 */
public String getHostname() {
    return hostname;
}

/**
 * This method sets the port number of the WebServer.
 * @param portNumber - the port number of the Webserver
 */
public void setPort(int portNumber) {
    port = portNumber;
}

/**
 * This method returns the port number of the Webserver.
 * @return port
 */
public int getPort() {
    return port;
}

/**
 * This method processes the request and returns the response.
 * It connects to the WebAccess server and then sends the request.
 * The response is stripped off of the Http headers and returned.
 * @param requestXml - the xml request to be processed.
 * @return the response from the request
 */
public String processRequest(String requestXml) {
    return sendReq(packReq(requestXml));
}

/**
 * The main entry point for XML client.
 *
 * @param args   Array of parameters passed to the application
 *               via the command line.
 */
public static void main (String[] args) {
    String sToken = "";

    XmlClient xmlClient = new XmlClient();
    try {
        int i, idx;
        // If no arguments supplied, print usage
        if(args.length == 0) {
            usage();
```

```
    System.exit(0);
}
// debug option
idx = getOptionIndex(args, "-g");
if (idx != -1)
    xmlClient.setDebugOn(true);

// help option
idx = getOptionIndex(args, "-h");
if (idx != -1) {
    usage();
    System.exit(0);
}

// mime option
idx = getOptionIndex(args, "-mime");
if (idx != -1)
    xmlClient.setMime(true);

// hostname
idx = getOptionIndex(args, "-host");
if (idx != -1)
    xmlClient.setHostname(args[idx + 1]);

// port
idx = getOptionIndex(args, "-port");
if (idx != -1) {
    try {
        xmlClient.setPort(Integer.parseInt(args[idx + 1]));
    }
    catch(NumberFormatException e) {
        System.out.println("argument for port should be a number");
        usage();
        System.exit(0);
    }
}


// Determine where the XML files are.
idx = getOptionIndex(args, "-d");
if (idx != -1) {
    // Change any backslashes to forward slashes
    String str = args[idx+1].replace('\\', '/');
    xmlClient.setXmlDir(str + "/");
}

//  URLs param
idx = getOptionIndex(args, "-url");
if (idx == -1) {
    System.out.println("No URL parm, use '-h' for help! ");
    usage();
    System.exit(0);
}
```

```
int numOfUrls = 0;
try {
    numOfUrls = Integer.parseInt(args[idx + 1]);
}
catch(NumberFormatException e) {
    System.out.println("argument for -url should be a number");
    usage();
    System.exit(0);
}
if(numOfUrls < 1) {
    System.out.println("Atleast one url should be specified");
    usage();
    System.exit(0);
}
URL[] url = new URL[numOfUrls];
// last URL index in the args
int last = idx + numOfUrls + 2;

//Create an array of URLs from the command line args
int numFiles = 0;
for (int k = idx + 2; (k < last ); k++) {
    url[numFiles++] = new URL("file://localhost/" +
                              xmlClient.getXmlDir() + args[k]);
} // for loop
// Process each XML file, one by one
// Get the login operation name
String loginOp = WAConstants.getOperationName(WAConstants.LOGIN);
for (int j = 0; j < numFiles ; j++) {
    try {
        // Read the request into a buffer and then create DOM trees
        // from the request. From these DOM trees get the operation
        // being performed by the request, If the operation
        String filename =  new String (url[j].getFile());
        File myFile = new File(filename);
   BufferedReader reader = new BufferedReader(new FileReader(myFile));
        String msg;
        Vector docs;

        char[] chars = new char[(int)myFile.length()];
        reader.read(chars);
        String singleDoc = new String(chars);
        docs = Util.splitDocument(singleDoc);
     AccessXml reqHdrXml = new AccessXml(Util.getESpeakHeader(docs));

        String reqTo = reqHdrXml.getValue(WAConstants.HDR_COMM_TO);
        if (reqTo != null) {
            String op = Util.getOperation(reqTo);
            if (op == null || !(op.equals(loginOp))) {
                // insert the sessionToken
                reqHdrXml.setValue(WAConstants.HDR_COMM_CONTEXT_TOKEN,
                        sToken);
                docs.setElementAt(reqHdrXml.getDocument(), 0);
            }
```

```
                    }

                    String requestXml = Util.requestToString(docs,
xmlClient.isMimeOn());

                    // Prepare HTTP Post request for this xml
                    String response = xmlClient.processRequest(requestXml);

                    AccessXml respHdrXml = null;
                    AccessXml respXml = null;
                    // response is already stripped off of HTTP header
                    // find out the result
                    if ((response == null) ||
                        (response.indexOf("HTTP Server Error" ) >= 0)) {
                        System.out.println("HTTP Server Error: " + response);
                        System.exit(0);
                    } else {
                        if (!((response.indexOf("<?xml" ) >= 0) ||
                              (response.indexOf("<html>" ) >= 0))) {
                            System.out.println(filename + " Failed");
                            System.exit(0);
                        } else {
                           System.out.println("HTTP Response for " + filename + "
begins: \n");
                            // response string has XML returned
                            try {
                                docs = Util.splitDocument(response);

                                // Print HTTP/XML reply regardless
                                // of success/failure
                                String str = "";
                                Enumeration enum = docs.elements();
                                int x = 1;
                                while(enum.hasMoreElements()) {
                                    Document doc =
                                              (Document)enum.nextElement();
                                    System.out.println("-- Document "
                                                       + x++ + " -----------------");
                                 System.out.println(AccessXml.getText(doc, true));
                            System.out.println("-- Document Ends --------------");
                                }
                                if(Util.isESpeakDocument(docs)) {
                                    respHdrXml = new
AccessXml(Util.getESpeakHeader(docs));
                            respXml = new AccessXml(Util.getESpeakDocument(docs));
                                }
                            }
                            catch (Exception e) {
                                System.out.println("response = "+response);
                                throw e;
                            }
```

```
                            System.out.println("HTTP Response for " + filename + "
ends. ");
                             if(respXml != null) {
                                 String status =
respXml.getAttribute(WAConstants.RESPONSE_STATUS);
                                 if (status == null ||
!(status.equals(WAConstants.REQ_SUCCEEDED))) {
                                     System.out.println(filename + " Failed.");
                                 }
                                 else {
                                     System.out.println(filename + " Passed.\n");
                                 }
                             } else {
                                 System.out.println(filename + " Passed.\n");
                             }
                         }
                     }

                     // Prepare for the next XML file.
                     // save the sessionToken for the next XML file
                     if(respHdrXml != null) {
                         String tempStr = respHdrXml.getValue(
                                          WAConstants.HDR_COMM_CONTEXT_TOKEN);
                         if (tempStr == null) {
                             System.out.println("sessionToken not returned, using
old one.");
                         } else
                             // In this case, I'd use a saved one, non-null
                             sToken = tempStr;
                     } else {
                    System.out.println("sessionToken not returned, using old one.");
                     }

                 } catch (Exception e) {
                     e.printStackTrace();
                 }
             } // for

         } catch (MalformedURLException e) {
             e.printStackTrace();
         } catch (Exception e) {
             e.printStackTrace();
         }
     }

     /**
      * Gets the argument corresponding to the passed option
      *
      * @param   args the arguments to parse for the option
      * @param   opt  the option to look for.
      *
      * @return  int  the position of the option in the args array.
      */
```

```
static int getOptionIndex(String[] args, String opt) {
    for (int i=0; i < args.length; i++) {
        if (args[i].equalsIgnoreCase(opt))
            return i;
    }
    return -1;
}

/**
 * Packages an xml or mime document (string) into the body of
 * a HTTP request.
 * @param   xml     xml document provided as string
 * @return  String  the string request containing HTTP POST header
 * and document.
 */
private String packReq( String xml) {
    String req  = "POST /servlets/WebAccess HTTP/1.0\r\n";
    // insert content-type based on the  document type
    if (isMimeOn()) {
        try {
            req += "Content-Type: multipart/form-data; boundary="+
                Util.getBoundary(xml)+"\r\n";
        }
        catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    } else {
        req += "Content-Type: application/xml-text\r\n";
    }
    req += "User-Agent: Mozilla/4.0\r\n";
    req += "Host: " + getHostname() + "\r\n";
    req += "Content-Length: " + xml.length() + "\r\n\r\n";
    req += xml;
    if (isDebugOn()) {
      System.out.println("\n\nRequest ----------------------------------------");
        try {
            System.out.println("HTTP Request: ----------------------");
            BufferedReader br = new BufferedReader(new StringReader(req));
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("----------------------------------------\n\n");
    }
    return req;
}

/**
 * Sends a packaged HTTP request out
```

```
 *
 * @param   req      HTTP/XML POST request to be sented host/port.
 * @return  String   the string response, stripped off of HTTP header.
 *                   may contain successful reply, or
 *                   error/exception string.
 */
private String sendReq( String req) {
    try {
        Socket ss = new Socket(getHostname(), getPort());
        ss.setTcpNoDelay(true);

        BufferedReader br = new BufferedReader(
                            new InputStreamReader(ss.getInputStream()));
        BufferedOutputStream dout =
        new BufferedOutputStream(ss.getOutputStream());
        dout.write(req.getBytes());
        dout.flush();

        String line = "";
        String respDoc = "";
        String data = "";

        boolean resStart = false;
        boolean mimeResp = false;

        while ((line = br.readLine()) != null) {
            if(line.startsWith("Content-Type: multipart/form-data")) {
                // response is MIME
                mimeResp = true;
            }

            if (isMimeOn() || mimeResp) {
                if (line.indexOf("--") == 0)
                    // Skip comments
                    if(line.indexOf("-->") != 0)
                        resStart = true;
            }
            else {
                if (line.indexOf("<?xml") >= 0)
                    resStart = true;
            }

            // When reply is from e-speak, create a repsonse string
            if (resStart) {
                respDoc += line;
                if (isMimeOn() || mimeResp) {
                    respDoc += "\r\n";
                }
            }
            else {
                data += line;
                data += "\n";
            }
```

```
                if (isDebugOn())
                    System.out.println("Reply = " + line);
            }

            if (!resStart) {
                respDoc += "status=\"failed\" HTTP Server Error";
                respDoc += "Didn't find expected output! ";
                respDoc += "\n---- Actually got this ----\n"+data;
            }

            br.close();
            dout.close();
            ss.close();

            if (isDebugOn()) {
                System.out.println("\n\nResponse " + "------------------------------
---------");
                System.out.println(respDoc);
              System.out.println("----------------------------------------\n\n");
            }
            return respDoc;

        } catch ( SocketException se) {
            se.printStackTrace();
        } catch ( UnknownHostException ue) {
            ue.printStackTrace();
        } catch ( IOException ie) {
            ie.printStackTrace();
        }
        return "\n\nEXCEPTION From SendRequest of XmlClient\n\n";
    }

    /**
     * Prints usage, called from several places.
     */
    static void usage() {
        System.out.println("usage:  ");
        System.out.println("java XmlClient <options> <filesInfo>");
        System.out.println(" <options> ");
        System.out.println("    -h    this help text  ");
        System.out.println("    -g    debug on, default is off ");
        System.out.println("    -mime sends request as multipart,");
        System.out.println("          default is off ");
        System.out.println("    -host Host where webaccess is running,");
        System.out.println("          default is localhost");
        System.out.println("    -port Port number of the webserver,");
        System.out.println("          default is 80");
        System.out.println("    -d    actual path where xml directory is,");
        System.out.println("          default is current \n");
        System.out.println(" <filesInfo>     -url <num_of_files>"
                           + "<actual_files> ");
        System.out.println(" <num_of_files>  integer");
        System.out.println(" <actual_files>  each file");
```

