Seamless Integration of Distributed Electronic Patient Records

WP2 Deliverable D2.1

Abstract

This document comprises one of the two main deliverables from *SynEx Work Package 2 - Norway SynEx Validation*. The objective of WP2 is to design and implementation a platform for software components that can be used to make clinical information available to health professionals using the paradigm of *shared, distributed electronic patient records*, based on the principles established by CEN/ENV 12265 and Synapses, and using the object-oriented paradigm and industry standard technology. As a proof of concept, WP2 includes a demonstration of how the platform and its support for shared, federated healthcare records can be used to support the continuity of cardiovascular care for patients that are examined at one of the verification hospitals (SiA) and undergoes cardiac surgery at the other hospital (RH).

This deliverable D2.1 presents the client-side components that enables a seamless integration of distributed electronic patient records. The other deliverable *"D2.2 A Platform for Electronic Patient Record Integration"* presents a set of server-side components that comprise a platform from which healthcare record information can be accessed on an extranet with common internet technology.

The work of WP2 is conducted by *Siemens Health Services (SHS)*, *Sentralsykehuset i Akershus (SiA)* and *Rikshospitalet (RH)*.

Siemens Health Services, P.O.Box 10, Veitvet,
N-0518 Oslo, Norway
Phone +47 22 63 30 00, Fax +47 22 63 48 80

Author	:	**Egil.Paulin.Andersen@nr.no** (Norwegian Computing Center, http://www.nr.no)		
Distribution	:	**All Consortium members**		
Date	:	**24th July 2000**	Version :	**1.0**
Status	:	**Final**		
Filing code	:	**SHS-025[WP2]**	Classification :	**Public**

Contents

Author	:	**Egil.Paulin.Andersen@nr.no** (Norwegian Computing Center, http://www.nr.no)		
Distribution	:	**All Consortium members**		
Date	:	**24th July 2000**	Version :	**1.0**
Status	:	**Final**		
Filing code	:	**SHS-025[WP2]**	Classification :	**Public**

1. Motivation and Background

The management of electronic patient data was relatively easy as long as the data was collected, stored and viewed in a closed environment like a hospital or doctor's practice. Heterogeneous and compatible IT systems were provided by one company and externally or internally employed, centralised system administrators were responsible for the smooth use of all installed components. No connection to the "outside world" was established which eased the protection of patient data immensely. Data exchange was only possible by paper, mail, fax or telephone, which lead to extra work and unreliability: feedback and results had to be manually entered into a new system. Increased computerisation throughout the health sector has given rise to a proliferation of independent systems storing patient data. However, the growing trend towards shared care requires that these systems are able to share their data. This has led to the development of projects such as *Synapses* [3][4][5] and its follow-up *SynEx* [1][2] which aims to provide healthcare professionals with integrated access to patient records and related information, regardless of where this information resides.

The goal of SynEx Work Package 2 is to support the continuity of care through *shared federated healthcare records (FHCR)*. That is, a migration from FHCRs in the Synapses perspective, where they are primarily used to integrate legacy systems, to FHCRs in the SynEx perspective where records are shared across extranet, and where parts of a record can be regarded as part of another record. More specifically, the tasks of WP2 was to design, implement and demonstrate

- *a platform for software components that make clinical information available to health professionals using the paradigm of distributed electronic patient records, based on the principles established by CEN/ENV 12265 and Synapses, and using the object-oriented paradigm and industry standard technology.*
- *a federated health care record (FHCR) supporting the continuity of cardiovascular care for patients that are examined at one of the verification hospitals (SiA) and undergoes cardiac surgery at the other hospital (RH).*

The work in WP2 has resulted in a distributed, component-based information system where users can access information in a Synapses compliant server on an extranet based on common internet technology.

2. Scenario for Demonstrating Shared, Federated Healthcare Records

The platform and software components developed in WP2 will be used to demonstrate, in a semi real life situation, clinical collaboration based on a federated healthcare record (FHCR) shared between two closely collaborating hospitals providing shared cardiovascular care. The FHCR can be used as an alternative for sharing information by being composed from two different records in two hospitals.

The Norway clinical sites for the installation and evaluation of SynEx products are:

- *The Department of Medicine at the Central hospital of Akershus (**SiA**–Sentralsykehuset i Akershus)*
- *The Department of Cardiac Surgery at the National Hospital (**RH** - Rikshospitalet)*

Both hospitals are reference sites for SHS prospects in the Norwegian market, and these two hospitals collaborate in the treatment of patients with angina pectoris that needs surgical treatment. Patients with Angina Pectoris in the County of Akershus are examined and considered for bypass operation at SiA. If candidate for operation, the patient is transferred to RH for the actual operation.

