e-speak
Migration Guide

Developer Release 3.01
June 2000

HEWLETT® PACKARD

Contents

Chapter 1 Introduction

This document describes changes in the e-speak architecture for Developer Release 3.01. It provides a road map for programmers migrating from Beta 2.2 to the new Developer Release 3.0. For detailed documentation on the features, you can also review the following documents:

- *E-speak Programmers Guide* defines the interface for e-speak programmers and system developers building e-speak-enabled applications.
- *E-speak Installation Guide* shows how to install e-speak and how to run some simple applications.
- *E-speak Contributed Services* describes several sample applications included with the distributed software.
- *E-speak Tools Documentation* shows how to use tools provided for analyzing the system.

What's new in Developer Release 3.01

The new implementation of e-speak offers the following benefits and functionality for the users:

- Faster Time to Market because of easier API
- More flexible discovery
- Extended inter operability

Changes to the E-speak Engine

- Added secure dynamic firewall
- Faster and more flexible negotiations between multiples e-services for matching
- Increased robustness of mediation
- Better scaling of the In-Memory repository discovering resources and data bases
- Support for Microsoft SQL 7.0 added.

Client Library Programming APIs

- Greater ability for descriptive information (attributes) from any e-service application.
- The net e-speak infra_client package is being deprecated.
- Support of Multi-Valued Attributes.
- Support of multiple vocabularies.
- More control over advertisement of services by applications
- More flexible control over time-outs
- More robust methods for application of security policies.
- Improved support for detecting interfaces offered by an application

Java Client Library

- Improved exceptions reporting in the Service thread
- Comparison of two found services for equality in a multi-core situation.

- Access control and cryptographic security added.
- Clients can discover interfaces supported by the service to be accessed.
- API methods added to directly query whether a resource is local or exported
- Persistent Scopes added.
- Improved performance for larger number of services.
- Improvements to the IDL compiler

JESI

- Support for exporting external resources by Value added.
- Selective resource exporting added to reduce time delays.

Advertising Services

- Multicast model advertising services added
- Access permissions added advertised services
- Advertising services across communities now available.

Virtual File System

- Event Monitoring enhanced
- Cabinet View functionality added
- Semantic File System added.
- Support for generic storage services added.

System Management

- More robust set of system management control methods.

Web Access

- Greater flexibility in offering Web enabled services including WAP.

Current Feature Limitations

Interceptors are one way. They can only be used to intercept inbound messages and not the data returned from a service.

Intended Audience

This guide describes changes to the lower-level interfaces of e-speak for:

- Implementors of Client libraries to provide a higher level of abstraction for e-speak
- Implementors of utilities and tools to manage and manipulate e-speak
- Implementors of e-speak emulation routines that are used in the run-time environment for legacy applications
- Implementors of extensions to existing services and resources used by Clients
- System administrators who implement policies for security and resource lookup
- Those designing and building their own implementations of e-speak

Structure

This specification consists of the following major sections, in the order listed:

- An overview of the e-speak architectural changes
- A detailed description of the changes categorized by the areas that the changes affect.

Conventions

There are several document conventions worth noting:

- New terms are introduced in the document flow with italics.
- Programmatically visible architectural abstractions are written with the first letter of each word capitalized, such as Protection Domain.
- Logical names, method names, and other programmatic labels are written in `Courier` font.
- Even though e-speak is independent of the programming language, the specification uses Java syntax.
- Sections describing material outside of the architecture are denoted with the word “Informational” in the chapter or section title

Chapter 2 The E-speak Engine

This chapter describes changes to the e-speak Engine that are different in Developer Release 3.01.

Persistence

The database Schema has changed and is no longer comparable with the Version 2.2 Beta Schema.

It is necessary to use the Repository Reset Utility to remove the old repository and do a Cold Start to create the new repository with the proper schema.

Failure to do so generates a Core Panic with the message:

"Schema Version Mismatch"

Clients interact with the e-speak Core by sending messages to Core-managed Resources. For example, the Resource Factory registers new Resource metadata. This section specifies the methods of each Core-managed Resource. It also describes the internal state that is passed if the Core-managed Resource is exported by value to another Logical Machine.

Secure dynamic firewall

Support for secure dynamic firewall has been added, as well as support for firewall transversal.

See Chapter 4, “The E-speak Security Model” for more detailed information on this feature.

Support for Microsoft SQL 7.0

Developer Release 3.01 adds support for Microsoft SQL 7.0 Database and Oracle, which was supported in Beta 2.2.

Chapter 3 Jesi

This chapter describes the changes to Jesi. The first part of this chapter describes the added functionality. The second part explains functinality that is removed. The last part of this chapter explains the added APIs.

Added Functionality

Multiple vocabularies

You can register a service in m7ultiple vocabularies.

You can also do a search in multiple vocabularies.

Multivalued attributes

An attribute can have multiple values.

For example, a printer service can have an attribute "doctypes" which takes values of "pdf", "doc", and so on.

Interception

This is a mechanism by which you can intercept the calls to and from the engine. This can be used to build load balancers, caching, auditing mechanisms etc.

View

View is a snapshot of the repository. You can add or delete services to/from a view. View can be used for specifying a scope for a lookup.

Find

The following are enhancement to the Find interface.

- There is a cursor mechanism for retrieving a set of entries from the engine.
- You can define the query in multiple vocabularies.
- You can specify the sorting order and arbitration policy.
- You can specify a view (ESView) to restrict the scope of a lookup.

Threading policy

There are two modes of threading available:

- Specify a thread pool to be used for servicing requests
- Create a separate thread per request.

Categories

Security

Security is now available. See Chapter 4, “The E-speak Security Model” for an overview of the new Security features avaiable to Jesi.

Account manager

Account manager can create, delete and validate user accounts.

Faster restart

The time to effect a Restart is reduced.

One service handler can be associated with hundredsof services. Previously, Restart was accomplished by recreating all existing services.

Restart is accomplished by using the service handlers alone.

Removed Functionality

Persistent Scopes

Persistent scopes are removed from this release.

Transient scopes contain all the services created except the folders. This allows for some optimizations to enhance performance.

Added APIs

Class : ESAccessor

ESAttribute getAttribute(String, ESVocabulary)

Obtain an attribute with name and vocabulary specified.

ESBaseDescription getDescription(ESVocabulary)

Get all the attributes in a vocabulary

ESUID getESUID()

Get the id of the service. Used in events.

ESBaseDescription[] getDescriptions()

Get all the attributes in all the vocabularies for the service

ESVocabulary[] getVocabularies()

Get all the vocabularies for this service.

public int getServiceType()

Get the type of the service. The type is an int as specified in ESConstants.ServiceType. This method is not guaranteed to work always.

void setDescription(ESBaseDescription)

Set one of the descriptions of the service.

void setDescriptions(ESBaseDescription[])

Set all the descriptions of the service.

void setServiceType(int)

Sets the service type of the accessor. Use the types specified in ESConstants.ServiceType

void setServiceOwner()

Sets the current connection to be the owner of the selected service.

void setServiceProxy(ESServiceHandler)

Sets the service handler to act as proxy for this service.

Class : ESAccountManager

boolean addDescription(ESAccountProfile, String, ESProfileDesc)

Add a new description to the account that has been created.

boolean authenticateUser(ESAccountProfile)

Verifies whether the user password in the account matches with the supplied profile.

String createAccount(ESAccountProfile)

create an account with the specified profile

String[] getAllAccounts()

Get all the accounts that has been created.

boolean deleteAccount(ESAccountProfile)

Delete the account specified by the profile.

Class : ESAccountProfile

ESAccountProfile(String)

Create an account profile.

ESAccountProfile(String, String, String)

Another constructor for account profile

void setPassPhrase(String)

String getPassPhrase()

Set/Get the pass phrase for the account. The user can be validated later with this pass phrase using authenticateUser method on ESAccountManager.

void setPreferences(ESProfileDescription)

ESProfileDescription getPreferences()

Set/Get the profile description associated with the account.

void setUserESURL(String)

public String getUserESURL()

Set/Get the url associated with the account.

public void setUserInformation(String)

public String getUserInformation()

Set/Get any additional information associated with the account.

void setUserName(String)

public String getUserName()

Set/Get the user name associated with the account.

void setUserType(String)

public String getUserType()

Set/Get the type associated with the account.

Class : ESAttribute

boolean isMultiValued()

Check whether the attribute has multiple values.

Object[] getValues()

Obtain the array of values associated with this attribute.

void addValues(Object[] values)

Add a set of values to existing set

void setValues(Object[] values)

Replace the existing values with new set of values. Note that the change is made only in the local copy. To get it to the core, you need to do setDescription or a setAttribute on the accessor

Class : ESBaseDescription

ESBaseDescription(ESConnection, ESXMLFile)

Replaces the old method which had ESXMLFile as the first argument

void addAttribute(ESAttribute)

Add an attribute to the description

Class : ESBaseServiceStub

boolean conforms(String)

Check whether the object conforms to a particular interface.

String getStubInterfaceName()

Get the interface name associated with the service.

Class : ESCategoryFinder

ESAccessor findByFullName(String)

Return the category associated with the string.

ESCategory findRootCategory()

Find the base category.

ESAccessor[] find(String)

Return array of categories matching the string.

ESAccessor[] findByKeyword(String)

Return array of categories matching the string

Class : ESCategory

ESCategory createCategory(String, String)

Creation of a category

void setParentCategory(ESCategory)

Set the parent of this category

String getCategoryDescription()

Get the description of the category

String getCategoryFullName()

Return the full name of the category

String getCategoryName()

Return the base name of the category

ESAccessor[] getSubCategories()

Return all the sub categories of this category

void removeCategory(ESCategory)

Remove a sub category from this category.

void attachCategory(ESCategory)

Attach a category (!)

Class : ESConfiguration

void setAsyncRecvTimeout(int)

public int getAsyncRecvTimeout()

Set/Get the timeout specified for message reception.

void setAsyncSendTimeout(int)

public int getAsyncSendTimeout()

Set/Get the timeout specified for sending messages.

void setCallTimeout(int)

public int getCallTimeout()

Set/Get the timeout for method invocations.

void setFinderTimeout(int)

public int getFinderTimeout()

Set/Get the timeout for finders.

int getFinderCacheSize()

int getMaxReadThreads()

int getMinReadThreads()

int getMaxWriteThreads()

int getMinWriteThreads()

int getNumProcessorThreads()

int getReadQueueSize()

int getThreadIncrSize()

int getThreadPolicy()

Getters for various parameters.