```
        System.out.println("\nExamples: \n");
        System.out.println("  2 XML files for XmlClient,");
        System.out.println("  ---- when using default directory home. \n");
        System.out.print("       java XmlClient " +
                  "-url 2 login.xml lookup.xml " + "\n\n");
        System.out.println("  ---- when giving absolute directory home. \n");
        System.out.print("       java XmlClient " +
                  "-d d:\\mydir -url 2 login.xml vocab.xml" + "\n\n");

    }
}
```

To print the usage, type

```
java XmlClient –h
```

To run the client with some Xml files from the directory, c:\myxml:

```
java Xmlclient -d c:\myxml url -3 login.xml sendrequest.xml logout.xml
```

To run the client using mime format:

```
java Xmlclient -d c:\myxml -mime url -3 login.xml sendrequest.xml logout.xml
```

Sample Requests for the Client

Login XML:

```
<?xml version='1.0'?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd">
  <communication>
    <to>es://localhost:12346/WebAccess/Login</to>
  </communication>
</header>
<?xml version='1.0'?>
<accountInfo xmlns="http://www.e-speak.net/Schema/E-speak.account.xsd">
  <userNamePass>
    <userName>esadmin</userName>
    <passPhrase>esadmin</passPhrase>
  </userNamePass>
</accountInfo>
```

Service Invocation XML:

```
<?xml version="1.0"?>
<header xmlns="http://www.e-speak.net/Schema/E-speak.header.xsd">
  <communication>
    <to>http://localhost/servlets/XmlService</to>
  </communication>
</header>

<?xml version="1.0"?>
<myRequest xmlns="http://www.mycompany.com/schemas/my.xsd">
  <req>hello</req>
</myRequest>
```

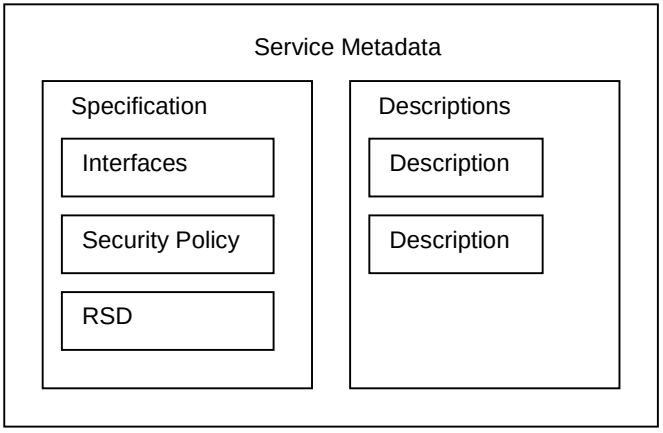
Chapter 7 WebAccess Service Registration and Lookup

This chapter contains descriptions on how you register services as a service provider and how you discover services through WebAccess as a user. In particular, it describes the framework of the WebAccess service registration and lookup and explains how you register your service using XML and how you specify a query in XML. Also, it provides brief account on the notion of vocabulary in e-speak.

The Framework

Registration

Services are described by specification and descriptions in e-speak. We refer to them as metadata. Service specification has information about how a service should be accessed and managed while service descriptions define the way a service is presented to clients. Clients find a service by using its descriptions. On the other hand, the e-speak engine mediates her access to the service by using the specification.



Service Specification

E-spek service specification consists of interfaces (called contract), security policy, a filter, and other resource specific data (RSD). Interface specification prescribes how customers interact with the service. Access control for services is specified in the security policy while visibility is specified with the filter. RSD can have any data pertinent to the service. They can be public and visible to other customers or they can be private and only the owner can access them.

The current release of WebAccess supports a subset of the specification: only filter and RSD are allowed. In addition, WebAccess allows a service provider to specify the real URL of the service as part of its specification.

Service Descriptions

E-speak allows services to have multiple descriptions. Each description is a set of attribute name value pairs. To avoid name collision and to facilitate validation, a description is required to be specified in a specific vocabulary. For more detailed description on vocabulary, see the section about Vocabulary in this chapter.

Lookup

In e-speak users specify in a query requirements on the services they are looking for; a user discovers services by constructing a query and presenting it to Web e-speak. A typcal query consists of vocabulary declarations, a constraint, zero or more preferences, and an arbitration policy. Vocabulary declarations define a mapping from local names to vocabularies.The local names are used in the constraint and the preferences to distinguish attributes from different vocabularies. The e-speak engine evaluates the constraint to determine the set of matching services. Preferences are used to order the lookup results and arbitration policy is used to limit the number of results to be returned.

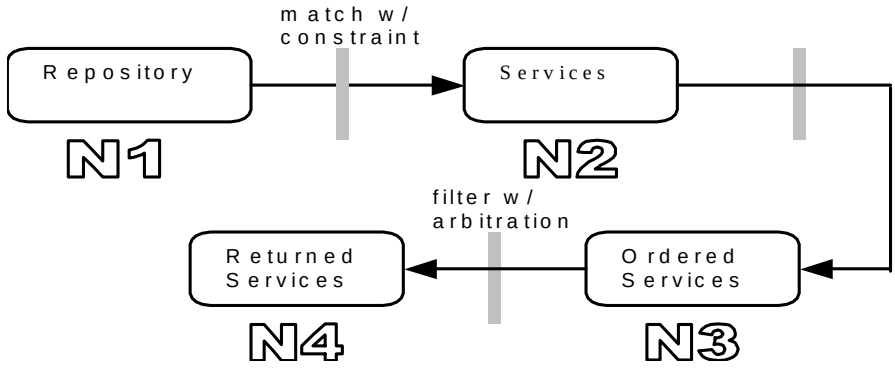


Figure 39 A pipeline diagram for the e-speak lookup process

Figure 1 is a pipeline diagram of the lookup process in the e-speak engine. When a query is given, the engine applies the constraint to the repository and generates N2. If preferences are given the engine apply them to N2. If the user specified an arbitration policy it is applied to N3 to produce N4. If no preference was given, then N2 = N3. If no arbitration policy is given, then N3 = N4

Vocabulary Declaration

A vocabulary declaration associates a local reference to a vocabulary. The local reference is used in a condition to disambiguate attributes from different vocabularies. Local references are simple string names. E-speak allows actual vocabularies to be referred toin three different ways: using a vocabulary name with which a vocabulary is registered with the e-speak engine, using another e-speak lookup query, or using an URI which points to the schema definition of a vocabulary. For example, the following declarations associates "seller" with a "PC seller" vocabulary and "box" with a vocabulary which is defined with an XML Schema located at `http://www.parcel.com/box.xsd`.

```
vd seller="PC seller"
vd box="http://www.parcel.com/box.xsd"
```

The current release of WebAccess supports vocabulary specifications with vocabulary names only.

Constraint

Constraint is a predicate over attributes that can be defined in different vocabularies. To distinguish the attributes, attribute names are qualified with local vocabulary references. For example the following query returns a list of PC manufacturers who sell their products to `ResellerA` and buy CPUs from `CPUMakerB`.

```
vd seller="PC seller"
vd buyer="CPU buyer"
constraint seller:customer="ResellerA" and
           buyer:supplier="CPUMakerB"
```

Identifiers in the constraint such as `customer` and `supplier` refer to attributes in the corresponding vocabularies specified by their prefixes such as `seller` and `buyer`.

Preference

Once the e-speak engine finds services that match the lookup constraint, it uses preferences to order services. E-speak defines three preference operators: `with`, `min`, and `max`. A lookup query can have any combination of preference expressions.

The `with` operator takes a condition and a weight expression. The condition is a predicate over attributes while any numeric expression can be used for the latter. When given multiple `with` preferences, the e-speak engine evaluates the condition of a `with` preference for each service that matches the constraint. The associated weight is added to the total weight when the condition is satisfied. After all evaluations have been completed, the matching services are sorted in the descending order of their total weight. For example, the following query looking for blue boxes also specifies that the user prefers boxes whose length is less than 5 and whose height is greater than 7, but she prefers taller ones to shorter ones.

```
vd box=”http://www.parcel.com/box.xsd”
constraint box:color=”blue”
with box:length < 5 weight 2
with box:height >7 weight 4
```

The `min` takes an expression and orders services in the ascending order of the value of the expression. The `max` attribute is the same as `min` except that services are ordered in the descending order. For example, if you prefer boxes with bigger volume, you can specify the preference as follows:

```
vd box="http://www.parcel.com/box.xsd"
constraint box:color=”blue”
max box:length * box:width * box:height
```

If multiple `max/min` preferences are specified in a query, the order that the preferences appear in the query becomes significant. The first preference provides the primary order, the second preference is used as a tiebreaker for the result of the first, and so on. The `with` preferences and the `min/max` preferences can also be specified together, but the `with` preferences are given higher priority. Thus the result is sorted according to the `with` preferences first. Then the first `min/max` preference is used to break ties, the second is used to break ties from the first, and so on.

Arbitration Policy

Arbitration policy limits the number of services that are returned to the user. It is similar to the return cardinality policy in OMG TOS [1]. Three operators are supported: `first`, `all`, and `any`. The `first` operator takes an integer argument so

that 'first n' instructs the e-speak engine to return at most n services. The all and any do not take arguments and make the engine return all services or any service it finds. The any operator can be used to implement transparent load balancing.

E-Speak Query Language in XML

To find a service through WebAccess, users need to construct their queries in XML and send the XML document to WebAccess. In this section we describe how to build e-speak queris in XML. The esquery element defines the structure of e-speak queries. We describe some of frequently used query constructs in this section. For the complete schema definition, see Appendix A

The Overall Query Structure

A typical e-speak XML query is composed of vocabulary declarations followed by a result specification followed by a constraiont, preferences, and arbitration.

```
<esquery>
   <vocabulary> ... <vocabulary>
   <result> ... </result>
   <where> ... </where>
   <preference> ... </preference>
   <arbitration> ... </arbitration>
</esquery>
```

When the query is delivered to WebAccess it forwards it to the e-speak Core that it is connected to. The core looks up the repository for services whose descriptions satisfy the constraint. Preference and arbitration are used to sort and filter the results. Note that the preference and the arbitration element can be omitted. With the search results, the core constructs the return result according to the specification in the result element.

Vocabulary Declaration

A vocabulary declaration binds a local reference to a vocabulary. It is defined with the `vocabulary` element. The `vocabulary` element defines two attributes, `name` and `src`. The `name` attribute specifies a local reference to a vocabulary; omitting the `name` attribute declares a default vocabulary in a constraint. Any unqualified attributes are assumed to be defined in the default vocabulary. Note that only one default vocabulary is allowed in a constraint. The `src` attribute specifies a vocabulary. The current release requires that only the name of a vocabulary with which the vocabulary is registered with the e-speak core be used in the `src` attribute.