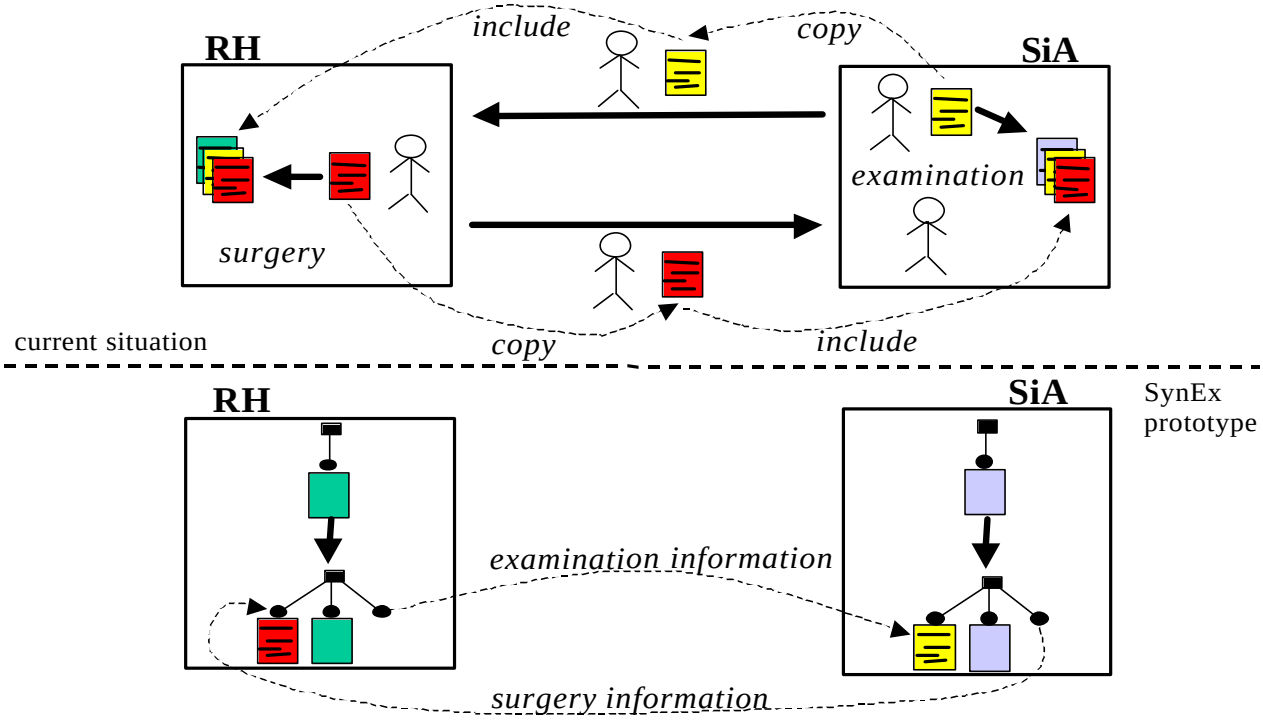


Figure 1. *Demonstration scenario.*

Current situation

Today, when a patient is admitted to SiA, a record, or a new section in an existing record, is created. Examinations are made and results recorded in the record. If surgery is required the relevant record information is copied, and a discharge letter written. This accompanies the patient to RH. At RH a new record, or a new section in an existing record, is created. The accompanying information is re-typed, and the required treatment recorded. When treatment is finalised the patient and the relevant parts of his record, together with a discharge letter, is transferred back to SiA.

Utilising Shared Federated Healthcare Records

Figure 1 illustrates a scenario for how shared FHCR's can be used to simplify this procedure. Relevant parts of the record at SiA will be made available to the appropriate doctors at RH, while other parts are hidden. These parts will be considered part of the patients record at RH, and thus part of the federated record.

Correspondingly, the part of the RH record relevant for the continuity of care will be made available to the appropriate doctors at SiA, and become part of their record.

Validation

Validation of the demonstrator will be based on combining several real patient cases and real patient data, in an anonymous way. It will thus behave as if it were a real life case, but it will not be used on actual cases. The current legislation in Norway prevents us from transferring real patient data across extranet.

Benefits

The benefits of shared FHCR as demonstrated by the prototype are:

- it facilitates the continuity of care between organisations
- relevant information is available and thus reduces the risk of making wrong decisions due to lack of information
- it reduces administrative work
- the basis for decision making is more explicit and available
- it facilitates quality monitoring/control, research, etc.

3. Using the Demonstrator

3.1 Introduction

This document presents a set of client-side components that are implemented for the demonstration of shared, federated healthcare records according to the above scenario. We will refer to this SynEx WP2 demonstrator as the *SHS Demonstrator*, and we can distinguish between two different groups of components made.

A set of COM components (within dll's) are made to achieve a seamless integration of information of a number of distributed electronic patient records. These COM components have no graphical user interface (GUI). Instead they offer a set of COM interfaces that can be used (programmatically) by other, e.g. GUI, components to retrieve FHCR information. Thus these components are not made for a specific application. Rather they can be used by any application that needs to work with SynEx compliant[1] FHCR's (as a front-end to the webserver).

In addition a set of ActiveX controls (.ocx's) are made specifically for this SHS Demonstrator. They provide the GUI of the demonstrator, but since the main topic of this project does not concern user interfaces and graphical presentations, this GUI is not of production quality. However, the demonstrator supports three different means for presenting information graphically (XSL, DHTML, ActiveX/Applets; see below), and any authorised user can improve presentation quality without having to change or recompile any components.