Class : ESConnection

ESAccountManager getAccountManager()

Returns the account manager for the connection

ESRemoteConnectionManager getConnectionManager()

Return the manager for remote connections.

ESR**emoteServiceManager getRemoteServiceManager()**

Return the manager for remote services.

ESVocabulary getDefaultAcctVocabulary()

Return the vocabulary for creating accounts.

ESVocabulary getEvntDistDfltVocab()

Return the default vocabulary for event distributors.

void switchAccount(ESAccessor)

Switch to this account.

void init()

Initialize after switching the account.

Class : ESContractDescription/ESContract

void setConversationScheme(String)

public String getConversationScheme

Set/Get the XML documents/Schemas input/output to the service

void setTermsOfUse(String)

public String getTermsOfUse()

Set/Get the terms of use for this contract

void setLicense(String)

public String getLicense()

Set/Get the license agreements.

Class : ESFolder

ESFolder(ESAccessor)

Construct a folder with an accessor

ESBaseService getService(String, String)

Previously used to return ESService, now returns ESBaseService

Class : ESIceptorControl

boolean addIceptor(ESIceptor, Object)

Adds an interceptor to the chain of interceptors.

boolean removeAllIceptors()

Removes all interceptors from the chain

boolean removeIceptor(String)

Remove the iceptor identified by the name from the chain.

void setAccessor(ESAccessor)

public ESAccessor getAccessor()

Set/Get for the associated accessor

void setConnection(ESConnection)

public ESConnection getConnection()

Description :Set/Get the connection.

void setServiceElement(ESServiceElement)

public ESServiceElement getServiceElement()

Set/Get the service element associated with the control.

Class : ESProfileDescription

ESProfileDescription(ESVocabulary)

Constructor. Rest all methods inherited from ESBaseDescription

Class : ESProperty

API : ESProperty(String, String)

Constructor which takes in attribute name and type.

ESProperty(String, String, ESValue, boolean, int, int, int, int)

Construct a property with name, type, default value, multivalued or not, range kind, minimum range, maximum range, index.

Class : ESQuery

void addVocabularyKey(String, ESVocabulary)

Set this key to represent ESVocabulary in the constraint

void setArbitPolicy(int)

public int getArbitPolicy()

Set/Get the arbitration policy

void addSorter(String, String)

Add a sorting policy specified by the arguments.

void addSorter(int, String)

Add a sorting policy specified by the arguments.

void setView(ESView)

Add a view for searching.

Class : ESRemoteConnectionManager

String openConnection(String)

Open connection to the core specified by the argument.

void closeConnection(String)

Close connection to the core specified by the argument.

String[] getConnections()

Obtain all the connections established by the core.

void exportResource(ESAccessor[], String, int, boolean)

Export the specified services to the specified core.

Class : ESRemoteServiceManager

void importResource(ESAccessor, String, int, boolean)

Import the specified service from the specified core.

void unexportResource(ESAccessor, String)

Unexport the previously exported service

void updateExportedService(ESAccessor, String, int, boolean)

Update the exported service.

void updateImportedService(ESAccessor, String, int, boolean)

Update the imported service.

Class : ESRequest

ESRequest()

Descript9ion : Constructor for the class

ESRequest(String, String)

Constructor takes in an interface name and method name

ESRequest(String, String, ParameterList)

Constructor takes in an interface name, method name and parameter List

ESRequest(Object[])

Constructor taking in an array of objects.

void setParamValue(String, Object)

public Object getParamValue(String)

Set/Get a parameter value

void setReturnValue(Object)

public Object getReturnValue()

Set/Get return values

String getInterfaceName()

Return the interface name.

String getMethodName()

Return the method name.

Object[] getArguments()

Return the arguments as object array.

void setMarshallComplete()

All added params from this point are interceptor specific and are not passed when the method gets invoked.

addParam(String, Object, String)

Add a parameter (name, value, type)

Class : ESServiceContext

public void setCategory(ESCategory)

public ESCategory getCategory()

Set/Get the default category

Class : ESServiceElement

ESServiceElement(ESConnection, ESServiceDescription[])

Construct a service element with multiple descriptions.

ESServiceElement(ESServiceHandler)

Construct a service element with a handler.

boolean addIceptor(ESIceptor, Object)

Add an interceptor to the service handling loop.

boolean removeAllIceptors()

Remove all the interceptors in the chain

boolean removeIceptor(String)

Remove the interceptor identified by the argument.

ESIceptorControl getIceptorControl()

Return the control object associated with the iceptors.

ESServiceHandler getHandler()

Get the associated handler.

ESServiceDescription[] getDescriptions()

Get the associated descriptions.

void update(ESServiceDescription[])

Update the descriptions

Class : ESServiceHandler

ESServiceHandler (ESAccessor)

Create a service handler with an accessor. This accessor may have been retrieved from a folder.

ESAccessor getAccessor()

Return the ESAccessor of the handler.

Class : ESServiceStub

boolean addInterceptor(ESIceptor, Object)

Add an interceptor to the interceptor chain

Class : ESServiceStub

boolean removeAllInterceptors()

Remove all the interceptors in the chain

boolean removeInterceptor(String)

Remove the interceptor as specified by the argument

ESIceptorControl getIceptorControl()

Get the control object associated with the interceptors.

Class : ESValue

API : ESValue(BigDecimal)

ESValue(Date)

ESValue(Double)

ESValue(Long)

ESValue(String)

ESValue(Time)

ESValue(Timestamp)

ESValue(byte)

ESValue(char)

ESValue(double)

ESValue(float)

ESValue(int)

ESValue(long)

ESValue(short)

Used for creating vocabulary properties.

Class : ESViewDescription

ESViewDescription()

Default constructor

ESViewDescription(ESVocabulary)

Constructor which takes in a vocabulary

ESViewDescription(ESConnection, ESXMLFile)

Constructor which takes in a connection and xml description

Class : ESViewElement

ESViewElement(ESAccessor)

Constructor

ESViewElement(ESConnection, ESViewDescription)

Default constructor

ESView register()

Registers a view with the engine.

Class : ESViewFinder

ESViewFinder(ESConnection)

Constructor

ESView find(ESQuery)

find one view

ESView find(ESQuery)

Find multiple views matching the query

ESView[] findNext()

Return the next set of entries.

Class : ESViewStub

boolean contains(ESAccessor)

Check whether the view contains the service

void add(ESAccessor)

Add an accessor to the view

void clear()

Clear the view

void remove(ESAccessor)

Remove the service from the view

Class : ESVocabularyDescription

ESVocabularyDescription(ESConnection, ESXMLFile)

Replaces the constructor where the parameters to the method were ESXMLFile and ESConnection in that order.

ESVocabularyDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

Class : ESServiceDescription

ESServiceDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

ESViewDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

Class : ESContractDescription

ESContractDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

Class : ESVocabularyDescription

void addProperty(ESProperty)

Add a vocabulary property as specified by the argument.

Class : ESVocabularyFinder

ESVocabulary[] findNext()

Find the next set of vocabulary in the list.

Class : ESVocabularyStub

ESAccessor[] getServices()

Return all the services in the vocabulary

ESProperty[] getProperties()

Return all the properties in the vocabulary

Class : ESXMLQuery

ESXMLQuery(ESConnection, ESXMLFile, ESXMLFile)

To support the new xml schema

Class : ESAbstractElement

void addPrivateData(String, byte[])

Moved from ESBaseDescription

void addPublicData(String, byte[])

Moved from ESBaseDescription

Class : ESAbstractElement

void setContract(ESContract)

Moved from ESBaseDescription

Class : ESAbstractFinder

void setSecurityLevel(boolean)

public boolean setSecurityLevel()

Set/Get the trust level. Based on this, a decision is made whether to obtain the stub from the other end. By default, stub is picked from the local address space.

boolean hasMoreResults()

Check whether there are any outstanding results.

void setMaxToFind(int)

public int getMaxToFind()

Set/Get the maximum number to find.

void setCacheSize(int)

Setthe cache size

void setFindInCacheFlag(boolean)

Specify whether to use cache or not.

void setInterfaceVerificationFlag(boolean)

Specify whether the check for whether the correct interface is supported by service should be made or not.

Class : ESAbstractFinder

void setSearchLevel(int)

Specify whether the search is in local core or to the advertising service.

Class : ESDelegatorImpl

abstract Object getHandler()

Return the object which ultimately handles the request

abstract void handleDone(Object)

Inform the delegator that reply has been sent back.

Class : ESIceptor

public ESIceptor()

Constructor for the class

abstract void initialize(Object)

To be implemented by the sub-classes. This is called by the interceptor control when the interceptor is initialized first.

boolean invokeNext(ESRequest)

To be called by all the sub-classes, invokes the next interceptor in the chain.

Class : ESServiceHandler

ESServiceHandler (ESAccessor)

Create a service handler with an accessor. This accessor may have been retrieved from a folder.

ESAccessor getAccessor()

Return the ESAccessor of the handler.

Class : ESServiceStub

boolean addInterceptor(ESIceptor, Object)

Add an interceptor to the interceptor chain

boolean removeAllInterceptors()

Remove all the interceptors in the chain

boolean removeInterceptor(String)

Remove the interceptor as specified by the argument

ESIceptorControl getIceptorControl()

Get the control object associated with the interceptors.

Class : ESValue

ESValue(BigDecimal)

ESValue(Date)

ESValue(Double)

ESValue(Long)

ESValue(String)

ESValue(Time)

ESValue(Timestamp)

ESValue(byte)

ESValue(char)

ESValue(double)

ESValue(float)

ESValue(int)

ESValue(long)

ESValue(short)

Used for creating vocabulary properties.

Class : ESViewDescription

ESViewDescription()

Default constructor

ESViewDescription(ESVocabulary)

Constructor which takes in a vocabulary

ESViewDescription(ESConnection, ESXMLFile)

Constructor which takes in a connection and xml description

Class : ESViewElement

ESViewElement(ESAccessor)

Constructor

ESViewElement(ESConnection, ESViewDescription)

Default constructor

ESView register()

Registers a view with the engine.

Class : ESViewFinder

ESViewFinder(ESConnection)

Constructor

ESView find(ESQuery)

find one view

ESView[] findAll(ESQuery)

Find multiple views matching the query

ESView[] findNext()

Return the next set of entries.