```
<vocabulary name=”pcvocab” src=”PC Vocabulary” />
<vocabulary src=”PC Vocabulary” />
```

A query cannot have any vocabulary declarations. In that case, the e-speak base vocabulary is assumed by default.

Result

The `result` element is used to specify what values the user wants to have as the lookup result. WebAccess defines three meta-variables for values of the `result` element: `$esurl`, `$serviceInfo`, and `$vocabulary`. `$esurl` returns e-speak defined reference to a service whereas `$serviceInfo` returns the descriptions of a service. Using `$vocabulary`, a user specifies that she wants vocabulary definitions, such as attribute names and types, rather than vocabulary descriptions.

Constraints

The `where` element specifies the constraint. It contains a predicate over attributes defined in the vocabularies specified in the vocabulary declarations. The predicate is contained in a `condition` element. For example, the following query discovers PCs that HP manufactures and whose price is less than 1500.00.

```
<vocabulary src="PC Vocabulary" />
<result> $esurl </result>
<where>
   <condition>
      manufacture = “HP” and price &lt; 1500.00
```

```
   </condition>
</where>
```

Identifiers in the query refer to attributes in the vocabulary specified by the vocabulary attribute. String literal values should be quoted with a pair of single or double quotes. Also, the greater-than and less-than symbols need be escaped with '<,' and '&gt', respectively. For example, '>=' is represented as '>='. In the example, a vocabulary which is registered with the name "PC Vocabulary" is used as the default vocabulary.

A constraint in multiple vocabularies is described by using multiple vocabulary declarations. Suppose that employees are registered in the employee vocabulary, and managers are described using both the employee vocabulary and the manager vocabulary. The following query finds HP employees who are a project manager and are working on the e-speak probject at the Ridgeview site.

```
<vocabulary src="manager" />
<vocabulary name="emp" src="employee" />
<result> $esurl </result>
<where>
   <condition>
      emp:loc='Ridgeview' and
      (rank='project manager' and project='e-speak')
   </condition>
</where>
```

Preferences

Preference is specified with the preference element. The preference element is composed of an operator element whose value can be min, max, or with, followed by an expression element. A with preference should have an additional weight element. For example, suppose we want to order boxes in the descending order of volume. The box vocabulary defines three attributes, length, width, and height. The expression should evaluate to a value of a type where comparing two values of the type makes sense.

```
<vocabulary src="box" />
...
<preference operator="max" >
   <expr> length*width*height </expr>
</preference>
```

When multiple max/min preferences are specified, the order that the preferences appear plays an important role. The first preference provides the primary order, the second preference is used as a tiebreaker for the result of the first, so on.

The with operator takes a Boolean expression and an expression for the weight. Any numeric expression can be used for the weight. Multiple with preferences are taken all together and processed collectively. A preference whose expression evaluates to true adds its weight to the total weight. If the expression evaluates to false, 0 is added. After evaluating all the with preferences, the result is sorted in the decreasing order of the total weight. For example, the following two preferences specify that the user prefers boxes whose length is less than 5 and whose height is greater than 7 but she prefers taller ones to shorter ones.

```
<vocabulary src="box" />
<preference>
   <operator> with </operator>
   <expr> length &lt; 5 </expr>
   <weight> 2 </weight>
</preference>
<preference>
   <operator> with </operator>
   <expr> height &gt; 7 </expr>
   <weight> 4 </weight>
</preference>
```

The max/min preferences and the with preferences can be specified together, but the with preferences are given higher priority. Thus, the result is sorted according to the with preferences. Then, the first max/min preference is used to break ties, the second is used to break ties from the first, so on.

Arbitration policy

Arbitration policy limits the number of services that are returned to the client. The `arbitration` element takes `cardinaltiy` as its child element. Three different kinds of cardinalities are supported: `all`, any, and a positive integer. The `all` operator lets the core return all the results that are discovered. The core returns the first N results if it gets an arbitration of N > 0. The any operator instructs the e-speak core to return a randomly selected service. This implies that a user can get a different service even if she inquires the WebAccess with the same query. Thus, the any operator can be used to implement a transparent load balancing scheme.

```
<arbitration> <cardinalrity> “all” </cardinalrity> </arbitration>
<arbitration> <cardinalrity> “any” </cardinalrity> </arbitration>
<arbitration> <cardinalrity> “5” </cardinalrity> </arbitration>
```

E-speak Constraint Operators

The following table lists the e-speak constraint operators.

TABLE 1. e-speak constraint operators

Operator	Description
=,!=,<, <=,>,>=	Binary operators. They can only be applied if the left and right operands are of the same simple type (Except boolean). The result is a boolean. Automatic type coercion is applied.
=,!=	Binary operators. They can only be applied if the left and right operands are both boolean. The result is a boolean.
-	Unary minus.
+,-,*,div	Binary numeric operators. They can only be applied to simple numeric types. The result is a numeric.
+	Binary string concatenation operator.
*	The wildcard character for wildcard string matching.

TABLE 1. e-speak constraint operators (Continued)

=	Binary string pattern matching operator. As a binary operator,it can only be applied if left and right operands are both strings.
in	Binary operator. It can only be applied if the left operand is one of the simple types and the right operand is a set of values of the same simple type (a multi-valued attribute). The result is a boolean.
exist	Unary operator. It can be applied to any attribute. The result is a boolean.
not	Unary operator. It can only be applied to a boolean operand. The result is a boolean.
and, or	Binary operator. It can only be applied to boolean operands. The result is a boolean.

The BNF syntax of the constraint expressions is given in Appendix B.

Precedence rules

The following precedence relations hold between e-speak operators in the absence of parentheses, in the order of highest to lowest.

```
() exist unary-minus
* div
+ -
in
= != < <= > >=
not
and
or
```

String pattern matching

The semantics of '*' is the same as the wildcard character in SQL. Suppose A and B are attribute names.

```
B = “*”+A+“*” // A is a substring of B
```

```
“*Doug*” = B // B contains 'Doug'
B = “*Doug” // B ends with 'Doug'
```

Boolean Logic

E-speak implements three valued Boolean logic with `true`, `false`, and `unknown`. The following table specify the semantics.

and	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

or	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

not	
T	F
F	T
U	U

Vocabulary

In general, attribute-based description and lookup leads to a potential name collision problem. Suppose a 21-inch monitor as well as a 21-inch TV is registered. When a user tries to find a TV using a query "size=21" she can unexpectedly get the 21-inch monitor as well. To avoid name collision and to facilitate description and query validation, e-speak introduces a powerful abstraction called vocabulary and requires descriptions be specified in a specific vocabulary. A vocabulary defines a name space, i.e., a set of valid attributes and their types in the vocabulary. Note that the name space is similar to XML name space defined by XML schema [3][2]. On the other hand a vocabulary defines a set of descriptions in the vocabulary as a type in programming languages defines a set of values in the type. Thus vocabularies naturally partition the search space of descriptions.

Vocabulary is a resource managed by the e-speak engine. Since it is a resource anybody can create and register a vocabulary. A vocabulary itself is described in other vocabularies. To end the recursion e-speak defines the base vocabulary with the Type and Name attributes which is unique across all e-speak engines. Since anyone can create vocabularies two identical vocabularies can exist in an e-speak engine. The vocabulary conflict problem is addressed in future releases.

The E-speak Base Vocabulary

E-speak requires that every resource be described in a vocabulary. Since vocabularies are e-speak resources, they should be described in some vocabulary. To end the recursion, e-speak defines the base vocabulary which is not described in any vocabulary. The base vocabulary defines a number of predefined attributes including `Name`, `Type`, `ESGroup`, and `Version`. The current WebAccess release exposes only two attributes, `Name` and `Type` to clients. For the XML schema defintion of the base vocabulary supported in WebAccess, refer to Appendix A. The complete definition of the base vocabulary is given in the e-speak Architecture Specification [4].

Vocabulary Creation and Registration

Vocabulary creation and registration is essentially the same as creation and registration of any other services. Only noticeable difference is that vocabulary creation requires a vocabulary definition. A vocabulary definition is a set of attribute specifications; an attribute specification defines the name and type of an attribute among other things. The schema for vocabulary definition is listed in Appendix A.

A vocabulary definition is specified using the `attrGroup` element which is defined as a set of `attrDecl` elements. An `attrDecl` has a `name` attribute which defines the name of the attribute to be defined and a `datatypeRef` as its child which defines the type of the attribute. In addition, it can have three attributes, `required`, `multivalued`, and `dynamic`, which specify whether an attribute is mandatory to create a service, whether it allows multiple values, and whether it is a dynamic attribute, respectively.

A `datatypeRef` element has a `name` attribute specifying the name of the type and four optional element, `default`, `minInclusive`, `maxInclusive`, and `expr`, as its children. The `default` element specifies the default value of the type and `minInclusive` and `maxInclusive` are used to define a subrange type. The `expr` element defines an expression for a dynamic attribute; however, the current release does not support dynamic attributes.

The following is an example XML document which creates a vocabulary with name `carVocab`. It defines four attributes, `ask`, `year`, `descr`, and `model`, all of which are mandatory.

```
<resource>
<resourceDes>
<attr name=”Name”> <value> carVocab </value> </attr>
<attr name=”Type”> <value> Vocabulary </value> </attr>
</resourceDes>
<attrGroup xmlns=”http://www.e-speak.net/Schema/E-speak.vocab.xsd” >
<attrDecl name=”ask” required=”true”>
<datatypeRef name=”Integer”
</attrDecl>
<attrDecl name=”year” required=”true”>
<datatypeRef name=”Integer”
</attrDecl>
<attrDecl name=”descr” required=”true”>
<datatypeRef name=”String”
</attrDecl>
<attrDecl name=”model” required=”true”>
```