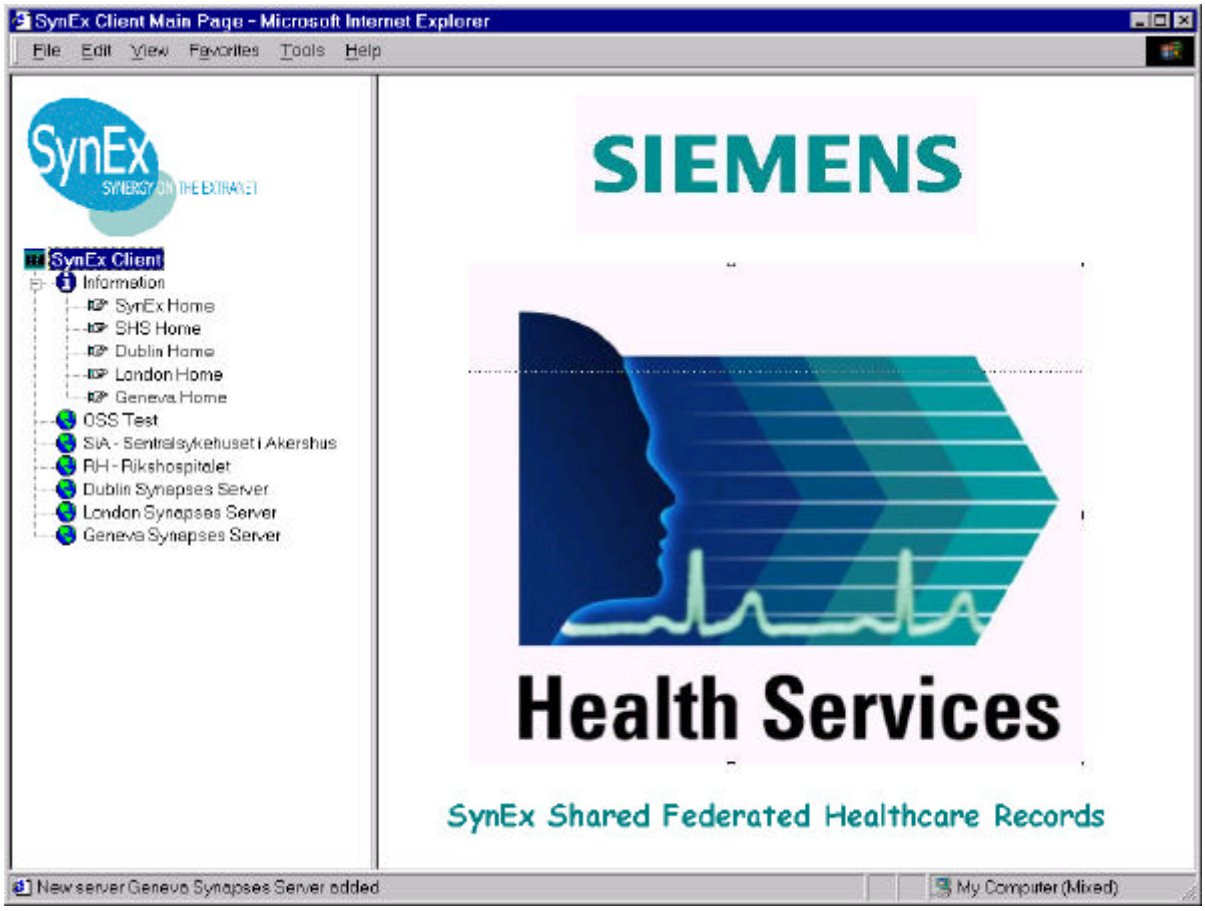


Figure 2. *Start page of the SHS Demonstrator.*

3.2 User View of the SHS Demonstrator

[1] i.e., FHCR's offered by Synapses servers in SynExML formatted XML.

The following is a brief description of how a session with the SHS Demonstrator will be perceived by a user.

A user first navigates in Internet Explorer v.5+ to the home page of the application. The user is then presented with the view in figure 2.The GUI consists of two resizable HTML frames. The left frame contains a tree-view control, and this is where the user interaction takes place. A right mouse click on the various tree-view items will present a list of possible actions. The right frame is for information presentation.

Initially, when starting the application, the tree-view control contains a set of items for sources of information on SynEx, and a set of items representing available servers to connect to. An information source like e.g. "SynEx Home" must be activated by a double-click (as opposed to the right click for every other item), and figure 3 illustrates the result of this.