Class : ESViewStub

boolean contains(ESAccessor)

Check whether the view contains the service

void add(ESAccessor)

Add an accessor to the view

void clear()

Clear the view

void remove(ESAccessor)

Remove the service from the view

Class : ESVocabularyDescription

ESVocabularyDescription(ESConnection, ESXMLFile)

Replaces the constructor where the parameters to the method were

ESXMLFile and ESConnection in that order.

ESVocabularyDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

Class : ESServiceDescription

ESServiceDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

Class : ESViewDescription

ESViewDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

Class : ESContractDescription

ESContractDescription(ESConnection, ESXMLFile, ESXMLFile)

Constructor to support the new xml schema.

Class : ESVocabularyDescription

void addProperty(ESProperty)

Add a vocabulary property as specified by the argument.

Class : ESVocabularyFinder

ESVocabulary[] findNext()

Find the next set of vocabulary in the list.

Class : ESVocabularyStub

ESAccessor[] getServices()

Return all the services in the vocabulary

ESProperty[] getProperties()

Return all the properties in the vocabulary

Class : ESXMLQuery

ESXMLQuery(ESConnection, ESXMLFile, ESXMLFile)

To support the new xml schema

Class : ESAbstractElement

void addPrivateData(String, byte[])

Moved from ESBaseDescription

void addPublicData(String, byte[])

Moved from ESBaseDescription

void setContract(ESContract)

Moved from ESBaseDescription

Class : ESAbstractFinder

void setSecurityLevel(boolean)

public boolean setSecurityLevel()

Set/Get the trust level. Based on this, a decision is made whether to obtain the stub from the other end. By default, stub is picked from the local address space.

boolean hasMoreResults()

Check whether there are any outstanding results.

void setMaxToFind(int)

public int getMaxToFind()

Set/Get the maximum number to find.

void setCacheSize(int)

Setthe cache size

void setFindInCacheFlag(boolean)

Specify whether to use cache or not.

void setInterfaceVerificationFlag(boolean)

Specify whether the check for whether the correct interface is supported by service should be made or not.

void setSearchLevel(int)

Specify whether the search is in local core or to the advertisingservice.

Class : ESDelegatorImpl

abstract Object getHandler()

Return the object that ultimately handles the request

abstract void handleDone(Object)

Inform the delegator that reply has been sent back.

Class : ESIceptor

ESIceptor()

Constructor for the class

abstract void initialize(Object)

To be implemented by the sub-classes. This is called by the interceptor control when the interceptor is initialized first.

public boolean invokeNext(ESRequest)

To be called by all the sub-classes, invokes the next interceptor in the chain.

Depricated APIs

The following APIs are depricated and most will be removed with the next release:

Class : ESAccessor

public ESAttribute[] getAttributes()

The attributes can belong to multiple vocabularies. Use getDescriptions() method instead to get the vocabulary information also.

public Object getAttribute(String)

This returns the value of the attribute. Use getAttribute(String, ESVocabulary) instead

void setAttributes(ESAttribute[] list, ESVocabulary vocab)

Use setDescription(ESBaseDescription) method instead.

Class : ESBaseDescription

AttributeSet getAttributeSet()

This method is intended for internal use. This method will be removed in the next version.

Class : ESConnection

ESConnection()

This constructor connects to default port and default host. Use ESConnection(Properties) for better flexibility.

ESConnection(String, int, String)

Use ESConnection(Properties) for better flexibility

ESShell getShell()

This method should not be used by application developer. This is used for internal purposes. This API will be removed in the next version.

Class : ESContractDescriptior

void setInterfaceDefinition(String)

For better platform portability, use setInterfaceDefintion(byte[])instead.

Class : ESFolder

public ESFolder(ESConnection, ESServiceDescription, boolean)

Use ESFolder(ESConnection, ESServiceDescription, String, boolean) for better control on folder names.

public void setName(String)

Suggested use is to do a getParent() and do a rename()

Class : ESQuery

ESQuery(ESVocabulary)

Use ESQuery(ESVocabulary, String)

SearchRecipe getRecipe()

This method is used for internal manipulations. This methodwill be removed in the next version.

String getConstraint()

This method is used for internal manipulations. This methodwill be removed in the next version.

Class : ESServiceContext

ESCommunity getCurrentCommunity()

Use getCommunity() instead

Class : ESServiceElement

ESServiceElement(ESConnection, String)

Use ESServiceElement(ESConnection, ESServiceDescription) instead.

void update(ESServiceDescription)

Use update(ESServiceDescription[]) instead.

Class : ESServiceMessenger

Object[] sendSynchronous(Object)

Method for internal use. Use public Object sendSynchronous (ESAccessor, Object) instead.

Class : ESVocabularyDescription

ESVocabularyDescription(ESXMLFile, ESConnection)

For sake of consistency, this is changed to ESVocabularyDescription (ESConnection, ESXMLFile).

Chapter 4 The E-speak Security Model

This chapter describes the new Security features in e-speak Developer Release 3.01.

E-speak security implements a global distributed single sign on using Public Key Infrastructure.

All entities in e-speak (users, services, cores etc.) now need to be identified by public keys. Verification takes place by insuring the entity knows the private key corresponding to the public key given.

Any entity can generate a key pair. For services to be used, a certificate (or certificates need to be obtained, granting access rights to the services

Certificates

E-speak certificates are signed (authenticated statements) linking the public key to a name or attribute, which typically states the access rights.

To make an access control decision, a service does the following

- Examine the attribute in the certificate to see what access rights it grants
- Checks the entity making the request know the corresponding private key.
- Verifies the certificated has been issued (is signed by) and entry it trusts.

E-speak certificates are issued by Certificate Issuers (CI) which can issue two kinds of certificates.

- Name Certificates
- Attribute Certificates

E-speak implements a split trust model. A service can trust a CI to issue certificates to only a subset of the services operations and the services does not have to trust every CI.

Simple Public Key Infrastructure

E-speak is an implementation of Simple Public Key Infrastructure (SPKI). It specifies a structure and a set of operations on attributes and name certificates.

These attributes are called tags and are further explained in the section on Jesi.

Certain tags E-speak ***Tags*** are defined and checked for e-speak operations.

Service Identity

The serviceID is intended for use by applications to identify services without using the resource name or access path (ESNames)

The serviceId field is a resource tag used to identify the service to the resource handler.

If no serviceID is specified, the core substitutes a default serviceId.

The benefits of using a ServiceId are as follows:

- Services ESNames can be changed without affecting authorization.
- Authorization can be revoked by changing serviceId without changing ESName.
- Service identity can be managed independently of persistence
- In a replicated service, replicas can all have the same identity.

Undeliverable Requests

New in Developer Release 3.0 is that this condition is ignored for security reasons and no exception is thrown. Instead applications must wait for a time out.

Allowing unauthenticated errors to be acted on allows for denial of service attacks on the e-speak core.

JESI

Resource masks

The default behavior when security is enabled is to require authorization for all operations. The masks are used to control this.

If a mask is set, operations matching the mask are permitted whether the requestor is authorized or not.

There are two masks:

- Metadata mask
- Resource mask

Masks are specified as tags.

Tags

Tags are given in BNF format.

The basic method tag format is

```
(net.espeak.method <interface name> <method name>)
```

In the metadata mask, the interface name is the core interface being specified, and the method name is the operation in that interface.

For metadata, the interface is likely to be ResourceManipulationInterface, and the method name one of its methods.

In the resource mask for a JESI service, the interface name is the fully-qualified name of the interface class.

The method name is the name of the method in the interface, plus the concatenated argument types. This allows overloaded methods to be distinguished.

The metadata mask is used by the in-core meta resource when performing metadata operations.

The resource mask is passed to the service handler by the core for the service handler to use when performing operations on the service itself.

The masks are completely general tags, so the mask tag itself, or any of its fields, may use the tag matching features such as sets, prefixes and ranges. The interface and method names, for example, do not have to be string literals, they can be sets or prefixes.

This tag masks method foo in interface net.espeak.examples

```
ExampleIntf:(net.espeak.method net.espeak.examples.ExampleIntf foo)
```

This tag masks all methods beginning with foo:

```
(net.espeak.method net.espeak.examples.ExampleIntf (* prefix foo))
```

This tag masks methods foo and bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf (* set foo bar))
```

Methods with prefix foo or bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf
(* set (* prefix foo) (* prefix bar)))
```

All methods in the interface:

```
(net.espeak.method net.espeak.examples.ExampleIntf)
```

This is equivalent to

```
(net.espeak.method net.espeak.examples.ExampleIntf (*)) since
missing trailing elements match anything.
```

Methods foo in Interface A and bar in Interface B:

```
(* set (net.espeak.method Interface A foo)
(net.espeak.method Interface B bar))
```

All methods:

```
(net.espeak.method)
```

or simply

```
(*)
```

When we introduced method tags above, we skipped some details. The full form of the method tag is actually:

```
(net.espeak.method <interface name> <method name> <service>)
```

In the normal case, the service handler is only interested in its own operations, so it does not care what the service field is. Since omitting a trailing field is equivalent to giving it the value (*), we omitted this detail above.

When a message invoking an operation is received, the service handler extracts the interface and method from it, and gets the service identifier from the information passed to the handler by the core. The service handler then constructs a method tag using this data and queries the service authorizer to see if the tag is authorized.

The authorizer first checks to see if the tag matches the resource mask, and if it does, the operation is permitted. If the tag does not match the resource mask, the authorizer uses the current security session to see if the tag is authorized.

Normal tag matching rules are used throughout, which is why we were able to omit the service part of a mask tag above.

General tags can be constructed using the following method in ESSecurityEnv:

```
ADR createTag(String s) throws IOException
```

The IOException subclass net.espeak.security.adr.ADRParseException is thrown on a parse error.

The parameter s is a string containing the input syntax for the tag.

Method tags can be created using ADR createMethodTag(String interfaceName, String methodName, ADR service)

For the purposes of resource masks, it is usual to use a tag containing simply (*) as the service parameter.

> **NOTE:** In advanced applications the service may want to set the service parameter to its service id, but this is not necessary.

After a mask tag has been constructed, it is used in ESAbstractElement methods:

```
void setResourceMask(ADR tag) throws ESException
void setMetadataMask(ADR tag) throws ESException
```

Before a service is registered these simply affect the local state. After registration, these set the local state and update the service metadata.