```
<datatypeRef name="String"
</attrDecl>
</attrGroup>
</resource>
```

Reserved Attributes

When creating a vocabulary, e-speak core adds attributes defined in the base vocabulary to the vocabulary definition by default. Consequently, clients cannot use the attribute names in vocabulary creation. Those reserved attributes are: `Name`, `Type`, `Subtype`, `ESGroup`, `Description`, `Keywords`, `Version`, `ESDate`, `ESTime`, `ESTimeStamp`, `HashAlgorithm`, and `HashCode`.

References

[1] OMG. Trading Object Service Specification.. In CORBAServices: Common Object Services Specification. chap. 16.

[2] World Wide Web (WWW) Consortium. Name Spaces in XML. W3C Recommendation. Jan 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114/.

[3] World Wide Web (WWW) Consortium. XML Schema Part 1: Structures. W3C Working Draft. April 2000. http://www.w3c.org/TR/xmlschema-1/.

[4] Hewlett-Packard. E-speak Architecture Specification. http://www.e-speak.net/library/library_white.html

Chapter 8 WebAccess Account and Session Management

Web based e-services (providing B2B and B2C interactions) need mechanisms to define and authenticate valid users and also need to delimit user activities based on sessions. These e-services need account and session management.

WebAccess Account Management

WebAccess uses e-speak account management. Users register with the account manager using a *user profil*e. Subsequently, the user profiles can be altered/modified via the account manager. The account manager also authenticates the user at the time of login.

Account

For every user that registers with the account manager, the account manager creates an *Account* resource and stores the user information as its state and maintains it in the e-speak core repository. The information about the user is maintained internally in the form of an Account Profile. The account profile is a superset of user-profile and contains additional information transparent to the user. Each registered user is assigned a globally unique (E-speak URL) ESURL, termed userESURL.

These accounts can be discovered based on certain attributes which are put in the resource description. These attributes are described in the baseAccountVocabulary. They are a part of the UserProfile information. UserProfile contains userName, UserType, UserInfo, and UserESURL along with other attributes. These attributes form the baseAccountVocabulary and are advertised by default. Attributes like UserInfo, UserType can be left blank if the user does not intend to reveal any information. These four attributes form the default resource description of the account. Users

can broaden the look up possibilities of their account resources by adding to the description of the account in arbitrary vocabularies. This provides for a large number of complex scenarios.

The rest of the UserProfile can contain any arbitrary attributes and can thus contain application specific data. The rest of the UserProfile is maintained as sensitive information in the state of the Account resource and is not visible to other users.

User Profile

A user while registering with the account manager provides a set of information in the form of a *UserProfile* which consists of *username*, *passphrase*, *userInfo*, *userType* and a set of *preferences*. The preferences is an extensible set of information stored on a per account basis. The user after successful login gets its userESURL in return. In addition to this information supplied by the user, the account manager creates certain attributes and maintains an *account profile.* The account profile is a superset of the *userprofile* and is not visible to the users.

UserName

The UserName is the name of the user.

Passphrase

pass phrase is a secret phrase by which the user is identified and along with UserName form the authentication information by which the accountmanager authenticates the user.

UserESURL

The concept of userESURL is to uniquely refer to a user and to identify it in a globally unique manner. Each e-speak engine creates unique users. The is of the form, hostname:portnumber/userName

UserInformation

One of the attributes which is part of the baseAccountVocabulary. It can be left blank or filled with the necessary information. This is advertised.

UserType

Users can have different types. This is another attribute that is advertised. Can be left blank if so desired or can be used to advertise and find e-speak accounts.

Preferences

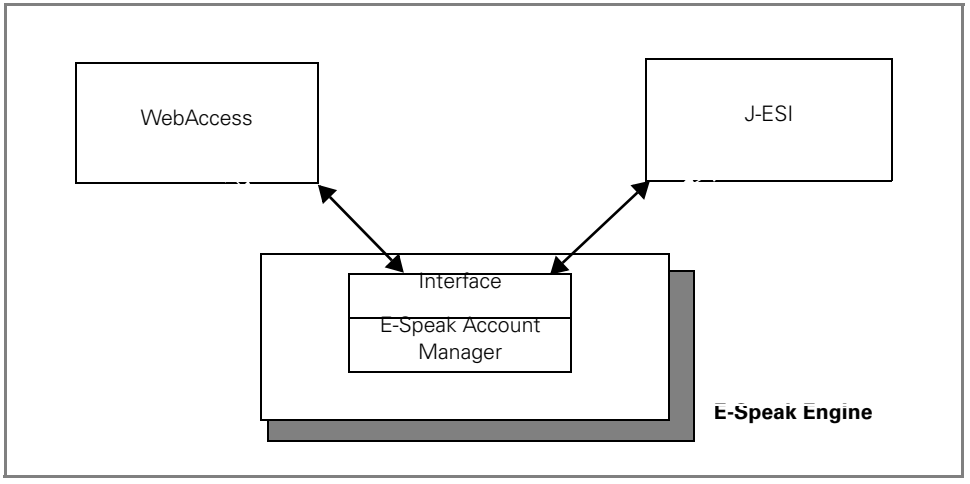
Preference may or may not be defined in a vocabulary. Preferences are meant to contain sensitive application-specific, user-related data. Preferences are not advertised to other users. Preferences are <name, value> pairs that store arbitrarily named objects.

Listed below is an example of a possible set of attributes:

- User coordinates: can contain postal address, credit card details etc.
- Preferred language.(English, French, Hindi..)

E-speak Account Manager

The account manager maintains user account profiles and authenticates users against their authentication information maintained in stored profiles.



The account manager is a core managed resource. It is a singleton service. It is exposed through a well-known client interface. WebAccess uses the client interface of the E-speak Account Manager. The Account Manager functionality involves registration of users, authentication of the user (during login), getting and setting user profiles, and adding to resource descriptions. During user login, the user request goes to the Session Manager. The Session Manager forwards the request to the Account Manager. The Account Manager authenticates the user using a SHA/MD5 or an SPKI based authentication mechanism. Once the user is successfully authenticated a session is created and a session token is given to the user.

The users are represented as[1] resources in e-speak. The user information is stored in the Account resource as a variable, namely AccountProfile in its state. The metadata of the resource contains the resource description which is described in a vocabulary specific to the account. A baseAccountVocabulary is used in this context. The users can be advertised and can be found by searching on their attributes defined in the baseAccountVocabulary. Users can add to their Account descriptions by creating /discovering relevant vocabularies and using *addDescription API* of the Account manager. This allows users the flexibility of expanding the number of searchable attributes.The account resource is persistent.

1 Account reuses the protection domains available in the core

WebAccess Session Management

WebAccess session management is light-weight and stateless. It provides a single sign-on mechanism only requiring the user to present their credentials (user-name/password, certificate,etc.) once. The login operation returns an encrypted token which contains identity and expiration data. The next request a client makes must include this token so that WebAccess can recognize and validate the client's identification. Each request returns a new token which must be included in the subsequent request. Traditional session management generates a static session ID that remains constant for the duration of the session. This differs from e-speak WebAccess session management which uses a different token for each request.

WebAccess session management is light-weight because it does not need to store transient data (name/value pairs) with the session. It only provides identification which expires after a period of idle time. It is stateless because all state information is stored in the token. The Session Manager validates the token by decryption. Since the client does not have the Session Manager's private key, it cannot create encrypted tokens that assume the identity of other users.

Scalability

Web-based e-services utilize single sign-on mechanisms for authentication that span servers, and possibly collaborating organizations. A problem associated with the authentication mechanism is synchronizing the authentication information across a network without requiring a user to present their credentials repeatedly. Traditionally this problem is solved using server side session tables and providing the user (after they present valid credentials) with an id pointing to a session table entry. That session id gives the user access to other entities on the network. The session id expires after a period of inactivity.

This traditional solution has scalability issues with environments that use multiple session managers to provide for large number of users. In order to provide for large number of users, these session states need to be shared among a set of session managers. This involves replication of state among these session managers and all the related consistency maintenance issues.

The Apache web server solves the problem by including routing information in a HTTP cookie so that requests for a particular session are always routed to the same JVM on a particular server. When a session is first created it is routed to a random server that becomes the designated server for the session. The state for that session is maintained only in the session tables on that one server. Since the routing information is stored in a session cookie, clients must implement generic HTTP cookie management. An XML thin client, such as a Palm Pilot (R) application, that does not use the browser would need more logic to handle the cookie management (traditionally used with browsers), which would consume valuable ROM/RAM on the device. Also, if that server fails, then the session information is lost.

Another solution as implemented by the BEA WebLogic application server is to mirror the session tables to all severs on the cluster. If a server fails, then the load balancing mechanism stops forwarding requests to that server. Any server is capable of servicing a user's session-based request. So the fail over is invisible to the user. This technique also has disadvantages for similar reasons as listed above. An XML client (not using a browser) would have to implement generic HTTP cookie management. This does not fit well with a document exchange model and forces the architecture to be protocol and environment specific. Using client certificates, such as SSL3, is another solution but requires a per client configuration which, in many cases, is an unacceptable business requirement.

WebAccess Session management in contrast is a scalable, lightweight and stateless solution.

Session Tokens

A token is a string (concatenation of the ESURL, last access time, user authentication information, and the home address of the e-speak engine) which is encrypted and hex encoded. The encryption key is constructed from a private key specified in the WebAccess configuration concatenated with a portion of the user's credential hash stored in the account manager.

The encrypted session token returns the following information:

- ESURL -- ES://server/home/userName
- Last Access Time -- April5,2000 12:09 pm

- Home Address -- Server:12346
- User Authentication Information, used to authenticate the user

A session can be thought of as a series of tokens with advancing last access times. Each request results in a new token with an updated last access time. A token is expired if the current time is greater than the last access time plus the expiration interval

Design

The Session Manager is a simple pipeline component that sits below the adapter dispatcher. If the message it receives is a **Login** operation then the message is forwarded down the pipeline. If the **Login** response from the pipeline is successful then a token is created and returned in the response. All other messages must have a valid token otherwise they are not forwarded down the pipeline. All responses coming back up the pipeline (except an unsuccessful **Login**) have a new token inserted in the response. Token validation is performed by decrypting the token and testing for expiration. If the token is valid, then the ESURL is placed in the 'message-Info' element's 'from' attribute.

Example Flow

This section describes the message flow in the WebAccess pipeline architecture with respect to the Session Manager.

Adapter Dispatcher

The Adapter Dispatcher passes a HTTP request to the appropriate adapter component.

Browser and XML Adapters

The Browser and XML Adapters are responsible for generating a DOM tree from the HTML, URL, or XML input. This DOM tree is passed down stream to the Session Manager.

Session Manager

If the header document specifies **Login** then the message is passed downstream to the Persistent Queue Manager. When a successful **Login** response is returned from the Persistent Queue Manager a new token is created and placed in the response header document. This is passed back upstream to the Translator Dispatcher.

Otherwise the Session Manager extracts the session token from the header document. It is decrypted and tested for expiration based on the last access time field. If the token is valid then the ESURL from the token is placed in the header document and the message is passed down the pipeline otherwise an error response is returned upstream.

Persistent Queue Manager

The Persistent Queue Manager looks for a polling operation and directly handle it. Otherwise it passes the DOM tree downstream to the Translator Dispatcher.

Translator Dispatcher

The Translator Dispatcher is responsible for "executing" requests. It has its own cache for mapping ESURL to ESConnection. If there is no mapping for the ESURL, then one is created.

E-speak Translator

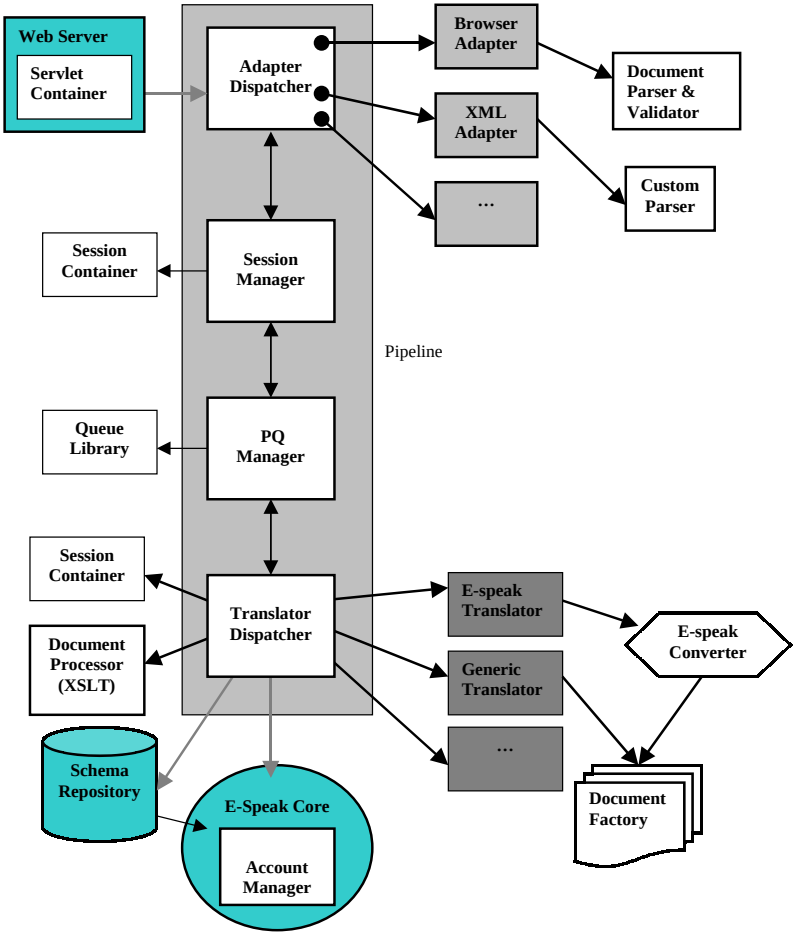
The E-speak Translator is responsible for translating requests to and from e-speak calls. It needs to directly support the following operations:

- Register -- Registers a new user.
- Login -- Creates a session.
- Logout -- Destroys a session.
- GetUserProfile -- Retrieves a user's entire profile.
- SetUserProfile -- Update user's profile.

Implementation

The session token is encrypted using Sun's Java Cryptography Extension (JCE) API. Sun provides a reference implementation and there is an internationally developed version available from http://www.cryptix.org. The token is encrypted using the "DES/EBC/PKCS5Padding" transformation.

WebAccess Callin Modules



Chapter 9 WebAccess Persistent Message Queue

The persistent message queue module provides the interfaces and the infrastructure for storing and retrieving messages.

About Messages

Before discussing the design for the persistent message queue module it is necessary to describe the ways in which messages can be configured:

- Delivery: Synchronous and Asynchronous

  Synchronous messages cause the system to wait so that a response can be returned to the sender. In Asynchronous messages the sender sends the message, then proceeds without waiting for the response.

- Storage: Persistent and Transient

  Persistent messages are stored in the queue for later processing. Transient messages are messages that do not require storage in the queue.

- Mode: Push and Pull

  In Push mode, a message is sent to the recipient. In Pull mode the message is queued and the recipient must poll for the message.

The two supported configurations are: a) **STP**: Synchronous, transient, push messages and b) **APP**: Asynchronous, persistent, pull messages.

Processing APP (Async Persistent Pull) Messages

When an APP message is sent, it is stored by the Call-in queue module of the sender's WebAccess engine. The Call-in queue module can be sent requests[1] from a recipient client to retrieve its APP messages. Message in the Call-in queue eventually timeout if they are not picked up. A service provider must connect to the system as a client in order to pickup APP messages.

Processing STP (Sync Transient Push) Messages

In a similar way, STP messages flow to the Call-out queue module of the e-speak core associated with the recipient. The Call-out queue tries to deliver STP messages for a certain number of retries. The recipient of a STP message must be connected to the system in order to receive the message. The STP messages are stored offline between retries to minimize run-time memory usage. STP messages sent to the Call-out queue module do not return a response until the messages are delivered or attempts to deliver them have failed. If a STP message cannot be delivered then an error response is created and returned to the sender.

PmqManager and the Queue

The Pmq module largely consists of the PmqManager and the queue. Having the PMQManager simplifies the interface to the queue so that any module that wants to use the queue has a single point of contact without having to know the underlying queue implementation. The PMQManager has an expiration thread that is responsible for expiring or timing out messages from the queue. The PMQManager provides a call-back mechanism using a retry thread to notify the module caller about message retries. All the messages to be retried are sent to the registered module via the call-back and the module then tries to send it. After the specifed number of retries, the message is deleted from the queue by the PMQManager retry thread.

1 Refer to the Schema and the examples to determine the request format for accessing the persistent message queue through WebAccess.

A configuration file supplies the manager with all the necessary information to start the queue. The QueueType field indicates the type of the queue, such as a file or database. The configuration file also indicates which implementation class to use for the queue. Below are configuration parameters:

- Queue Type (DATABASE or FILE)
- Queue Impl class (Java class that implements the queue Interface)
- JDBCDriver (In case of database the jdbc driver to use)
- ConnectionString (JDBC url for connecting to the database)
- Username (if needed)
- Password (if needed)

NOTE: The forthcoming release address only a database queue. Future releases have other queue types.

The queue implementation is based on an interface so that the actual persistent store and the access to it, become independent to provide more flexibility in choosing the persistent store. The main operations that must be supported by any implementation are:

- Put/get the message
- Delete the message

NOTE: The forthcoming release provides the implementation for Oracle and MySql database queues. These database implementations can serve as an example for any database that can be connected using JDBC.

Any queue implementation must implement the WAPmqIntf.

WAPmqIntf Interface

This interface is the prime interface that is implemented for any storage or database which is used for the persistent queue. The following attributes should be stored so that the queue can be queried, using these attributes. These are the attributes most likely to be used for querying.

- MessageID[2]
- Timeout (to get information on the expiration)
- Timestamp (to get information on the time of the message)
- Number of retries
- Retry interval
- Source ESURL (esurl of the sender)
- Destination ESURL (esurl of the receiver)
- Priority (this is just a place holder for now)

These attributes are stored with the XML message in the persistent store.