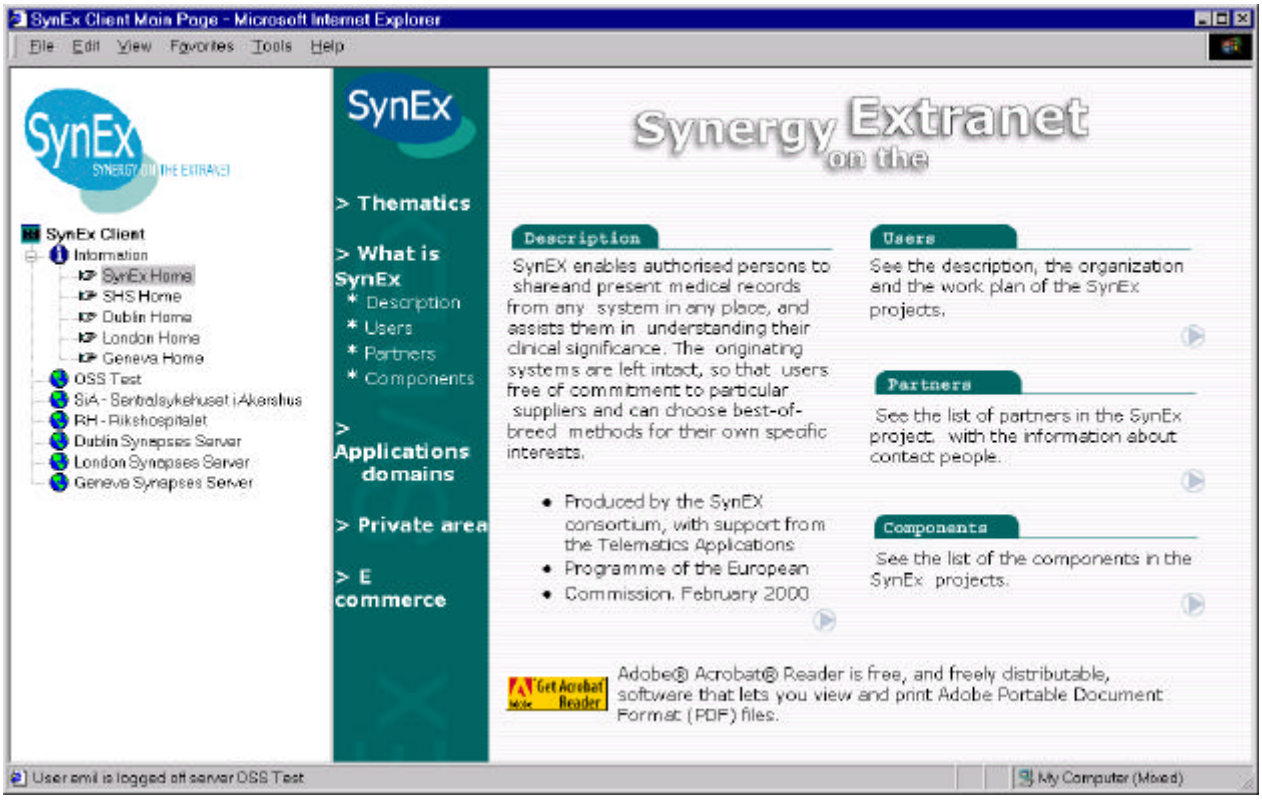


Figure 3. *The SynEx home page.*

Information on a particular server, e.g. "OSS Test", will be provided by selecting "Properties" after a right click on this tree-view item. By selecting "LogOn" the user will be asked for a user name and a password for this server, as illustrated in figure 4.