Masking can be turned on or off using ESAuthorizer:

```
void setMasking(Boolean x)
```

When masking is off, the resource mask is ignored by the service authorizer even if set.

> **NOTE:** Setting masking off in the authorizer has no effect on the resource metadata, or the in-core meta resource handling metadata operations.

Masking can be turned off completely, in the core and handler, by setting a mask to null.

ESConnection has methods for controlling the default resource and metadata masks used when services are registered:

```
void setDefaultResourceMask(ADR mask)
ADR getDefaultResourceMask()
void setDefaultMetadataMask(ADR mask)
ADR getDefaultMetadataMask()
void setMasks(ADR metadataMask, ADR resourceMask)
```

After a default mask is set, all resources registered use it until it is changed.

NOTE: Unless the default masks are set explicitly, ESConnection use null for them, causing authorization to be checked for all operations.

XML

Chapter 5 Management

Two concepts that underpin the manageability of e-speak Resources and e-speak-Clients.

- Managed State: a defined service state embodying the life cycle of a service.
- Managed Variable Table: sets of values that can be affected by a manager for the purposes of configuration and control.

State Descriptions

Initializing

The internal dynamic state of the service is being constructed, for example: a policy manager is being queried for configuration information and resources are being discovered via search recipes or yellow pages servers. When the service finishes this work, it moves asynchronously into the ready or error states.

Ready

The service is in a ready to run situation, this state is also equivalent to a stopped or paused state.

Running

The service is running and responding to methods invoked on its operational interfaces. If an error occurs which implies that the service cannot continue to run it should move into the error state.

Error

The service has some problem and is awaiting management action on what to do next.

Closed

The service has removed/deleted much of its internal state and awaits either a cold-Reset or remove transitions.

Inputs

An input is the trigger that causes a state transition to occur. In any given state there is a defined set of permissible inputs that are available, i.e. only those that are depicted in the diagram as leaving the current state and connecting with the next state. To attempt to perform any other transition is illegal. Note that many inputs can have the same name (e.g. error) yet there is no ambiguity as long as the originating state is different.

Clients can provide any input with impunity. However a management agent can request only provide external inputs. For example, the manager might reasonably request that a client perform a warm reset, but not to become ready, the client alone can provide this input, i.e. when it's internal initialization process has completed.

The available inputs are as follows.

- start: move into the running state. Start to handle invocations on operational interfaces.
- stop: move into the ready state. Stop handling invocations on operational interfaces.
- ready: move into the ready state having finished initialization.
- error: move into the error state, this transition is valid from any state.
- shutdown: clean up any internal state required and move into the closed state. This transition should not cause the deregistering of resources from the repository.

- coldReset: cause a from complete reinitialization of the service and move into the initializing state. The only exemption is that resources that are already registered should not be reregistered.
- warmReset: cause a partial reinitialization of the service i.e. retaining some of the existing service state move into the initializing state.
- remove: cause the service to remove itself from existence. Any non-persistent resources should be deregistered from the repository.

Managed Variable Tables

A managed variable table is at it's simplest a table of name/string value pairs that exist within the client but to which a manager has some level of access. Thus a management agent can control those aspects of a services behavior that is affected by those variables to which it has access.

There is a degree of configurability associated with managed variables and their variables that permit something more sophisticated than the simple get and set operations one would expect to find.

Each table itself has a name to distinguish it from other tables. As we shall see later, the managed service model itself provides for two such tables.

There is a restriction on variable table usage: each name in a variable table must be unique within that table. It is not possible to implement lists by having many entries with the same name.

Configuration Parameter Table

The configuration parameter table is an instance of a managed variable table with a reserved name that identifies it as such. The table holds generic configuration data for the client.

Resource Table

The resource table is another instance of a managed variable table, identical in behavior to the configuration parameter table except that the names in the client's table refer to other services with which the client has some relationship. For example, if a particular client makes use of a mail service then this relationship can be made visible to a management agent through the resource table. Thus a management agent might reconfigure the client to use an alternative but equivalent service. While there might seem no obvious need to separate out this particular aspect of configuration, doing so makes it possible for a management agent to discover the topology and integrity of a network of connected services without the need for service specific interpretation of the variable table (all entries in the resource table are resources).

The name used for an entry in a resource table can be any symbolic name the client chooses, while the value must be the valid e-speak ESName of the actual service.

Managed Service Interface

All e-speak Resources that are manageable implement the ManagedService interface. This applies whether the Resources are external to the e-speak Core, or Core-managed.

```
interface ManagedServiceIntf{
   String getName()
      throws ESInvocationException;

   String getDescription()
      throws ESInvocationException;

   String getOwner()
      throws ESInvocationException;

   String getUptime()
      throws ESInvocationException;
```

```
String getVersion()
   throws ESInvocationException;

String getErrorCondition()
   throws ESInvocationException;

String getStaticInfo()
   throws ESInvocationException;

void coldReset()
   throws IllegalStateTransition,ESInvocationException;

void warmReset()
   throws IllegalStateTransition,ESInvocationException;

void start()
   throws IllegalStateTransition, ESInvocationException;

void stop()
   throws IllegalStateTransition, ESInvocationException;

void shutdown()
   throws IllegalStateTransition, ESInvocationException;

void remove()
   throws IllegalStateTransition, ESInvocationException;

int getState()
   throws ESInvocationException;

VariableEntry[] getVariableEntries()
   throws ESInvocationException;

String[] getVariableNames()
   throws ESInvocationException;
```

```
VariableEntry getVariableEntry(String name)
   throws ESInvocationException, NoSuchVariableName;

void setVariable(String name, String value)
   throws ESInvocationException;

ResourceEntry[] getResourceEntries()
   throws ESInvocationException;

String[] getResourceNames()
   throws ESInvocationException;

ResourceEntry getResourceEntry(String name)
   throws NoSuchVariableName, ESInvocationException;

void setResource(String name, ESName resource)
   throws ESInvocationException;
}
```

The method getName return String containing the service name. This name should be used when registering the service resource in the service vocabulary.

The method getDescription returns a human readable description of the service for display on a management console.

The method getOwner returns a string indicating the owner of the service.

The method getUptime gets the time for which the service has been running. The format of the string is “years.days.hours.minutes.seconds”.

The method getVersion returns a string indicating the version of the service.

The method getErrorCondition returns a string indicating the error condition. This returns null if the service is not in an error state.

The method getStaticInfo returns an XML document of the following form.

```
<staticInfo>
<name>the name of the resource </name>
<owner> the name nameof the onwning service </owner>
```

```
<description> the decription here </description>
<version> the version string </version>
<uptime> the uptime string </uptime>
</staticInfo>
```

The coldReset transistion function cause the service to move into the initializing state and completely reinitialize. The exception IllegalStateTransitionException is thrown if the state is not in the ready, error or closed states.

The warmReset transistion function cause the service to move into the initializing state and partially reinitialize.The exception IllegalStateTransitionException is thrown if the state is not in the ready or error states.

The start transistion function cause the service to move into the running state and service client requests. The IllegalStateTransitionException exception is thrown if the state is not in the ready state.

The stop transistion function cause the service to move into the ready state and stop serving client requests. The exception IllegalStateTransitionException is thrown if the state is not in the running state

The shutdown transistion function clean up any internal state required and move into the closed state. This transition should not cause the deregistering of resources from the repository. The exception IllegalStateTransitionException is thrown, if the state is already in the closed state.

The remove transition function causes the service to remove itself from existence. Any non-persistent resources should be deregistered. The exception IllegalStateTransitionException is thrown if the state is not in the closed state.

The method getState return the current state: an integer value from 0 to 4.

The value returned is interpreted as follows.

- Initializing(0) - the service is constructing its internal data structures and finding other services which is needs to function.
- Ready(1) - the service is fully constructed and ready to run.
- Running(2) - the service is running and handling methods on its operational interfaces.

- Closed(3) - the service has deleted much of its internal state and closed any open connections to files or other services.
- Error(4) - The service has encountered an error preventing the service from continuing to operate.

The Variable table

Each manageable Resource maintains a table of name value pairs, which contains whatever information that Resource wishes to expose to the management agent. The table entries can be either read only or read write.

```
class VariableEntry {
   String name;
   String value;
   int updateType;
}
```

The updateType is interpreted as follows (**Dave Stephenson needs to provide this information - E-mail sent awaiting response.**)

The method getVariableEntries returns the table as an array of VariableEntry's. Each VariableEntry object contains the name, the value & update information.

The method getVariableNames returns an array of strings - one element in the array for each variable.

The method getVariableEntry returns the entry in the table for variable identified in the parameter name.

The method setVariable sets the variable identified by the parameter name to the string in the value parameter.

The Resource Table

The managed Resource maintains a table of name-Resource pairs. This table contains all the Resources that the element depends on i.e. uses. The table entries can be either read only or read write.

```
class ResourceEntry {
   String name;
   ESName resource;
   int updateType;

}
```

The method getResourceEntries returns the table as an array of ResourceEntry. Each entry contains a string that name for the resource, the ESName of the resource (URL) and the update information. The updateType is interpreted as **(Need information from David Stephenson. Mail sent awaiting response.)**

The method getResourceNames returns an array of strings, one element for each entry in the resource table.

The method getResourceEntry(String name) returns the entry in the table for the named resource.

The method setResource sets the Resource identified by the name parameter to the ESName supplied in the resource parameter.

Chapter 6 Web Access

This chapter briefly describes the new Web Access feature which allows programmers to construct HTML and SSL interfaces to the e-speak engine.

Web Access enables users to interact with the e-speak core or services through standard web browsers, by returning HTML or XML documents in HTML or XML

Web Access is based on XML (Extensible Markup Language)

Web access provides the interface for e-speak to standard environments in the Internet and XML-based e-Services solutions. Of particular interest are:

- Provide access to e-speak services through standard web browsers,
- Enable e-speak services acting as services in the web (web services),
- Allowing invocation of standard, non-e-speak enabled web services from e-speak clients,
- Provide access from and to XML-based e-speak services,
- Allow e-speak services to interact based on the XML document exchange model using various transports (HTTP, TCP, VPN connections).

The architecture of Web Access is defined in the Web Access Architecture Document. Web Access internally uses "e-speak XML"[1] to represent "content".

The term **"content"** refers here to all kinds of information related to e-speak, processed by Web Access and the e-speak core. It captures functions of the e-speak core.