```
public interface WAPmqIntf
{
      void putMessage(WAPmqMsgInfo[3] msgInfo) throws Exception;
String getMessage(String mesgID) throws Exception;
Enumeration getMessage(String esurl[4], boolean bSrc) throws Exception;
Enumeration getTimeoutMessages(Timestamp expireTime) throws Exception;
Enumeration getRetryMessages(Timestamp retryTime) throws Exception;
void setNextRetryTime(String mesgID,
                                    Timestamp retryTime) throws Exception;
void setRetryNumber(String mesgID,
                          int retryNumber) throws Exception;
void deleteMessage(String mesgID) throws Exception;
void startQueue(WAPmqInfo qInfo) throws Exception;
}
```

2 MessageID generation is currently done by the queue. But a sender can generate its own messageid and send it to the queue.

3 For more information on the other classes refer to the API documentation.

4 esurl is a unique identification of the user as described in the WebAccess User Management.

The *putMessage* method takes in a WAPmqMsgInfo object as parameter. This object has all the information regarding the message including the message itself and any attachments to it. The way the message is stored in the queue is left to the specific implementation of the persistent queue. All XML messages for a given srcESURL or dstESURL can be retrieved. Similarly given the message ID, a message can be retrieved or deleted. An important method in the interface is the *startQueue* method. This method takes in as parameter a WAPmqInfo object that holds the configuration information about the queue. This method actually initialises the queue.

Default Queue Implementation

The default implementation of the persistent message queue provides queue implementations for Oracle and MySql database queues.

Oracle Queue Implementation

The Oracle queue implementation implements the WAPmqIntf. It creates the following tables if not already present: waversion, wapmq and the sequence wasequence.

The waversion table is used to indicate the version of Webaccess. This tracks version changes that may occur in the future. The waversion table contains the following field: Version.

There is only one value in the table and that is WA.03.00.

The wapmq table stores the message and its properties. The wapmq has the following table fields:

- MessageID (varchar)
- Priority (smallint)
- Timeout (int)
- Timestamp (Date)
- Expirationtime (Date)
- RetryNumber (int)
- RetryInterval (int)

- NextRetry (Date)
- SrcESURL (varchar)
- DstESURL (varchar)
- Message (long raw)

The fields Expirationtime and NextRetry time are fields that are calculated based on other parameters and are used in expiration and retry of messages.

The wasequence is a sequence that is responsible for generating the message IDs if there is no message ID specified.

```
public class WAOracleQueueImpl implements WAPmqIntf
{
   private Connection dbConn;
   private Statement dbStmt;
   private ResultSet dbRSet;

   public WAOracleQueueImpl()
   {
   }

   // Implement the WAPmqIntf methods
public void startQueue(WAPmqInfo qInfo) throws Exception
{
      // Load database driver
      Class.forName(qInfo.getJdbcDriver()).newInstance();

      // Open database connection
      dbConn = DriverManager.getConnection(qInfo.getConnString());

      // Create Statement
   }

   // All other WAPmqIntf methods

}
```

MySql Database Queue

The MySql queue implementation also implements the WAPmqIntf. It creates the following tables if not already present: waversion, wapmq and wasequence.

The waversion table is used to indicate the version of Webaccess. This tracks version changes that may occur in the future. The waversion table has the following field: Version.

There is only one value in the table and that is WA.03.00.

The wapmq table stores the message and its properties. The wapmq has the following table fields:

- MessageID (varchar)
- Priority (smallint)
- Timeout (int)
- Timestamp (Timestamp)
- Expiriationtime (Timestamp)
- RetryNumber (int)
- RetryInterval (int)
- NextRetry (Timestamp)
- SrcESURL (varchar)
- DstESURL (varchar)
- Message (long varbinary)

The fields Expiriationtime and NextRetry time are fields that are calculated based on other parameters and are used in expiration and retry of messages.

The wasequence is a table that is responsible for generating the message IDs if there is no message ID specified.

```
public class WAMySqlQueueImpl implements WAPmqIntf
{
   private Connection dbConn;
```

```
   private Statement dbStmt;
   private ResultSet dbRSet;

   public WAMySqlQueueImpl()
   {
   }

   // Implement the WAPmqIntf methods
public void startQueue(WAPmqInfo qInfo) throws Exception
{
     // Load database driver
     Class.forName(qInfo.getJdbcDriver()).newInstance();

     // Open database connection
     dbConn = DriverManager.getConnection(qInfo.getConnString());

     // Create Statement
   }

   // All other WAPmqIntf methods
}
```

Configuration of the Persistent Message Queue

The persistent message queue can be configured using the webaccess.xml configuration file. This file has two sections, the WACallinQueue and the WACalloutQueue. Both sections contain the queue configuration. The configurable parameters at the time of this release are:

- Queue Type (DATABASE or FILE)
- Queue Impl class (Java class that implements the queue interface)
- JDBCDriver (In case of database the jdbc driver to use)
- ConnectionString (JDBC url for connecting to the database)
- Username (if needed)

- Password (if needed)

A sample configuration for the MySql database is shown below:

```
<WACallinQueue enable="yes">
  <QueueConfiguration>
    <QueueType>Database</QueueType>
    <ImplClassName>
      net.espeak.webaccess.util.modules.pmq.WAMySqlQueueImpl
    </ImplClassName>
    <JDBCDriverName>org.gjt.mm.mysql.Driver</JDBCDriverName>
    <JDBCConnectionUrl>
      jdbc:mysql://rgelpc304.rgv.hp.com/pmqueue?user=root
    </JDBCConnectionUrl>
  </QueueConfiguration>
</WACallinQueue>