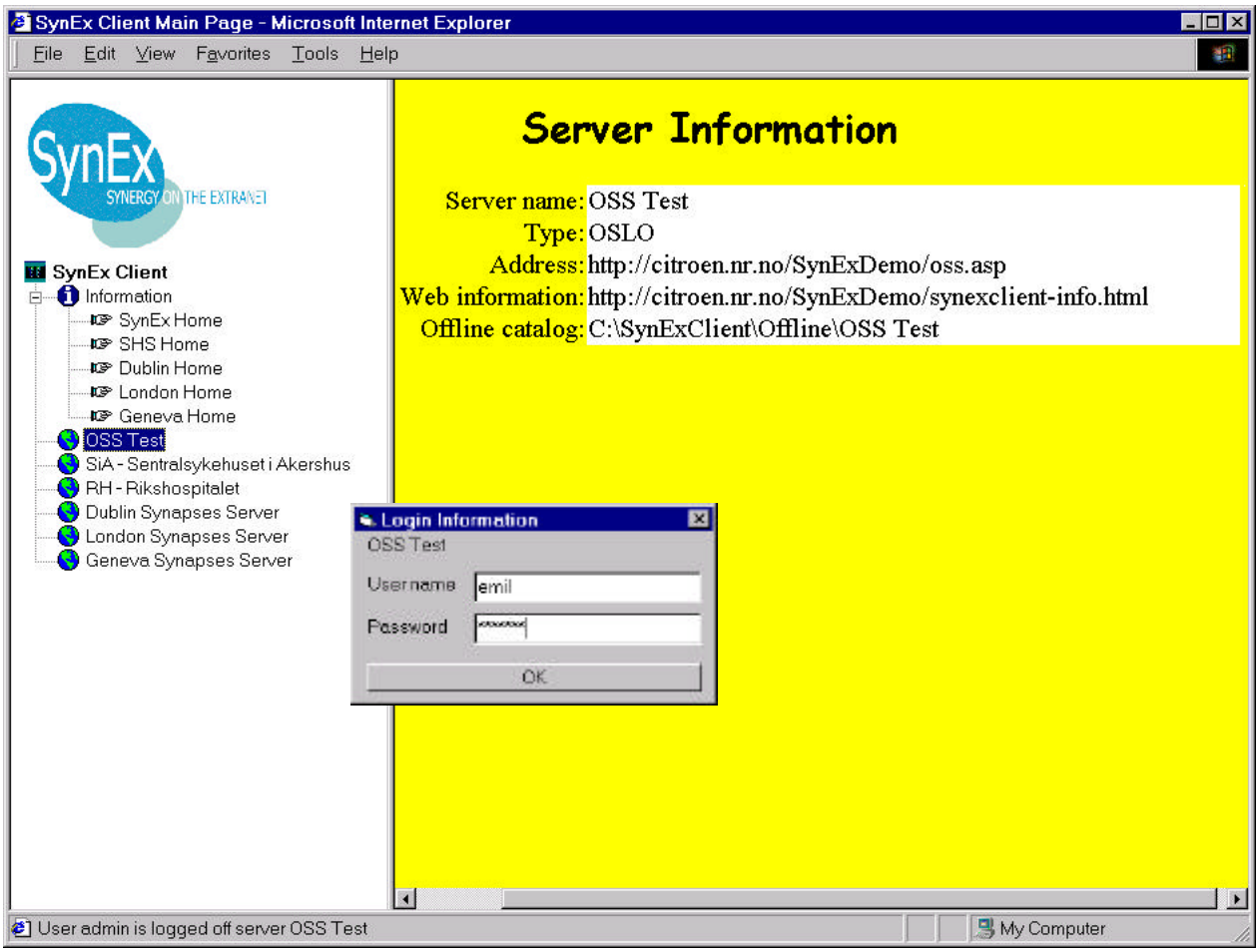


Figure 4. *Log on to server "OSS Test".*

After a successful logon, the user can select "Record Search" in the login menu (right click the Login tree-view item) to retrieve information on a particular record, folder or document.

Notice that the current means for requesting a particular record, folder or document is not very user friendly, to say the least, since the user must know the exact identifier of the information requested. Thus an important part of further work on the SynEx FHCR specification will be to agree on how to standardise record search, and search for parts of records.

After a successful record request the client will receive information on the folders and documents of this record, i.e., information on the record structure, and the user will have them presented in the tree-view control with documents as leaf nodes. Afterwards the user can retrieve more information on individual documents and/or folders by right clicking the corresponding tree-view items. For example, by right clicking on a particular document, e.g. "UsrNor_Registration", and selecting "View Document", this document will be presented in the right frame according to either a default, or alternatively a customised, specification of how it should be presented (e.g. XSL, DHTML; see section 3.5 for more information). Figure 6 illustrates this.