Examples are search queries in order to find e-speak services, vocabulary descriptions, e-speak management information, service invocations and results passed back to requesting services and so forth.

1 "e-speak XML" refers to XML in accordance with the e-speak DTD/Schema definition

"Content representation" (encoding) is different at different stages in the system, and content needs to be transformed to interface with external systems.

The connection points to external systems are referred to as **"adapters"** (inbound) and **"agents"** (outbound).

For instance, when a user requests a service discovery through a web browser, this query arrives in Web Access as a HTTP FORM POST request. This representation of the query content needs to be transformed into internal e-speak XML for further processing.

Reversely, the result represented in e-speak XML needs to be transformed into HTML as expected by the browser an sent back in the HTTP reply message. Primarily for the browser interface, content needs to be presented visually. **"Content presentation"** is a special kind of a transformation and is based on **XSL style sheet transformations**.

Which Interface should I use?

Often, the first question to arise is whether to use the Web Access (XML) base interface or Jesi (Java) based interface. Both provide similar functionality but with totally different paradyms.

The **Java Model** is oriented toward traditional API interfaces.

Services are described by having an API or a set of APIs.

The client can make calls to discover services, retrieve a stub object and then invoke the services. These are typically synchronous methods with calls to methods producing results which the client waits on.

The **XML Model**, on the other hand, is a document based interface that is fundamentally asynchronous.

Services are described not by a set of APIs, but by a Schema which describes a set of XML documents which those services can understand.

To find a service, a document defining the query for Services is sent to Web Access which then returns a document describing the Services which fit the query criteria.

Considerations

Computational Services

Computational Services fit well with the API style (Java) Model. For instance, the Virtual File System is based on the Java model and exposes a core set of functional methods (Read, Write, Open, Close) which can be invoked by a client.

Business Services

Informational, business or broker type services fit well with the document mode. For instance the Book Broker Example included in the Appendices is a perfect example of a service that is ideally suited to Web Access.

Interface Knowledge

The API Model typically assumes that the programmer has knowledge of the exact interface at programming time, usually through importing the IDL definitions at compile time to generate the stubs needed. This means that the interface must remain immutable though the life of that version of the client. If the interface changes or is extended, the clients must be recompiled to handle or take advantage of the changes.

Changes or extensions in the document model can be discovered by the Client when it downloads the Schema. On the one hand the document model requires some additional effort in parsing the Schema and handling different formats for documents, but on the other hand this allows greater flexibility for the Client software since it is possible to handle a wider range of changes with recompiling.

Migrating Java to XML

Considerations when Migrating an e-speak 2.2 client or server to Web Access are mapping the Java APIs that were used to the schema.

Since all the APIs are exposed, it is actually possible to create an XML document that caused the Java APIs to be called and the results passed back as another document. This is not really using the attributes of XML, however it allows you more flexibility because you are not tied to the IDL compiler, but can change your generated document if the API interface changed. Unfortunately, it requires you to still know all the interfaces and their parameters.

The pure document approach requires that you only know the schema and create documents that refer to the schema.

In some cases, a service can publish its schema and you create an XML document to request the Schema. This can be helpful if you find several services that satisfy your query and they have different schemas.

Finally, keep in mind the asynchronous nature of XML. Not every request returns a document, In Java, each request returns some value which leads to all synchronous operations. Web Access only returns a document if the originating document generates a request that requires it.

XML Architecture

XML offers a robust solution as the underlying architecture for data in three-tier architectures. XML can be generated from existing databases using a scalable three-tier model. With XML, structured data is maintained separately from the business rules and the display. Data integration, delivery, manipulation, and display are the steps in the underlying process as summarized in the following diagram.

Three-tier Web architecture for flexible Web applications.

XML TYPE Layer

This layer provides the interface for mapping “arbitrary” XML into e-speak XML following the e-speak DTD/Schema. The Content Transformation Layer transforms content into XML. For cases, where content is already represented in XML, which is not e-speak XML, the XML TYPE Layer needs to transform this XML into e-speak XML by performing some transformation. These cases occur when e-speak inter-

acts with other XML-based systems such as BizTalk. The Content Transformation Layer only extracts XML from messages, and because content is already represented in XML, not apply any further transformation.

XML TYPE transformations are not within the scope of this document.

Functional Definition of E-speak Content

Content refers to e-speak functions accessible by services connecting to e-speak. Currently supported functions are:

- login
- logout
- **register a service** (based on a description and a URL)
- **unregister a service** (based on a description or URL)
- **register / unregister** a user
- **discover services** (according to a description delivering all attributes and values)
- **define a vocabulary** (based on a list of attribute [name, type] pairs)
- **discover a vocabulary** (based on a description in the default vocabulary delivering all attributes and types)
- **remove a vocabulary** (based on description using the default vocabulary or URL)
- **invoke a service** (based on an API description)
- (**send** an asynchronous message?).

Based on these basic functions e-speak provides, content representations can be defined:

- in e-speak **XML** (from an e-speak service, or to an e-speak service)
- in general **XML** (for interaction with other XML-based services or frameworks)

- for **web browser** interaction (HTML FORM -- FORM POST request -- HTML result)
- from inbound HTTP services using **FORM POST** requests
- for interaction with a **standard web site** (send HTTP request -- get HTML result)
- for **interaction** from an e-speak web servers

There is a representation matrix based on these dimensions.

	login	logout	register	unregister	discover	define vocabulary	remove vocabulary	invoke service	
e-speak XML	X	X	X	X	X	X	X	X	
web browser	X	X	X	X	X	X	X	X	
inbound HTTP POST requests	X	X	X	X	X	X	X	X	
outbound HTTP requests								X	

Figure 1 Content Representation Matrix

The content representation matrix also defines transformations which are possible to be done. If a transformation is requested not being defined in the content representation matrix, an **TransformationException** is thrown by the transform() method in the Transformation classes.

```
interface ContentTransformationIntf {

      Content transform(Content ct) throws TransformationException;

}
```

Chapter 7 Virtual File System

This chapter describes changes to the Virtual File System which entail the addition of the Semantic File System, Generic Store services and Security.

In addition, many minor enhancements have been incorporated to augment the robustness and functionality of the Virtual File System.

The Semantic File System

Addition of the SFSFolder. The Semantic File System folder allows the caller to display a subset of the files in their cabinets based upon the semantics of the files. By this we mean that a SFSFolder specifies a constraint to describe which files should appear in that folder.

For example, a constraint might be of the form:

"all files modified since May 5th, 2000 greater than 1000 bytes long"

When this folder is opened, only files that match this constraint appear in the folder.

This becomes more interesting if there are more attributes associated with each file. There are two new mechanisms for doing this.

The first is the file store will lookup a service that can "filter" the file type being processed. For example, if the file is a Microsoft Word Document, the file store looks for a service that processes Word Documents. If it finds one, it passes the VFS file to this service for processing.

Included in this release is an example filter service for Microsoft Office documents which extracts the Microsoft provided properties of the office document. Attributes such as Last Author, Page Count, etc., and extracted and stored as attributes to the VFS file.

Now, using the SFSFolder you can create more interesting constraints such as:

"all files written by "my name" having more than 10 pages".

The second mechanism is for the application to associate a custom vocabulary and attribute set with each file. This can be done through the standard J-ESI interfaces to add attributes to a resource.

The VFS Browser uses this approach to allow the user to create a private vocabulary and set attributes within this vocabulary.

See the SFSFolder class in the package vfs.clientapi.

Generic storage services

The original VFS was a Virtual File System.

But often the issue arises where users wanted to store more generic "documents" into the same folder structure provided by VFS?

The solution was to provide set of abstract classes with the necessary interfaces to communicate with the Virtual File System.

- VFSObject
- VFSStore

We've also created concrete classes which extend these abstract classes to re-implement the **File** and **FileStore** objects. But now you can use these as examples for creating your own store and objects.

For example, you may want to expose records in a database using VFS.

You can create a new store which manages the rows in a table and a new Row class that manages the contents of the row. The package **vfs.server** now contains the abstract classes and **vfs.server.filestore** contains the implementation for the file store service.

Security

VFS now uses the security features of e-speak to control access to folders and objects.

Refer to the e-speak security document for more details on the underlying security mechanisms.

Additional Enhancements

There are many smaller enhancements to VFS to enhance functionality and increase the robustness of the system

Please refer to the code or the Contributed Services Document chapter on VFS for more detailed information.

- The VFS Browser has gone through more rigorous testing. It is still largely a demonstration application, but it should fail less often.
- The service interface classes are not fully ported to using the esidl compiler. The value-added functions have been moved into concrete classes under the **vfs.clientapi** package. All the other clientapi classes now return these new concrete classes. So, where in Beta 2.2 the VFSFolder.getFiles returns **VFSFileStub** classes it now returns **VFSObject** classes. VFSObject still extends VFSObjectIntf, but adds all the non-service provided methods. *Note that the change from VFSFile to VFSObject is to reflect the generic store change discussed above.*
- The VFSShell has undergone a complete rewrite to simplify the implementation and provide more Unix-like command syntax.
- The cabinet view has been re-instated now that ESView support has been added to the J-ESI library.
- The management interfaces have been updated to support the new management interface.
- XAM trace points have been added to the code base to track performance

Chapter 8 Glossary

Terms used in E-speak

The following is a quick reference of terms commonly used in referring to e-speak Services, applications and APIs.

Term	Meaning
Advertising Service	A service for looking up resources not registered in the local Repository. It returns zero or more Connection Objects.
Arbitration policy	A specification within the search request accessor for naming that provides the logic to resolve multiple matches found for a name search.
Attribute Vocabulary	See **Vocabulary**.
Base Vocabulary	A Vocabulary provided at system start-up.
Builder	An entity identified by a Remote Resource Handler that is used to construct the internal state of a Resource imported by value.
Certificate	A data structure assigning a Tag or name to a Subject. Certificates are signed using cryptographic techniques so they cannot be tampered with.

Term	Meaning
Certificate Issuer (CI))	A service issuing certificates to Subjects.
Client	Any active entity (e.g., a process, thread, service provider) that uses the e-speak infrastructure to process a request for a Resource.
Client library	The interface specification that defines the interface for e-speak programmers and system developers that will build e-speak-enabled applications.
Connection Manager	A Logical Machine's component that does the initial connection with another Logical Machine.
Contract	See **Resource Contract.**
Core	The active entity of a Logical Machine that mediates access to Resources registered in the local Repository.
Core Event Distributor	A Core-managed Resource whose purpose is to collect information on e-speak Events and make such information available to management tools within the infrastructures.
Core-managed Resource	A Resource with an internal state managed by the Core.
Distributor Service	A service that forwards published Events to subscribers.
Event	A message that results in the recipient invoking a registered callback.