<WACalloutQueue enable="yes">
  <QueueConfiguration>
    <QueueType>Database</QueueType>
    <ImplClassName>
      net.espeak.webaccess.util.modules.pmq.WAMySqlQueueImpl
    </ImplClassName>
    <JDBCDriverName>org.gjt.mm.mysql.Driver</JDBCDriverName>
    <JDBCConnectionUrl>
      jdbc:mysql://rgelpc304.rgv.hp.com/pmqueue?user=root
    </JDBCConnectionUrl>
  </QueueConfiguration>
</WACalloutQueue>
```

To use a new queue implementation, modify the webaccess.xml file to usethe correct parameters for the queue and restart WebAccess. The WAPmqManager reads this information and instantiate the implementation class, passing in all the other configuration parameters. What the class does with these parameters is left to the class implementation.

Appendix A E-speak XML Schema Specification

Schema For the E-speak Document Header

The e-speak document header contains information that is used for handling and processing of the document. It consists of one required and two optional components:

- **Communication --** specifies routing related information (required).
- **Description** -- is used for system administration (optional).
- **Content --** provides the more information about the document. For example, how it is encrypted or compressed (optional).

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.header.xsd"
        xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:es-header="http://www.e-speak.net/Schema/E-speak.header.xsd">

   <annotation>
      <documentation>E-speak document header </documentation>
   </annotation>

   <attributeGroup name="messagePropertiesType">
      <attribute name="timeout" type="integer"/>
      <attribute name="synchronous" type="boolean" use=”default” value="true"/>
      <attribute name="persistent" type="boolean" use=”default” value="true"/>
      <attribute name="retry" type="integer"/>
      <attribute name="retryInterval" type="integer"/>
      <attribute name="requireReceipt" type="boolean" user=”default”
value="false"/>
      <attribute name="pull" type="boolean" use=”default” value="false"/>
      <!-- other used defined attributes -->
      <anyAttribute namespace="##any"/>
   </attributeGroup>

   <complexType name="communicationType" content="elementOnly">
      <annotation>
         <documentation>routing information and routing-related properties
```

```
      </documentation>
    </annotation>
    <complexType name="addressType" base="uri_reference"
                          derivedBy="extension">
      <attribute name="encoding"  type="string" />
   </complexType>
    <element name="to" type="es-header:ddressType"/>
    <element name="from" type="es-header:addressType"/>
    <element name="notify" minOccurs="0" type="es-header:addressType"/>
    <element name="context" minOccurs="0">
      <annotation>
        <documentation>context encapsulates the routing information that's
          valid across documents. It indicates properties that can be reused
           for many different documents exchanges. e.g. session token.
        </documentation>
      </annotation>
      <complexType>
        <element name="sessionToken" type="string" minOccurs="0"/>
        <!-- any extra elements -->
        <any minOccurs="0" maxOccurs="unbounded"
             namespace="##any" processContents="lax"/>
      </complexType>
    </element>
    <element name="messageInfo" minOccurs="0">
      <complexType content="elementOnly">
        <element name="messageId" type="string"/>
        <element name="replyToId" type="string" minPOccurs="0"/>
        <!-- timestamp of the message -->
        <element name="sentTime" type="timeInstance" minOccurs="0"/>
        <element name="messageProperties" minOccurs="0">
          <complexType>
            <attibuteGroup ref="es-header:messagePropertiesType"/>
          </complexType>
        </element>
      </complexType>
    </element>
</complexType>

<complexType name="descriptionType" content="elementOnly">
    <annotation>
      <documentation>information of the document for the system admins
      </documentation>
    </annotation>
    <element name="manifest" minOccurs="0">
      <complexType>
        <any namespace="##any" processContents="lax"/>
      </complexType>
    </element>
    <element name="properties" minOccurs="0">
      <complexType>
        <any namespace="##any" processContents="lax"/>
      </complexType>
    </element>
```

```
   </complexType>

   <complexType name="contentType" content="elementOnly">
      <annotation>
         <documentation>
            information of the body of the document, e.g., encrption
         </documentation>
      </annotation>
      <element name="document" minOccurs="0">
         <complexType>
            <any namespace="##any" processContents="lax"/>
         </complexType>
      </element>
      <element name="attachment" minOccurs="0" maxOccurs="unbounded">
         <complexType>
            <any namespace="##any" processContents="skip"/>
         </complexType>
      </element>
   </complexType>

   <complexType name="esHeaderType">
      <element name="communication" type="es-header:communicationType"/>
      <element name="description" type="es-header:descriptionType"
               minOccurs="0"/>
      <element name="content" type="es-header:contentType" minOccurs="0"/>
   </complexType>

</schema>
```

Some of the tags are described in Table 2. Notice the message header is defined as an open model, so that the users can add their own message elements and attributes by extending the basic model defined here.

Table 2 Message Header Tags

Tag	Required	Occurs	Description
Communication	**yes**	1	Contains routing information and message-specific information that can impact the message delivery.
To	**yes**	1	The receiver of the message
From	**yes**	1	The sender of the message

Table 2 Message Header Tags (Continued)

Tag	Required	Occurs	Description
NotifyTo	**no**	0 or 1	The event handler of the message
Context	**yes**	1	Encapsulates contextual routing information that is valid across messages. For example, the security context to be used to send the message; the session the message belongs to.
MessageInfo	**yes**	1	The message-specific information that is related to routing. For example, messageId, sent timestamp, timeout, synchronous or asynchronous, etc.
Description	**no**	0 or 1	Indicates what information is being transmitted in a human-readable form. It's primarily useful for system administrator to build log information.
MessageContent	**no**	0 or 1	The content information of the message, e.g. how it is compressed, encoded and the attachments
Encoding	**no**	0 or 1	Logical identifier of the encoding of address type of the sender (from)/receiver(to)
sessionToken	**no**	0 or 1	Session identifier.

Table 2 Message Header Tags (Continued)

Tag	Required	Occurs	Description
messageId	**yes**	1	A unique id that identifies the message.
replyToId	**no**	0 or 1	An identifier used to correlate messages.
sent	**no**	0 or 1	Time stamp of the message.
messageProperties	**no**	0 or 1	A group of message attributes. e.g. synchronous, timeout, priority, retry, push/pull, etc.
manifest	**no**	0 or 1	A human-readable description of the message. Primarily used by system administrators to build logs.
properties	**no**	0 or 1	Structural information of the message. Primarily used by system administrators to build logs.
document	**no**	0 or 1	The information about message body needed by applications that handle it. For example, how it is compressed, encoded, etc.
attachment	**no**	0 or *	The information about attachments within the messages needed by applications that handle them. For example, how they are compressed, encoded, etc.

Schema For E-speak Account Management

The schema for e-speak user account management is used to register accounts, update or retrieve account information. It specifies three components:

- **Authority,** the credential of the operator, which takes the form of either user-name/password or certificate.
- **accountDes,** the description of the account.
- **Preferences,** the account specific data. It can contain {name, value} paris.

The Login operation also uses the credential definition.

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.account.xsd"
    xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:es-acct="http://www.e-speak.net/Schema/E-speak.account.xsd"
        xmlns:es-base-acct
="http://www.e-speak.net/Schema/E-speak.base-account.xsd">

<import namespace="http://www.e-speak.net/Schema/E-speak.base-account.xsd"/>

   <annotation>
      <documentation>Schema of e-speak account management</documentation>
   </annotation>

   <!-- user credential-->
   <group name="credential">
      <choice>
         <element name="userNamePass">
            <complexType content="elementOnly">
               <element name="userName" type="string"/>
               <element name="passPhrase" type="string"/>
               <element name="homeAddress" type="string" minOccurs="0" />
            </complexType>
         </element>
         <element name="certificate">
            <complexType content="empty">
               <element name="certificateSubject" type="string"/>
               <element name="certificateAuthority" type="string"/>
               <element name="certificateLevel" type="string" minOccurs="0" />
            </complexType>
         </element>
      </choice>
   </group>

   <element name="preference" content="empty" minOccurs="0">
       <complexType>
           <attribute name="name" type="string" use="required"/>
```

```
            <attribute name=”value” type=”string” use=”required”/>
        </complexType>
    </element>

    <element name="authority">
       <complexType>
             <element ref="es-acct:credential" />
       </complexType>
    </element>

     <element name=”accountInfo”>
         <complexType>
              <!--  description of the account   -->
        <element name=”accountDes” type=”es-base-acct:baseAcctVocab” />
             <!--  authority information may needed for the account manipulation  -->
               <element ref=”es-acct:authority” />
               <!--  preferences -->
             <element ref=”es-acct:preference” minOccurs=”0” maxOccurs=”unbounded”/>
          </complexType>
     </element>
</schema>
```

Schema For Creating an E-speak Vocabulary

A vocabulary consists of a set of attribute properties in e-speak, which has a name, an associated type, and optional default value, value range, mandatory attributes, etc. A vocabulary is a resource that can be described in another vocabulary and discovered by other resource. In XML, each attribute property is specified as an element declaration.

The principle issues for representing vocabularies in XML using schemas are datatype issues and whether attribute properties are scalar-valued or vector-valued. For now, WebAccess can only support the e-speak base data types[1] and scalar-valued attribute properties. Furthermore, the vocabularies can only be described in the e-speak base vocabulary. The XML data types used in specifying the individual attributes are defined in XML Schema Part 2: Datatypes[2] document. The mapping between e-speak basic data types and XML built-in primitive data types are listed in Chapter 5. Typical values of a property are static; their values are specified when the resource is registered.

1 See e-speak architecture specification

2 See http://www.w3.org/TR/2000/WD-xmlschema-2-20000407/#built-in-primitive-datatypes.

The schema of creating an E-speak vocabulary:

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.vocab.xsd"
    xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:es-vocab="http://www.e-speak.net/Schema/E-speak.vocab.xsd">

   <annotation>
      <documentation>Schema of creating an e-speak vocabulary</documentation>
   </annotation>

  <!-- define the attribute property set -->
  <element name=”attrGroup” minOccurs=”1”>
       <complexType>
            <element name=”attrDecl” minOccurs=”1” maxOccurs=”unbounded”
                          content=”elementOnly”>
                 <complexType>
                      <element ref=”es-vocab:datatypeRef”>
                      <attribute name=”required” type=”boolean” use=”default”
value=”false” />
                      <attribute name=”multivalued” type=”boolean” use=”default”
value=”false” />
                      <attribute name=”name” type=”string” use=”required” />
                 </complexType>
            </element>
            <!-- a descriptive name for a vocabulary -->
            <attribute name=”name” type=”string” />
       </complexType>
   </element>

   <element name=”datatypeRef”>
       <complexType>
           <!-- the type of the following is specified by the above type attribute
-->
             <!-- it is defined as string type just for validation purpose -->
             <element name=”default” type=”string” minOccurs=”0” maxOccurs=”1”/>
          <element name=”minInclusive” type=”string” minOccurs=”0” maxOccurs=”1”/>
          <element name=”maxInclusive” type=”string” minOccurs=”0” maxOccurs=”1”/>
      <!-- the name of the datatype -->
             <attribute name=”name”  type=”string” use=”required”/>
       </complexType>
   </element>
</schema>
```

The Schema For an E-speak Base Vocabulary

Since vocabulary is a resource and any resource must be described in some vocabulary, there is a recursion. The e-speak base vocabulary is predefined in e-speak and is used to end the recursion.