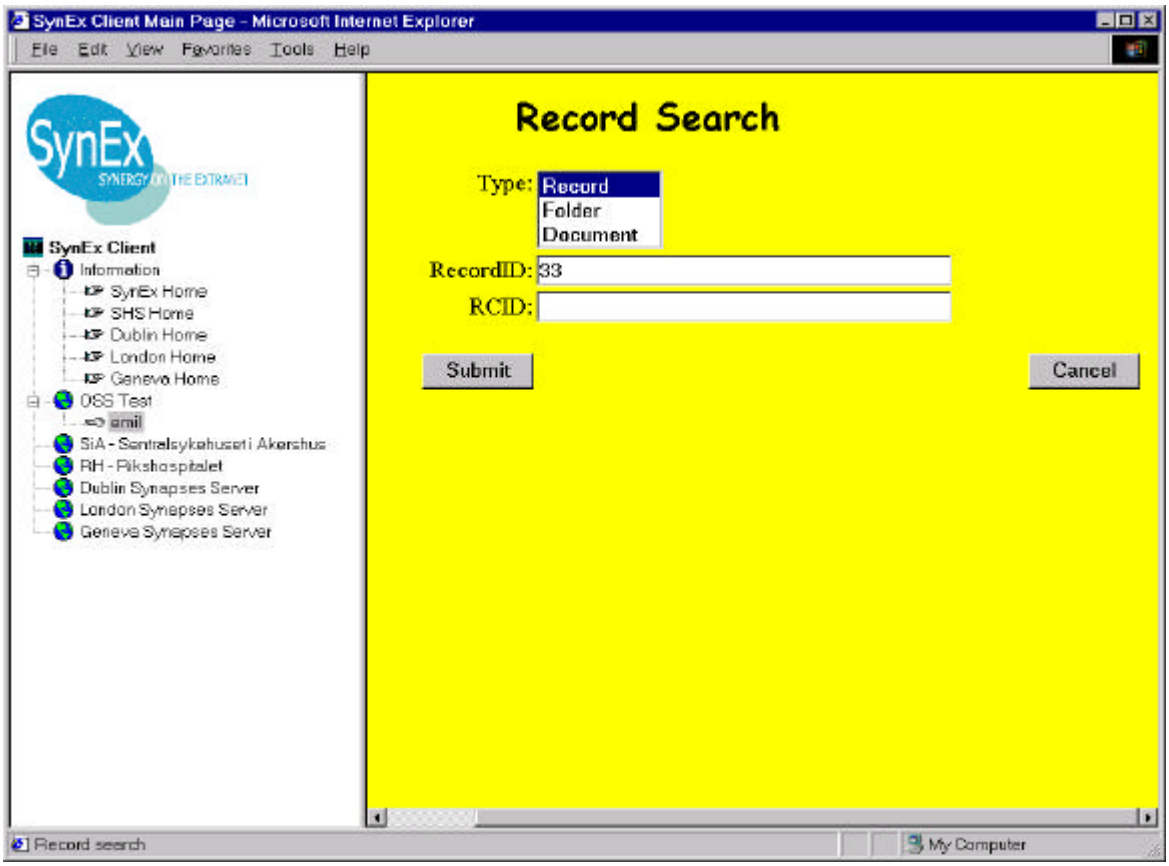


Figure 5. *Request information on a particular record, folder or document.*

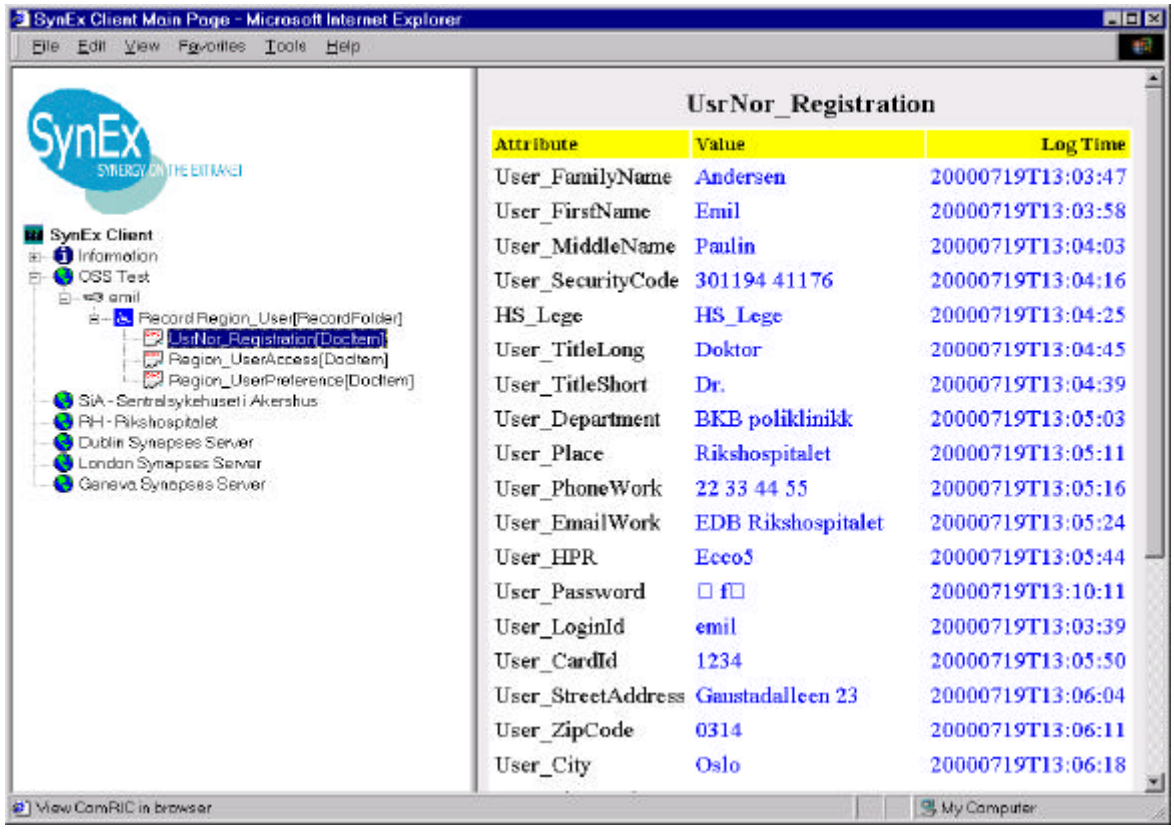


Figure 6. *Viewing a particular document.*

The following section provides a brief description of the various kinds of objects that constitute Synapses healthcare records.

3.3 Federated Healthcare Records according to Synapses

The FHCR model developed in *Synapses* has been used as the basis for the core XML DTD in SynEx (*SynExML (SynEx Markup Language)*). This section describes briefly the concepts of the Synapses Record Architecture in a simplified way, the *SynOM (Synapses Object Model)* and the *SynOD (Synapses Object Dictionary)*. In contrast to the very detailed and highly specific record architectures of for example *HL7 (Health Level 7)*, the SynOM is a generic and flexible common object model. It extends the model of the EPR (Electronic Patient Record) described in *ENV12265* from *CEN/TC251*. More detailed and technical descriptions can be found in [3] [4] [5].

The Synapses Record Architecture consists of a single class hierarchy, and every Synapses patient record consists of a set of objects where each object is instantiated from a class in this hierarchy. Each class in the hierarchy belongs to one out of two main groups of classes. The structure of a record is made out of objects instantiated from the "structural classes", called *Record Item Complexes (RICs)*, while the data (information) within a record consists of objects instantiated from "data value classes", called *Record Items*.

The structure of a Synapses record corresponds to a tree structure of RIC objects, and each record can have unidirectional links to other such tree structures, i.e., to other records. The tree structure of each record is rooted in a single particular *RecordFolder* object, i.e., instantiated from class *RecordFolder*, which represents the overall record. Below this object there will be a structure of folders (*FolderRIC* objects) and documents (*ComRIC* objects). Each document will itself consist of a tree structure of objects, which can be *DataRIC* objects and/or *ViewRIC1* objects. The former contains information that is explicitly recorded in the record, while the latter are used to represent computed or derived information.