Term	Meaning
Event filter	A subscription specification expressed as a set of attributes in a particular Vocabulary that must match those in the Event state in order for a Client to receive notification on publication of an Event.
Event state	A reference within an Event to its expressed set of attributes in a particular Vocabulary. These attributes must match the Event filter in order for the subscriber to receive notification of the Event.
Explicit Binding	An accessor that contains a Repository Handle.
Import Name Frame	A container that holds a name for each imported Resource.
Inbox	A Core-managed Resource used to hold request messages from the Core to a Client.
Issuer	An entity issuing a certificate. The Issuer is denoted in a certificate by its Public Key
Logical Machine	A Core and its Repository.
Lookup request	Resources with attributes matching the lookup request will be bound to a name in the Client's name space.
Lookup Service	The component that performs lookup requests used to find Resources that match attribute-value pairs in the Resource Description of Resources registered in the Repository.

Term	Meaning
Mailbox	Either an Outbox or an Inbox.
Mapping Object	An object binding an ESName to Resources or a Search Recipe.
Message	Means of Client-Core communication.
Metadata	Data that is not part of the Resource's implementation, but is used to describe and protect the Resource.
Name Frame	A Core-managed Resource that associates a string with a Mapping Object.
Name Search Policy	A name conflict resolution tool used by the Core to find the appropriate strings when looking up names in a Name Frame.
Outbox	The location where the Client places a message to request access to a Resource.
Pass-by value	A metadata field, which, when set to true, includes the state of the Resource in the Export Form.
Principal	The entity holding the Private Key corresponding to a given Public Key
Private Key	This is secret data. An entity demonstrates knowledge of this secrete data by cryptographic techniques to authenticate itself. Private Keys must be kept secret

Term	Meaning
Private Security Environment (PSE)	A cryptographically secure store for Private Keys.
Protection Domain	The environment associated with a particular Outbox from which Resources can be accessed.
Publish	A request sent to the Distributor Service to publish Events.
Public Key	Non-secret data that is associated with a given Private Key by cryptographic techniques
Public Key Infrastructure (PKI)	A set of services and protocols that support the use of public and private key pairs by applications for security.
Repository	A passive entity in the Core that stores Resource metadata and the internal state of Core-managed Resources.
Repository entry	The metadata of a Resource as stored in the Repository and made available to the Core when a Client's requests to access Resources are processed.
Repository Handle	An index into the Repository associated with the metadata of a Resource.
Repository View	A Resource that can be used to limit the search for particular Resources in a large Resource Repository, much as a database view restricts a search within a database.

Term	Meaning
Resource Contract	A Resource denoting an agreement between the Client and the Resource Handler for use of a particular Resource. The agreement includes a provision for the Client to use an API known to the Resource Handler when making the request for the Resource.
Resource	The fundamental abstraction in e-speak. Consists of state and metadata.
Resource Description	The data specified for the Attribute field of the metadata as represented by the Client to the Core. See also Resource Specification.
Resource Factory	An entity that can build the internal state of a Resource requested by a Client.
Resource Handler	A Client responsible for responding to requests for access to one or more Resources.
Resource Specific Data	A metadata field of a Resource. Carries information about the Resource. Can be public or private to the Resource Handler.
Resource Specification	Consists of all metadata fields, except the Attributes field, as represented by the Client to the Core.
Session Layer Security Protocol (SLS)	The low level message protocol used by all e-speak Cores and Clients for remote communication.
Service Identity (ServiceID)	A field in the metadata that identifies a service or Resource

Term	Meaning
Simple Public Key Infrastructure (SPKI)	A specific variant of PKI developed within the Internet Engineering Task Force and used by e-speak.
State	Data a Resource needs to implement its abstraction.
Subject	The entity to which the access right or name has been issued. In a certificate the Subject is denoted by its Public Key.
Vocabulary	A Resource that contains the set of attributes and value types for describing Resources.
Tag	The field in a certificate expressing an access right
Vocabulary Builder	A Core-managed Resource registered by the Lookup Service that is used to create new value types, attributes, and Vocabularies.
Vocabulary Translator	A reference to a mechanism that is used to provide interoperation between different Vocabularies by mapping attributes from one Vocabulary into another through a Translator Resource.

AppendixA Resource Descriptions

E-speak makes a distinction between the data representing the state of a Resource and the data describing the management of the Resource. The Core mediates access to any registered Resource. However, e-speak is concerned only with the Resource state of Core-managed Resources, not with the Resource state of non-Core-managed Resources.

A Resource is described to e-speak by its metadata. The metadata is composed of a *Resource Specification* and a *Resource Description*. The Resource Description consists of information that provides the means of discovery for Clients. The Resource Specification includes:

- An Inbox that can be connected to the Resource Handler responsible for managing the Resource
- A specification of the security restrictions
- A variety of control fields

A Client registers a Resource by sending a message to a *Resource Factory* containing a Resource Description and a Resource Specification.

Together, Resource Descriptions and Resource Specifications include all information the Core needs to enforce the policies specified by the Client registering the Resource. If the registration succeeds, the Core returns a name bound to this Resource to the Inbox specified by the Callback Resource in the Outbox envelope.

Resource Specification

The `ResourceSpecification` class is defined below.

```
public class ResourceSpecification
```

```
{
   boolean byValue;
   ESName contract;
   FilterSpec filter;
   ADR metadataMask;
   ADR resourceMask;
   ADR ownerPublicKey;
   ADR ServiceId;
   ESMap privateRSD; //Not exported if export by reference
   ESMap publicRSD;
   ESName owner; //Not exported
   ESName resourceHandler; //Not exported
   int eventControl;
   ESUID uid;
   String URL;
}
```

The type `ESMap` is serialized as `ESArray`. The e-speak convention for `ESArray` is that it consists of a sequence of pairs. Thus, the first and second element are a pair, the third and fourth element are a pair, and so on.

The current implementation of `ResourceSpecification` uses the type `ResourceReference` where `ESName` is given. `ResourceReference` is the abstract base class for `ESName`. `ESName` and not `ResourceReference` is passed by the e-speak Application Binary Interface (ABI).

The owner and resourceHandler fields are not included when the ResourceSpecification is serialized for export, and the privateRSD field is only included in an export serialization if the export is export by value.

boolean byValue;

If the `byValue` flag is True, the internal state of this Resource isincluded with the Resource Specification and Resource Description sent to another Logical Machine. The Core provides the value for Core-managed Resources. Currently, no mechanism is defined for providing the value of non-Core-managed Resources.

ESName contract;

The `contract` field is the name of the Contract Resource associated with the Resource. A Contract embodies the contract between the user and the provider of a Resource. It denotes such things as the Application Programmer Interface (API) passed through the payload of a message. Every Resource must be registered in some Contract.

FilterSpec filter;

```
class FilterSpec{
   ESSet Vocabularies;
   String constraint;
}
```

The filter field consists of a set of vocabularies and a constraint. This is intended for use by a service provider to register a constraint that can be evaluated to determine if the Resource should be returned in response to an evaluation of a SearchRecipe in a lookup request in a NameFrame. The constraint uses the UserProfile associated with the client (xref to UserProfile). In this way a Resource can only be discovered by certain Clients that satisfy the constraint.

ADR metadataMask;

The metadataMask controls which operations manipulating the Resource's meta-data have security disabled. The interface name in the metadataMask are always the ResourceManipulationInterface. The format of the Resource Masks is specified in (Chapter 4, "The E-speak Security Model").

(The following text needs to be moved to the security chapter. It is here because Steve is currently editing the security chapter.)

Masks are specified as tags. The basic method tag format is

```
(net.espeak.method <interface name> <method name>)
```

In the metadataMask the interface name is the core interface being specified, and the method name is the operation in that interface. For metadata this isthe ResourceManipulationInterface, and the method name one of its methods. For the resourceMask the interface name isone of the interfaces supported by the Resource.

In the resource mask for an external Resource the interface name is the fully-qualified name of the interface class. For a Core-managed Resource, the interface name is the not qualified, so we just have "NameFrameInterface" and "ProtectionDomainInterface" etc. The method name is the name of the method in the interface, plus the concatenated argument types. This allows overloaded methods to be distinguished.

The metadata mask is used by the in-core meta resource when performing metadata operations. The resource mask is passed to the service handler by the core for the service handler to use when performing operations on the service itself.

The masks are completely general tags, so the mask tag itself, or any of its fields, can use the tag matching features such as sets, prefixes and ranges. The interface and method names, for example, do not have to be string literals, they can be sets or prefixes.

This tag masks method for interface net.espeak.examples. ExampleIntf:

```
(net.espeak.method net.espeak.examples.ExampleIntf foo)
```

This tag masks method foo in interface net.espeak.examples.ExampleIntf and method bar in interface net.espeak.examples.Example2Intf

```
(net.espeak.method (*set
      (net.espeak.examples.ExampleIntf foo)
      (net.espeak.examples.Example2Intf bar)
      )
)
```

This tag masks all methods beginning with foo:

```
(net.espeak.method net.espeak.examples.ExampleIntf (* prefix foo))
```

This tag masks methods foo and bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf (* set foo bar))
```

Methods with prefix foo or bar:

```
(net.espeak.method net.espeak.examples.ExampleIntf
 (* set (* prefix foo) (* prefix bar)))
```

All methods in the interface:

```
(net.espeak.method net.espeak.examples.ExampleIntf)
```

This is equivalent to

```
(net.espeak.method net.espeak.examples.ExampleIntf (*))
```

since missing trailing elements match anything.

All methods foo in Interface A and bar in Interface B:

```
(* set (net.espeak.method Interface A foo)
       (net.espeak.method Interface B bar))
```

All methods:

```
(net.espeak.method)
```

or simply

```
(*)
```

ADR resourceMask;

The resourceMask determines which operations supported by the Resource has security disabled.

ADR ownerPublicKey;

This field contains the owner public key.

ADR ServiceId;

This field contains the serviceId of the Resource.

ESMap publicRSD;

The first element in each pair of `ESMap` is a `string` used to tag the second element. The second element is of type `byte[]`. The `PublicRSD` field (public Resource-specific data) can be of interest to users of the Resource. Therefore, the Client registering the Resource can include information in this field. It is an error if either the tag or byte array is null or if the tags are not unique.

ESMap privateRSD;

The first element in each pair of `ESMap` is a `string` used to tag the second element. The second element is of type `byte[]`. The `privateRSD` field (private Resource-specific data) is used by the Resource Handler when a Client sends a message to this Resource. Therefore, the Client registering the Resource includes information in this field. This data is delivered to the Resource Handler. The intent is that only the Resource Handler have access to this data, but permission can be granted to any task using the e-speak security mechanisms. It is an error if either the tag or byte array is null or if the tags are not unique. This field is most often used to carry the Resource Handler's designation for the Resource.

ESName owner;

The `owner` field is the `ESName` of the active Protection Domain of the Client that registered the Resource. This field can be changed to another Protection Domain by any Client that unlocks the proper permission. It is an error if the `ESName` is not bound to a Protection Domain.

ESName Resource Handler;

Messages sent to this Resource isdelivered to this Inbox. This field is `NULL` for Core-managed Resources. The Client that has connected to this Inbox receives messages for this Resource. This field can be `NULL` only if the Resource being registered is Core-managed. It is an error if the `ESName` specified by the Client is not bound to an Inbox.

int eventControl;

If `eventControl` is non-zero, then whenever the Resource metadata (the Resource Description or the Resource Specification) is changed, an Event is published to the Core's Event distributor.

ESUID

```
public class ESUID
{
   byte[] UniqueId;
}
```

An ESUID contains a byte array that is up to 64 bytes long. An ESUID is guaranteed by probabilistic means. In the current implementation is consists of a

a Core identity component and Resource identity component as well as an indication if the associated Resource is local or remote (imported). The Core identity is unique (to a high probability) and is 20 bytes long. The Resource identity is unique within a given Core and is 12 bytes long.

String URL;

This field is the ESName (represented as a string) by which the registering entity refers to the Resource. It is an ESName (URL) which others can use to access the Resource.

Resource Description

`ResourceDescription` contains an array of Vocabularies and the attributes associated with each. Clients can specify a search request and ask the Lookup Service to find Resources with attributes that match the lookup request. An attribute specification includes a Vocabulary in which to interpret the attributes that describe the Resource.

`ResourceDescription` is an array of `AttributeSet` as shown below.

```
public class ResourceDescription
{
   AttributeSet[] attribSets;
}
```

Each element in the `ESArray` is an `AttributeSet`.

An `AttributeSet` consists of the `ESName` of a Vocabulary Resource and an `ESMap` of name-attribute pairs.

The Vocabulary is one in which the attributes have meaning

The first element of an `ESMap` pair is a string, the second element is an `Attribute`. The string is the name of the `Attribute`. It is an error if `ESName` is not bound to a Vocabulary or if `Attributes` or their values are not valid in the named Vocabulary.

```
public class AttributeSet
{
   ESName attrVocab;
   ESMap attributes;
}

public class Attribute
{
   String name;
   Value value;
   Boolean essential;
}
```

The `name` field is the name associated with `Attribute`. It can contain a single value or a set of values (sets of values are not supported in the current release).

If `essential` is true, then this attribute must be included in any search request to discover a Resource with this attribute in its Resource Description.

Resource type

The e-speak Core associates a type with every Resource registered. The following defines the currently recognized Resource types.

```
class resourceType{
   static int INBOX_CODE = 0;
   static int META_RESOURCE_CODE = 1;
   static int PROTECTION_DOMAIN_CODE = 2;
   static int RESOURCE_FACTORY_CODE = 3;
   static int CONTRACT_CODE = 100;
   static int CORE_DISTRIBUTOR_CODE = 110;
   static int IMPORTER_EXPORTER_CODE = 120;
   static int MAPPING_OBJECT_CODE = 140;
   static int NAME_FRAME_CODE = 150;
   static int REPOSITORY_VIEW_CODE = 160;
   static int SECURE_BOOT_CODE = 170;
   static int SYSTEM_MONITOR_CODE = 180;
   static int VOCABULARY_CODE = 190;
   static int CORE_MANAGEMENT_SERVICE_CODE = 200;
   static int DEFAULT_VOCABULARY_CODE = 210;
   static int DEFAULT_CONTRACT_CODE = 220;
   static int FINDER_SERVICE_CODE = 230;
   static int CONNECTION_MANAGER_CODE = 240;
   static int REMOTE_RESOURCE_MANAGER_CODE = 250;
   static int EXTERNAL_CODE = 1000;
   static int EXTERNAL_RESOURCE_CONTRACT_CODE = 1001;
   static int INBOX_CODE = 0;
   static int META_RESOURCE_CODE = 1;
   static int PROTECTION_DOMAIN_CODE = 2;
   static int RESOURCE_FACTORY_CODE = 3;
   static int CONTRACT_CODE = 100;
   static int CORE_DISTRIBUTOR_CODE = 110;
   static int IMPORTER_EXPORTER_CODE = 120;
   static int MAPPING_OBJECT_CODE = 140;
   static int NAME_FRAME_CODE = 150;
   static int REPOSITORY_VIEW_CODE = 160;
```

```
static int SECURE_BOOT_CODE = 170;
static int SYSTEM_MONITOR_CODE = 180;
static int VOCABULARY_CODE = 190;
static int CORE_MANAGEMENT_SERVICE_CODE = 200;
static int DEFAULT_VOCABULARY_CODE = 210;
static int DEFAULT_CONTRACT_CODE = 220;
static int FINDER_SERVICE_CODE = 230;
static int CONNECTION_MANAGER_CODE = 240;
static int REMOTE_RESOURCE_MANAGER_CODE = 250;
static int EXTERNAL_CODE = 1000;
static int EXTERNAL_RESOURCE_CONTRACT_CODE = 1001;
```

Appendix B XML Book Broker Example

XML Book Broker Example

An example demonstrates how e-speak can be used for service composition based on existing services in the Internet. Assuming that not all services offered in the Internet are e-speak enabled, it is in particularly important to support bridging between different "service worlds". E-speak allows it by so-called proxy services. Proxy services are regular e-speak services connected to an e-speak engine. Proxy services provide the mapping between the e-speak service world and other service worlds.

Today's services in the Internet are mostly designed for direct customer interaction (b2c). Content and its presentation is mixed in HTML. The term content refers here to the "bare information" about a subject such as the price and delivery information of a book. Content is stored in databases behind the web servers. Presentation means wrapping content into final HTML, including all information needed for the dialog with the user (such as all the links the user can chose for the next dialog step, session handling etc.). Since presentation is a major differentiator among web sites with direct customer interaction, it is becoming more and more sophisticated and overloads the actual content information making it difficult to build interacting service applications (b2b) and service composition.

Search engines are early examples of application services obtaining and classifying static content from web sites and offering searches on the cached content. However, a big portion of content is dynamic today and not retrievable through search engines.

Two major challenges need to be addressed in the web today in order to enable dynamic service composition:

1. Content must be directly accessible through XML separate from its presentation in browsing devices,
2. Service composition cannot be based on cached content, content must be obtained directly from the original service provider in "real time" on request.

XML allows keeping content and its presentation separate. Merging with the presentation information can be processed at a late stage potentially in browsers. XML and related technologies provide means to overcome handicaps service composition is faced today. XML allows to:

1. Separate content from its presentation.
2. Use a common data format to represent both content (in XML) as well as its presentation (XML/XSL).
3. Describe service provider specific, individual XML formats in XML itself by XML schema. This opens a potential for generic translators rather than pairwise translators between XML formats.
4. Define commonly accepted semantics for schemas for various business areas by common vocabularies as currently proposed by several working groups [1], [2].

E-speak's vision is to provide a homogeneous platform for this upcoming world of service composition in the Internet using the capabilities XML technologies provide. However, there is still a way to go that providers in the Internet can open direct access to their contents through XML.

The purpose of the Book Broker example is to demonstrate e-speak's vision of how service composition becomes enabled in the near future as well as to demonstrate what the possibilities and limitations are today. The Book Broker demonstration software has been part of e-speak since release Beta 2.0 as complete source code.

Overview of the E-speak Book Broker Service

Three existing online bookstores are referred to as real, external service provides in the Internet. All of them are not e-speak enabled. Proxy services must be represented by respective proxy services in e-speak. They are registered with the e-speak

engine and discoverable based on attributes using an online bookstore vocabulary. Proxy services run outside the engine. They are "dynamically plugable" by dynamic registration with the e-speak engine and can be provided by a third party or by the online bookstores themselves.

The overall architecture is shown in the following figure:

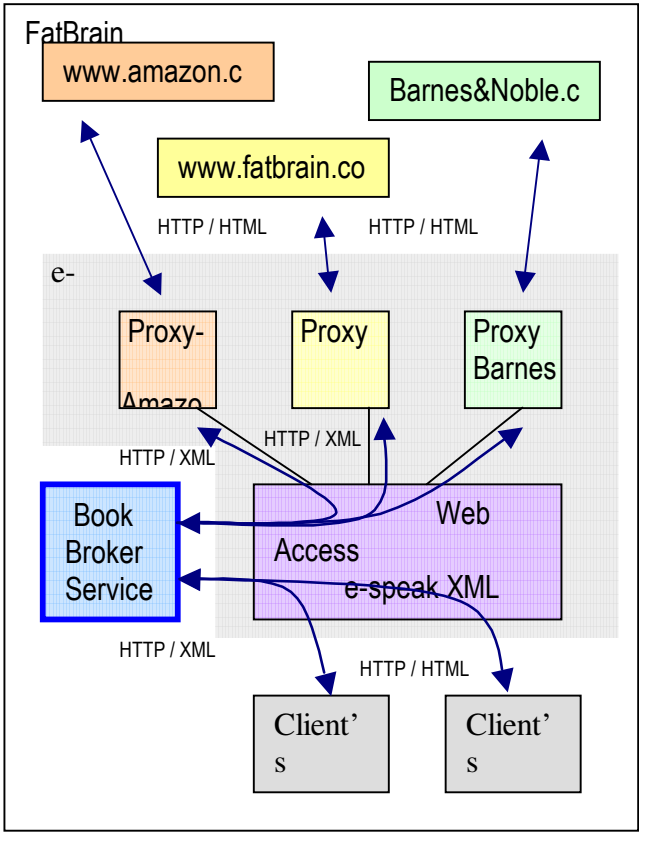

Figure 2 An Overview of the Book Broker Service

The main task of proxy services is to provide the bridge between non-e-speak services accessible through HTTP and HTML and e-speak services connected through XML. Eventually, the need for proxy services becomes obsolete when online bookstores directly provide a XML content interface. Online bookstores can then directly connect and register with e-speak omitting the need for proxy services.

From an e-speak perspective, the Book Broker service itself is an external service. It is connected to the e-speak engine and discoverable by clients (which can be human users or other services) using a broker vocabulary. Clients are not aware of how many bookstores have currently registered with the e-speak engine. They also cannot directly interact with them. A client discovers the Book Broker service and interacts with this service only. If multiple Book Broker services have registered with the engine, the user (or the application) can chose one or work with multiple of them.

The Book Broker service offers services such as "find all offers for a certain book". Which offers finally are found depends on how many online bookstores have registered with the e-speak engine at a particular time and are currently accessible. Online bookstores can connect and disconnect at any time. The number and kind of bookstores varies over time. E-speak's dynamic service (un-)registration and discovery directly reflects the dynamic nature of the Internet. The only assumption is that services are described and registered using commonly shared vocabularies. Vocabularies are discoverable themselves in the e-speak engine. They are described by attributes defined in one of the "vocabulary vocabularies". The recursion of vocabulary definitions is grounded by the base vocabulary.

When a client sends a book query to the Book Broker service, the Book Broker queries the e-speak engine for currently registered online bookstores and forwards the query to all of them in parallel, currently going through proxy services as shown in the figure. The online bookstores process the query and return results to the Book Broker service. The Book Broker service collects all results and returns a list of book offers to the client. Web Access decide whether the internally used XML should be returned to application clients directly or a XML to HTML (WML) translation is performed for browsing devices.

The functionality of the Book Broker service is currently not very sophisticated. There is no negotiation and no participation in auctions. Enhancements are planned for the future using protocols currently under development. However, even in its current simplistic form, the Book Broker service provides some features which differentiate it from existing "composition web-sites" or search engines:

1. Results are obtained based on real-time queries directly at the providing web-sites, not based on pre-queried or cached information which can easily be out-of-date.
2. There is no knowledge required in clients as well as in broker services about which providers have currently registered and where and how they can be contacted. This information is dynamically obtained from the engine's repository per invocation request (per use). New service providers are automatically found as soon as they have registered with the e-speak engine. Services becoming unavailable are no longer discoverable and referred to as soon as their description data disappears from the engine's repository.
3. Programs and business logic in clients and in broker services becomes freed from connection states and configuration information about where and how to interact with other service providers. The dynamic discovery of e-speak enables on-time, up-to-date control flows between services unaware of each other.
4. E-speak's Web Access performs all necessary translations between XML and HTML or WML according to the needs of the client application or browser. XML to XML transformations based on XSL are planned for future releases.

Book Broker DTD and XML Examples

The Book Broker uses XML to represent queries as well as the resulting book lists. The format of this XML is defined in a simple Book Broker DTD:

```
<!DOCTYPE BOOKBROKERDTD [
<!-- E-speak Book Broker DTD -->
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT subject  (#PCDATA)>
```

```
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT offeredby (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT shipsin (#PCDATA)>
<!ELEMENT URLtoOrder (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT bookdesc (author?,title?,subject?,publisher?,isbn?)>
<!ELEMENT bookoffer (author, title, publisher, year, offeredby,
                          price, shipsin?, URLtoOrder, comments?)>
<!ELEMENT booklist (bookoffer*|bookdesc+)>
<!ELEMENT queryresult (booklist)>
<!ELEMENT bookquery (booklist)>
]>
```

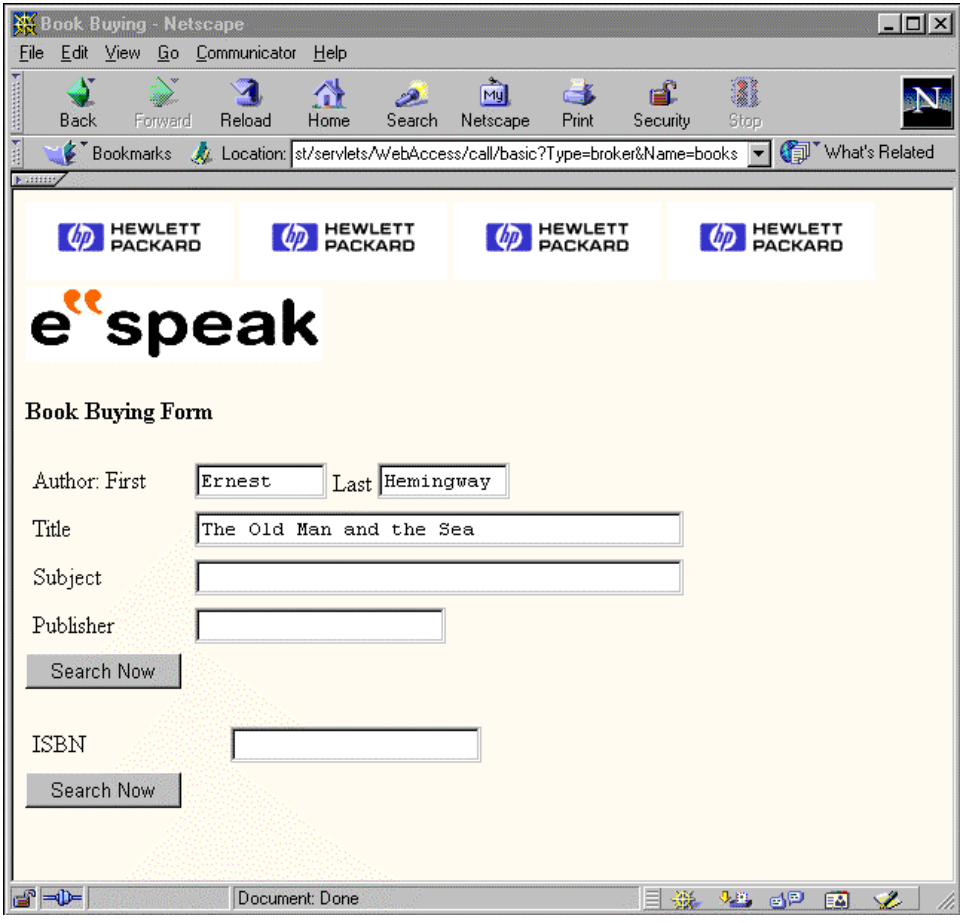

Figure 3 Query Screen presented in the Browser

The query for the shown book can be generated in a client application and sent to Web Access in the body of a HTTP request. Or it can be generated from a browser's FORM POST request from a query form as shown in Figure 2.

Web Access performs the necessary transformations into XML. An example of a query in XML is shown below. Multiple queries can be submitted with one request:

```
<bookquery>
  <booklist>
    <bookdesc>
      <author> Hemingway </author>
      <title> The Old Man and the Sea </title>
   </bookdesc>
  </booklist>
</bookquery>
```

A XML representation of a returned result might be:

```
<queryresult>
  <booklist>
    <bookoffer>
      <author> Hemingway </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 </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>
</queryresult>
```

The Book Broker service returns the resulting booklist to client applications in XML without any modifications as response to the initial HTTP query request. For users behind web browsers, the booklist is automatically translated into HTML or WML by Web Access.

The resulting screen is shown in the next figure. In order to purchase a particular book, the user can click on one of the URLs shown in Figure 3 and are directly referred to the offer at the bookstore's original web site (<URLtoOrder> element in the XML result document)

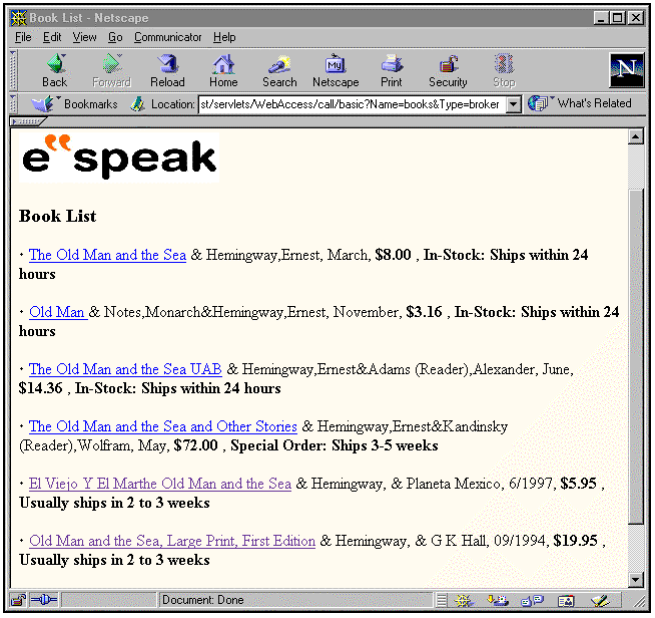

Figure 4 Final List of Book Offers returned to the Browser

Dependencies and Limitations

The current version of the Book Broker example has two main limitations:

- The Proxies are only capable to perform single-step invocations on the book-store's web sites. This means that all invocation information must be delivered (and accepted) in one HTTP request. It is currently not possible for proxies to emulate a user stepping through multiple pages to the desired content. Some web sites require these steps to be carried out in order to perform a login, to set cookies or to collect user's information. The final HTTP request then requires dynamic information as gathered while going through previous pages on the web site. The three considered online bookstores do not require it.
- Another limitation is the dependency in the proxy code on the formats the respective web-site accepts for invocations as well as for interpreting the result obtained from the response.

As mentioned, the proxy services perform an encoding of the information obtained from a XML book query into site-specific formats (URL parameters) of the HTTP request issued by the proxy to the web-site. This information is usually contained in the form page shown in a user's browser to make a selection. Since this form page is loaded from the same web-site, the web-site can control and change the format of the final HTTP request at any time. The proxy's programs currently hard coded one particular format which needs to be adjusted when the web-site changes this format.

The same applies for the results returned from the web-sites in HTML. The proxies parse this HTML according to hard-coded rules in their programs. Programs need to be changed when the web-site changes the presentation.

Both limitations are related to the problem of mixed content and presentation used today in human user oriented web-sites. Both limitations as well as the need for proxy services are overcome after web-sites open access to their content directly. E-speak takes advantage of these upcoming changes in the Internet.