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.base.xsd"
       xmlns="http://www.w3.org/1999/XMLSchema”>

   <annotation>
      <documentation>Schema of e-speak base vocabulary</documentation>
   </annotation>

   <!-- define the e-sepak base vocabulary: Name & Type -->
   <element name="Name"  type="string"/>
   <element name="Type"  type="string"  minOccurs="0" maxOccurs=”1”/>

   <complexType name=”baseVocab”>
       <element ref=”base:Name”/>
       <element ref=”base:Type”/>
   </complexType>

</schema>
```

The Schema For an E-speak Base Account Vocabulary

The e-speak base account vocabulary is predefined and used for describing an e-speak account.

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.base-account.xsd"
     xmlns="http://www.w3.org/1999/XMLSchema”
         xmlns:baseAcct="http://www.e-speak.net/Schema/E-speak.base-account.xsd">

   <annotation>
      <documentation>Schema of e-speak base account vocabulary</documentation>
   </annotation>

   <element name="UserName"  type="string"/>
   <element name=”PassPhrase” type="string" minOccurs="0" maxOccurs=”1”/>
   <element name="UserType"  type="string" minOccurs="0" maxOccurs=”1”/>
   <element name=”UserInfo” type=”string” minOccurs="0" maxOccurs=”1”/>
   <element name=”UserESURL” type=”string” minOccurs="0" maxOccurs=”1”/>

   <complexType name=”baseAcctVocab”>
        <element ref=”baseAcct:UserName”/>
        <element ref=”baseAcct:PassPhrase”/>
```

```
        <element ref=”baseAcct:UserType”/>
        <element ref=”baseAcct:UserInfo”/>
        <element ref=”baseAcct:UserESURL”/>
   </complexType>
</schema>
```

Schema For an E-speak Service Registration

Services should contain metadata that enable access as well as provide account managment and discovery information when registered with e-speak. The metadata consists of resource specification and description.

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.register.xsd "
        xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:es-reg="http://www.e-speak.net/Schema/E-speak.register.xsd">

   <annotation>
      <documentation>
    E-speak basic schema for describing a resource.
      </documentation>
   </annotation>

   <element name="resourceSpec">
       <complexType>
            <!-- define the resource specification -->
            <element name=”srcAddress” type=”uri-reference” minOccurs=”0”
                          maxOccurs=”unbounded”/>
            <element name="locator" type="uri-reference" minOccurs="0"
                           maxOccurs="unbounded"/>
            <element name="filter" type="string"
                           minOccurs="0" maxOccurs="unbounded"/>
            <element name="data" minOccurs="0" maxOccurs="1">
               <complexType>
                  <any namespace="##any" processContents="skip"/>
               </complexType>
            </element>
       </complexType>
   </element>

   <element name="resourceDes" minOccurs="1" maxOccurs="unbounded">
      <annotation>
         <documentation> A service registration can contain several resource
description
              items. Each of them consists of one of more properties defined in
a vocabulary.
             The vocabulary is specified with its registered name (ESURL) or its
external
```

```
                URI. It works as a namespace for the XML elements.
            </documentation>
        </annotation>
        <complexType>
             <!-- specify the vaocabulary. it can be the its name or the ESURL -->
             <element name=”vocabulary” type=”string”/>
             <element name=”attr” minOccurs=”1” maxOccurs=”unbounded”/>
                  <complexType>
                      <!-- the type of the value is defined by the vocabulary. It’s
defined as ‘string”
                           here just for validation. Proper type casting is done
by e-speak -->
                       <element name=”value” type=”string”/>
                       <attribute name=”name” type=”string”/>
                  </complexType>
             </element>
             <attribute name=”name” type=”string”/>
        </complexType>
    </element>

    <element name=”resource”>
        <complexType>
             <element ref=”es-reg:resourceSpec” minOccurs=”0” maxOccurs=”1”/>
            <element ref=”es-reg:resourceDes” minOccurs=”1” maxOccurs=”unbounded”/>
        </complexType>
    </element>
</schema>
```

Schema For an E-speak Service Query

The discovery of services registered with e-speak is through a query on the meta-data of the services.

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.query.xsd "
        xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:es-query="http://www.e-speak.net/Schema/E-speak.query.xsd">

    <element name=”esquery”>
        <complexType>
          <!-- from: where the query is applied. It refers to the uri of an e-speak
repository>
             <element name=”from” type=”uri-reference” minOccurs=”0”
maxOccurs=”unbounded”/>
             <element name=”vocabulary” type=”es-query:vocabType” minOccurs=”0”
                       maxOccurs=”unbounded” content=”empty”/>
             <element name=”result” type=”es-query:meta-variable” minOccurs=”0”
maxOccurs=”1”/>
```

```
          <element name="where" type="es-query:constraintType" minOccurs="1"
maxOccurs="1"/>
         <element name="preference" type="es-query:preferenceType" minOccurs="0"
                        maxOccurs="unbounded"/>
       <element name="arbitration" type="es-query:arbitrationType" minOccurs="0"
                        maxOccurs="1"/>
      </complexType>
   </element>

   <!-- result specifies the output of the query -->
   <simpleType name="meta-variable" base="string">
       <!-- the ESURL of a service -->
       <enumeration value="$esurl"/>
       <!--the specification and description of a service -->
       <enumeration value="$serviceInfo"/>
       <!--  vocabulary  -->
       <enumeration value="$vocabulary"/>
   </simpleType>

   <complexType name="vocabType">
       <!-- the prefix. If not presented, it is the default vocabulary -->
       <attribute name="name" type="string" use="optional"/>
        <!-- the name or ESURL of the vocabulary -->
        <attribute name="src" type="name"/>
   </complexType>

   <!-- specify the constraint -->
   <complexType name="constaintType">
         <element name="condition" type="string"/>
   </complexType>

   <!--  preference : prioritize the query results -->
   <complexType name="preferenceType">
       <element name="operator" base="string">
           <simpleType>
               <enumeration value="min"/>
               <enumeration value="max"/>
               <enumeration value="with"/>
           </simpleType>
       </element>
       <element name="expr" type="string" />
       <element name="weight" type='string" minOccurs="0" maxOccurs="1" />
   </complexType>

    <!--   arbitration:  pick one or more results -->
    <complexType name="arbitrationType">
        <!--  cardinality cab be one of the special values, "first", "all",
"negotiate", or a positive
                number  --->
       <element name="cardinality" type="string" minOccurs="0" maxOccurs="1'/>
   </complexType>
</schema>
```

Schema For Responses From E-speak Internal Services

```
<?xml version="1.0"?>
<schema targetNameSpace="http://www.e-speak.net/Schema/E-speak.response.xsd"
                xmlns="http://www.w3.org/1999/XMLSchema"

xmlns:es-register=”http://www.e-speak.net/Schema/E-speak.register.xsd"
                xmlns:es-vocab=”http://www.e-speak.net/Schema/E-speak.vocab.xsd"
            xmlns:es-account=”http://www.e-speak.net/Schema/E-speak.account.xsd">

<import namespace="http://www.e-speak.net/Schema/E-speak.account.xsd"/>
<import namespace="http://www.e-speak.net/Schema/E-speak.register.xsd"/>
<import namespace="http://www.e-speak.net/Schema/E-speak.query.xsd"/>

<annotation>
      <documentation>E-speak response message</documentation>
</annotation>

<element name="responseType" content="elementOnly">
  <complexType>
   <!-- This element is for extra information whenever there is a failure -->
   <element name="responseInfo" type="string" minOccurs="0" maxOccurs="1"/>
   <!-- This element is for Poll message response -->
   <element name=”messageList” minOccurs=”0”>
      <complexType>
         <element name=”message” maxOccurs=”unbounded”>
         <complexType>
            <element name=”messageId” type=”string”/>
            <element name=”from” type=”uri-reference”/>
         </complexType>
         </element>
      </complexType>
   </element>
   <!-- This element is used to return response for RegisterAccount -->
   <element name="userESURL" type="uri-reference" minOccurs="0" maxOccurs"1"/>
   <!-- This element is used to return response for RegisterService/FindService,
etc. -->
   <element name="resource" type="es-register:resource" minOccurs="0"

maxOccurs="unbounded"/>
   <!-- This element is used to return response for GetAccountProfile -->
   <element name="accountInfo" type="es-account:accountInfo" minOccurs="0"
maxOccurs="1"/>
  <!-- This element is used to return response for CreateVocabulary/FindVocabulary
-->
   <element name="attrGroup" type="es-vocab:attrGroup" minOccurs="0"
maxOccurs="1"/>
   <attribute name="status">
      <simpleType base="string">
         <enumeration value="succeeded"/>
```

```
            <enumeration value="failed"/>
          </simpleType>
        </attribute>
      </complexType>
    </element>
    </schema>
```