In addition there are objects that represent links to other records, called *ViewRIC2* objects. These are the key to the SynEx integration of Synapses records. They contain the unique identification of another RIC object, and they are used as follows. The root object of a record, or a folder within a record, may contain a single ViewRIC2 object that references another record or folder object, respectively. In addition, a document may contain one or more ViewRIC2 objects that each reference some subset of other documents. The RIC object referenced by a ViewRIC2 object is either local, intra- or inter-record, or remote, and if remote then the ViewRIC2 object contains an URL that identifies the server where the target RIC object resides.

Each RIC object instantiated from one of the "structural classes" will have a small set of static, predefined attributes, e.g. as required for their unique identification, or the target address of a ViewRIC2 object. However, most of the information content of a record, and all the medical information, exists in *RecordItem* objects instantiated from the "data value classes". That is, a set of RecordItem objects can be attached to a structural RIC object and thus function as its *dynamic attributes* with actual data values like e.g. a blood pressure measurement. The RecordItem objects that belong to a particular RIC object can also themselves be organised into a tree structure. Due to these RecordItem objects, the information content of a record can by dynamically extended over time, and with information of a kind that may not have been foreseen at the time the record itself was created.

The distinction between RIC classes versus RecordItem classes comprises a kind of "vertical" grouping of the overall Synapses class hierarchy. In addition the class hierarchy is split "horizontally" into a predefined set of base classes common to every Synapses server, called the *Synapses Object Model (SynOM)*, and an *extendable* set of classes that are derived from these SynOM classes, called the *Synapses Object Dictionary (SynOD)*. The above RIC classes RecordFolder, FolderRIC, ComRIC, etc, all belong to the SynOM, and they define the core part of Synapses' generic record model. The SynOD classes on the other hand, which are site specific and thus may differ for each Synapses server, are the classes from which the actual patient record objects are instantiated. Thus while every record object has the above SynOM characteristics and properties, they can also be customised to the needs of each individual site.