Appendix B BookBroker Service Source Code

The source code of the complete BookBroker example is made available with the source distribution of e-speak. For the purpose of a brief overview, the source code of the BookBroker service is shown here.

```
package net.espeak.webaccess.htdocs.servlets;
/**
* The Book Broker is an example of how to use WebAccess as a programming
 * platform. The Book Broker initially sends out a book query form in which
 * the user fills in a book description. After SUBMIT, the book broker reads
 * the parameters from the FORM POST request and maps them into a XML book
 * query form to be sent to proxy services.
 */
public class BookBroker extends HttpServlet {

    ServletConfig _config = null;
    boolean replyHTML = true;

private String bookQueryForm = ""
      + "  <FORM METHOD=POST NAME=\"BBForm\" ACTION=\"/servlets/BookBroker\">\r\n"
        + "  <table border=0><td>   <td>\r\n"
        + "   <table border=0>\r\n"
       + "    <td>Author<td>    <INPUT TYPE=\"text\" NAME=\"author\" size=\"36\">
<tr>\r\n"
        + "    <td>Title<td>     <INPUT TYPE=\"text\" NAME=\"title\" size=\"36\"
><tr>\r\n"
       + "   <td>Subject<td>   <INPUT type=\"text\" NAME=\"subject\" size=\"36\">
<tr>\r\n"
      + "   <td>Publisher<td> <INPUT type=\"text\" NAME=\"publisher\" size=\"28\">
<tr>\r\n"
        + "    <td>ISBN<td>      <INPUT type=\"text\" NAME=\"ISBN\" size=\"28\">
<tr>\r\n"
        + "    <td colspan=2>"
        + " <br><br><br><INPUT type=\"submit\" value=\"Search Now\">\r\n"
        + " <tr><td><br><br>"
        + " <table border=0 cellspacing=0 cellpadding=0>" + " <td>"
        + " <small><small><font color=gray>reply:</font></small></small>"
        + " <td> <font color=red>   >></font>"
        + " <td> <input type=radio name=\"WA_TGTCT\" value=\"TYPE5_DOMRPL\">"
        + " <small><small><font color=gray>RXML</font></small></small>"
```

```
        + " <td> <input type=radio name=\"WA_TGTCT\" value=\"TYPE5_HTMLRPL\"
checked>"
        + " <small><small><font color=gray>RHTML</font></small></small>"
        + "  <tr>" + " </table>" + "   </table><tr>" + "  </table>\r\n"
        + "  </FORM>\r\n" + "</body>\r\n" + "</html>\r\n";

private String XMLRespHeader = "" + "<html><?xml version=\"1.0\"?>\r\n"
      + "<header xmlns=\"http://www.e-speak.net/Schema/E-speak.header.xsd\">\r\n"
        + "<communication>\r\n"
        + "<to>es://localhost:123456/whoEver</to>\r\n"
        + "<from>http://localhost/servlets/BookBroker</from>\r\n"
        + "</communication>\r\n" + "</header>\r\n";

private final static String _rLogin = "WA_OP=OPS_LOGIN&"
        + "PAR_USERNAME=esadmin&" + "PAR_PASSWORD=esadmin&";

private final static String _rCreatBrokerVocab = "WA_OP=OPS_CRTVOC&"
        + "PAR_VOCAB=brokerVocab&" + "PAR_VOCATTRS=name,goods,location&";

private final static String _rRegBookBroker = "WA_OP=OPS_REGSRV&"
        + "PAR_VOCAB=brokerVocab&"
        + "PAR_URL=http://localhost/servlets/BookBroker&"
        + "name=myBookBroker&" + "location=Cupertino&" + "goods=books&";

private final static String _rQueryBookstores = "WA_OP=OPS_DSCSRV&"
        + "PAR_VOCAB=bookstoreVocab&" + "goods=books&";

private final static String _rCreatBookstoreVocab = "WA_OP=OPS_CRTVOC&"
        + "PAR_VOCAB=bookstoreVocab&" + "PAR_VOCATTRS=name,goods&";

private final static String _rRegProxyBarnes = "WA_OP=OPS_REGSRV&"
        + "PAR_VOCAB=bookstoreVocab&"
        + "PAR_URL=http://localhost/servlets/ProxyBarnes&"
        + "name=ProxyBarnes&" + "goods=books&";
private final static String _rRegProxyAmazon = "WA_OP=OPS_REGSRV&"
        + "PAR_VOCAB=bookstoreVocab&"
        + "PAR_URL=http://localhost/servlets/ProxyAmazon&"
        + "name=ProxyAmazon&" + "goods=books&";

private final static String _rRegProxyFatBrain = "WA_OP=OPS_REGSRV&"
        + "PAR_VOCAB=bookstoreVocab&"
        + "PAR_URL=http://localhost/servlets/ProxyFatBrain&"
        + "name=ProxyFatBrain&" + "goods=books&";

private final static String _rLogout = "WA_OP=OPS_LOGOUT&";


/**
     * Initializes variables at the servlet level and starts the ServiceDaemon
     * @param  config  the configuration to be used for initialization
     */
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
```

```
        _config = config;
        BookBrokerHelper.webServerInfo();
    }


/**
     * Posts the HTTP request. *
     * @param      req    the request in the form of HTTP
     * @param      res    the response in the form of HTTP
     * @exception  IOException       when an input/output error occurs
     * @exception  ServletException  when an error occurs with the servlet
     */
    public void doPost(HttpServletRequest req,
        HttpServletResponse res) throws ServletException, IOException {
        replyHTML = true;
        try {
            PrintWriter out = res.getWriter();
            if (!req.getContentType().equals("text/xml")) {
                // extract parameters from FORM POST request
                String author = req.getParameter("author");
                String title = req.getParameter("title");
                String subject = req.getParameter("subject");
                String publisher = req.getParameter("publisher");
                String ISBN = req.getParameter("ISBN");
                String TGTCT = req.getParameter("WA_TGTCT");
                if (TGTCT != null) {
                    if (TGTCT.equalsIgnoreCase("TYPE5_DOMRPL")) {
                        replyHTML = false;
                    }
                }
              String bqXML = "";    // create book query xml sent to proxy services
                bqXML += "<?xml version=\"1.0\"?>\r\n";
                bqXML += "<bookquery>";
                bqXML += "  <author>" + author + "</author>\r\n";
                bqXML += "  <title>" + title + "</title>\r\n";
                bqXML += "  <subject>" + subject + "</subject>\r\n";
                bqXML += "  <publisher>" + publisher + "</publisher>\r\n";
                bqXML += "  <ISBN>" + ISBN + "</ISBN>\r\n";
                bqXML += "</bookquery>\r\n";

                // prepare discovery of proxy services in the core
                String sessID = BookBrokerHelper.doReq(_rLogin, "_rLogin", "");
                String requ = BookBrokerHelper.packWAReq(_rQueryBookstores,
                    sessID, "TYPE5_DOMRPL");
   // execute the login for the proxy service lookup
                String resp = BookBrokerHelper.sendReq(requ);
                sessID = BookBrokerHelper.getSessID(resp);
                String status = BookBrokerHelper.getStatus(resp);
                // String to collect the final XML booklist returned from proxy
services
                String queryResultXML = "";
                queryResultXML += "<?xml version=\"1.0\"?>\r\n";
```

```
                queryResultXML +=
                    "<booklist>";    //
xmlns=\"http://www.e-speak.net/Schema/BookBroker/booklist.xsd\">\r\n";
                for (int size = 0; true; ) {
                    // find proxy services
                    String begPat = "</font>&lt;locator&gt;<font color=blue>";
                    String endPat = "</font>&lt;/locator&gt;";
                    int i = resp.indexOf(begPat, size) + begPat.length();
                    int j = resp.indexOf(endPat, size);
                    if (i <= 0 || j < i)
                        break;
                    size = j + endPat.length();
                    String url = resp.substring(i, j);
                    // send book query XML to proxy service
                    String proxyReq = BookBrokerHelper.packHTTPReq(url,
                        "text/xml", bqXML);
                    queryResultXML += BookBrokerHelper.sendReq(proxyReq);
                }
                queryResultXML += "</booklist>\r\n";
                if (replyHTML) {
                    String queryResultHTML =
                        mapQueryResultToHTML(queryResultXML);
                    out.println(queryResultHTML);
                } else {
                  AccessXml aml = new AccessXml(new StringReader(queryResultXML));
                    String reply = aml.getText(true);
                    out.println(reply);
                }
            } else {
                // initial invocation through web access, return XML reply
                // plus the book query form to be returned from the browser
                out.println(htmlBookBrokerHeader("Book Query Form"));
                out.println(bookQueryForm);
            }
        } catch (Exception e) {
            res.setContentType("text/html");
            PrintWriter out = res.getWriter();
            out.println("<h3>" + e + "</h3>");
            out.println("</BODY></HTML>");
        }
    }


/**
     * Posts the HTTP request.
     * @param      req    the request in the form of HTTP
     * @param      res    the response in the form of HTTP
     * @exception  IOException       when an input/output error occurs
     * @exception  ServletException  when an error occurs with the servlet
     */
    public void doGet(HttpServletRequest req,
        HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
```

```
        PrintWriter out = res.getWriter();
        out.println(htmlBookBrokerHeader("test page, " + (new Date())));
    }


/**
     * returns the HTML version of the <booklist> query result
     * @param xml <booklist> xml to be mapped into HTML
     * @return HTML version of the <booklist> query result
     */
    private String mapQueryResultToHTML(String xml) throws IOException {
        String html = htmlBookBrokerHeader("Book Query Result");
        html += "<TABLE border=0 width=440>\r\n";
        AccessXml aml = new AccessXml(new StringReader(xml));
        Document dom = aml.getDocument();
        int n = aml.countNodes("booklist/bookoffer");
        for (int i = 0; i < n; i++) {
            String author = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/author");
            String title = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/title");
            String publisher = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/publisher");
         String year = Util.getElement(dom, "booklist/bookoffer[" + i + "]/year");
            String offeredBy = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/offeredBy");
            String price = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/price");
            String shipsin = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/shipsin");
            String URLtoOrder = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/URLtoOrder");
            String comments = Util.getElement(dom, "booklist/bookoffer[" + i +
"]/comments");
            int ii = i + 100;
            html += "<TD><TABLE border=0><TD>";
            if (author != null)
                html += "<small>" + author + ":</small> ";
            if (title != null)
                html += "<i>\"" + title + "\"</i>, ";
            if (publisher != null)
                html += "<small>" + publisher + "</small>, ";}
            if (year != null)
                html += "<small>" + year + "</small>, ";
            if (price != null)
                html += "<small>offered for</small> <b>" + price + "</b>, ";
            if (shipsin != null)
                html += "<small>usually ships in " + shipsin + "</small>, ";
            if (comments != null)
                html += "<small>" + comments + "</small>";
            html += ".<TR>";
            html += "<TD><small>direct order: <a href=\"" + URLtoOrder;
```

```
            html += "\">" + offeredBy + "</a>.</small>";
            html += "<TR><TD><HR><TR>";
            html += "<TR></TABLE><TR>\r\n";
        }
        html += "</TABLE>\r\n";
        html += "</BODY></HTML>\r\n";
        return html;
    }
/**
     * returns a version of the string where '<' and '>' are replaced
     * by respective special charaxters used in HTML. This function
     * is needed to display XML text in regular browsers
     * @param str string to unTagify
     * @return tagified string
     */
    private String htmlify(String str) {
        StringBuffer strb = new StringBuffer(str);
        String result = "";
        boolean inQuotes = false;
        for (int i = 0; i < strb.length(); i++) {
            char c = strb.charAt(i);
            switch (c) {
            case '<':
                result += "</font>";
                result += "&lt;";
                break;
            case '>':
                result += "&gt;";
                result += "<font color=blue>";
                break;
            case '"':
                inQuotes = !inQuotes;
                if (inQuotes) {
                    result += "\"<b>";
                } else {
                    result += "</b>\"";
                }
                break;
            default:
                result += c;
            }
        }
        return result;
    }
/**
     * generated the HTML header part for the Book Broker
     * @param title title of page
     * @return HTML header part
     */
    String htmlBookBrokerHeader(String title) {
        String html = "<HTML>\r\n";
        html += "<HEAD><TITLE>...myBookBroker's </TITLE></HEAD>\r\n";
        html += "<BODY bgcolor=\"EEFBEE\">\r\n";    // bgcolor=\"EEFBEE\">\r\n";
```

```
        html +=
            "  <big><big><b><font color=\"blue\">. . . myBookBroker's</font>\r\n";
        html += "      ";
        html += "       <small><font color=\"gray\">" + title + "</font>";
        html += "       </small></big></big></p>\r\n";
        html += "<HR align=left width=440><br><br>";
        return html;
    }
```

The BookBroker service as shown is implemented as a servlet. It does not need a main() function. The main() function shown on this page is used to run the BookBroker from the command line to register all needed services in order to run the BookBroker scenario.

```
/**
     * main function used to automatically all resources needed to run the
     * BookBroker demo *
     * @param argsv args passed from the command line
     */
    public static void main(String[] argsv) {
        String req;
        String resp;
        String sessID;
        String status;    // boolean status;
        sessID = BookBrokerHelper.doReq(_rLogin, "_rLogin", "");
        sessID = BookBrokerHelper.doReq(_rCreatBrokerVocab,
            "_rCreatBrokerVocab", sessID);
        sessID = BookBrokerHelper.doReq(_rRegBookBroker, "_rRegBookBroker",
            sessID);
        sessID = BookBrokerHelper.doReq(_rCreatBookstoreVocab,
            "_rCreatBookstoreVocab", sessID);
        sessID = BookBrokerHelper.doReq(_rRegProxyBarnes, "_rRegProxyBarnes",
            sessID);
        sessID = BookBrokerHelper.doReq(_rRegProxyAmazon, "_rRegProxyAmazon",
            sessID);
        sessID = BookBrokerHelper.doReq(_rRegProxyFatBrain,
            "_rRegProxyFatBrain", sessID);
        sessID = BookBrokerHelper.doReq(_rLogout, "_rLogout", sessID);
    }
}
```