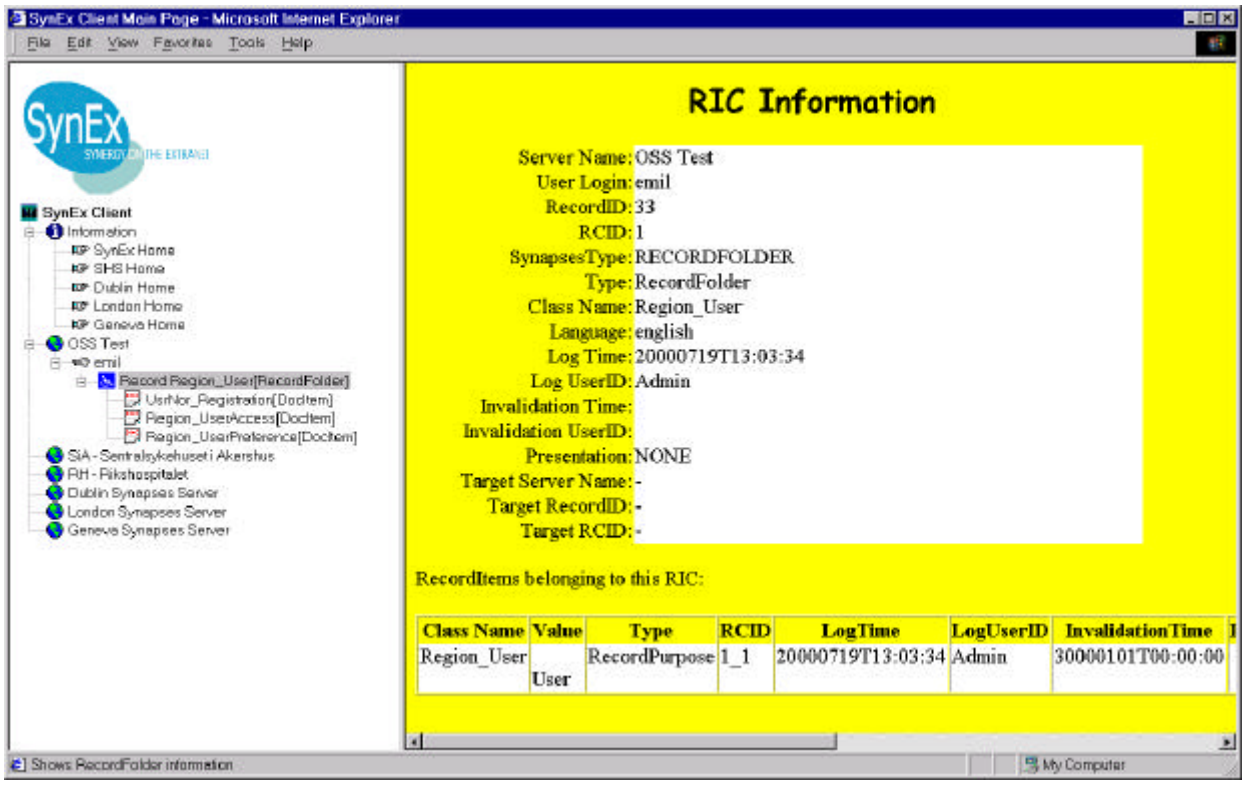


Figure 7. *Viewing detailed information on a RecordFolder.*

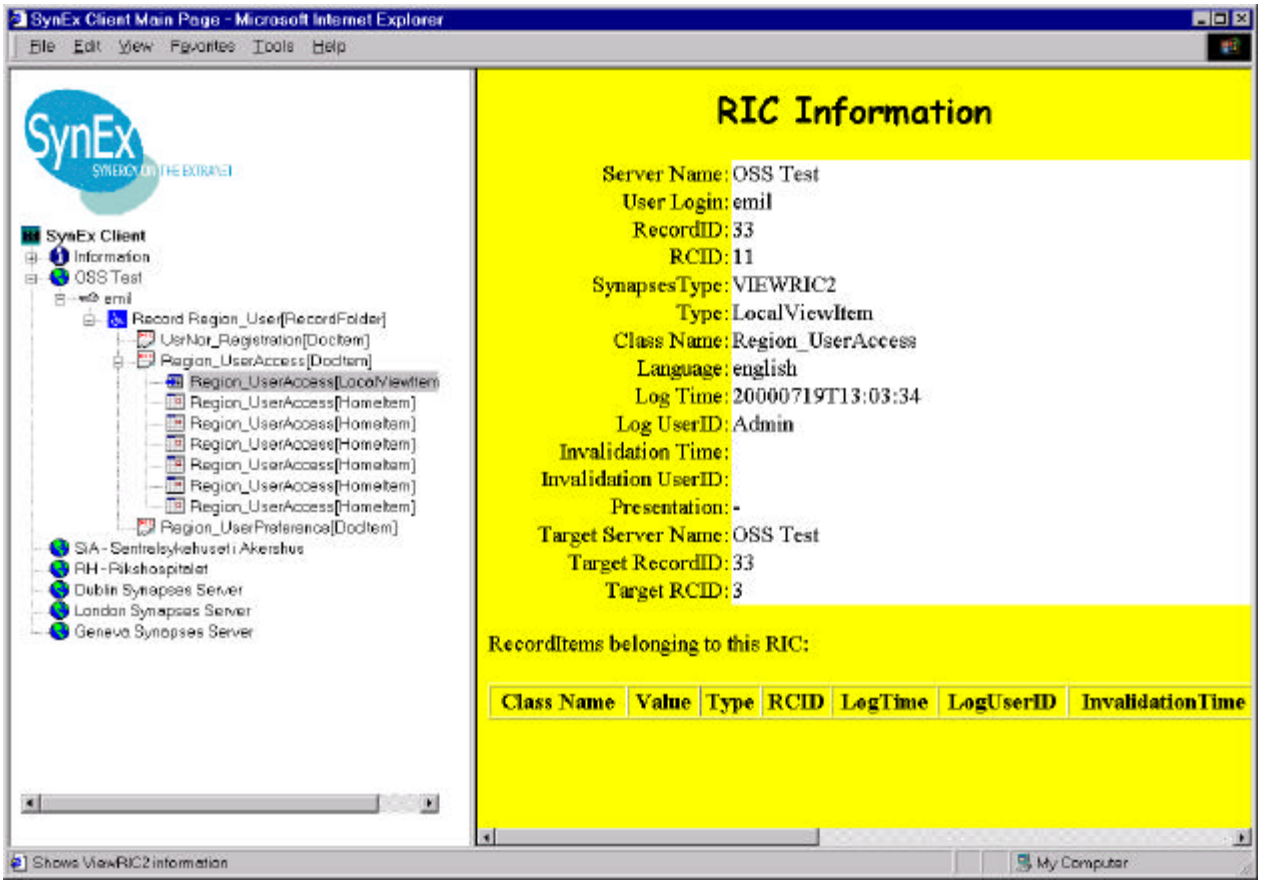


Figure 8. *Viewing detailed information on a ViewRIC2 (in this case a local link within the same document).*

As illustrated in figure 7 and figure 8, by right clicking on a "RecordFolder" item in the tree-view control, e.g. "Region_User", or an item that represents an object within a record, e.g. a local ViewRIC2 within the document "Region_UserAccess", and selecting "Properties" from the menu, detailed information on this RIC will be provided; e.g. which RecordItem's, if any, are connected to this RIC, and so on.

3.4 An Example Scenario

Consider a user that connects to a SynEx compliant site such as RH (Rikshospitalet) in Oslo. The user's access rights are collected and stored from an initial login operation. After the successful login, the user chooses what he wants to browse from a set of available records and record fragments. Parts of a particular selected record may reside at other sites such as SJH (St. James's Hospital) in Dublin. When presenting the user with information from this record, the fact that the information content is distributed should be transparent to the user. If a document from RH is requested it will be retrieved from the current server, while if a document from SJH is requested then the client will forward the request to SJH transparently to the user and retrieve the requested document from there. The information to forward the request is entered at the first request of data from SJH and will be stored at RH whereas the data itself resides at SJH. Of course, using this technique might lead to multiple levels of redirection, but it ensures a consistent storage of data at exactly one place.

Figure 9 illustrates an example healthcare record sharing between SiA and RH, while figure 10 extends this to also include a Dublin hospital.

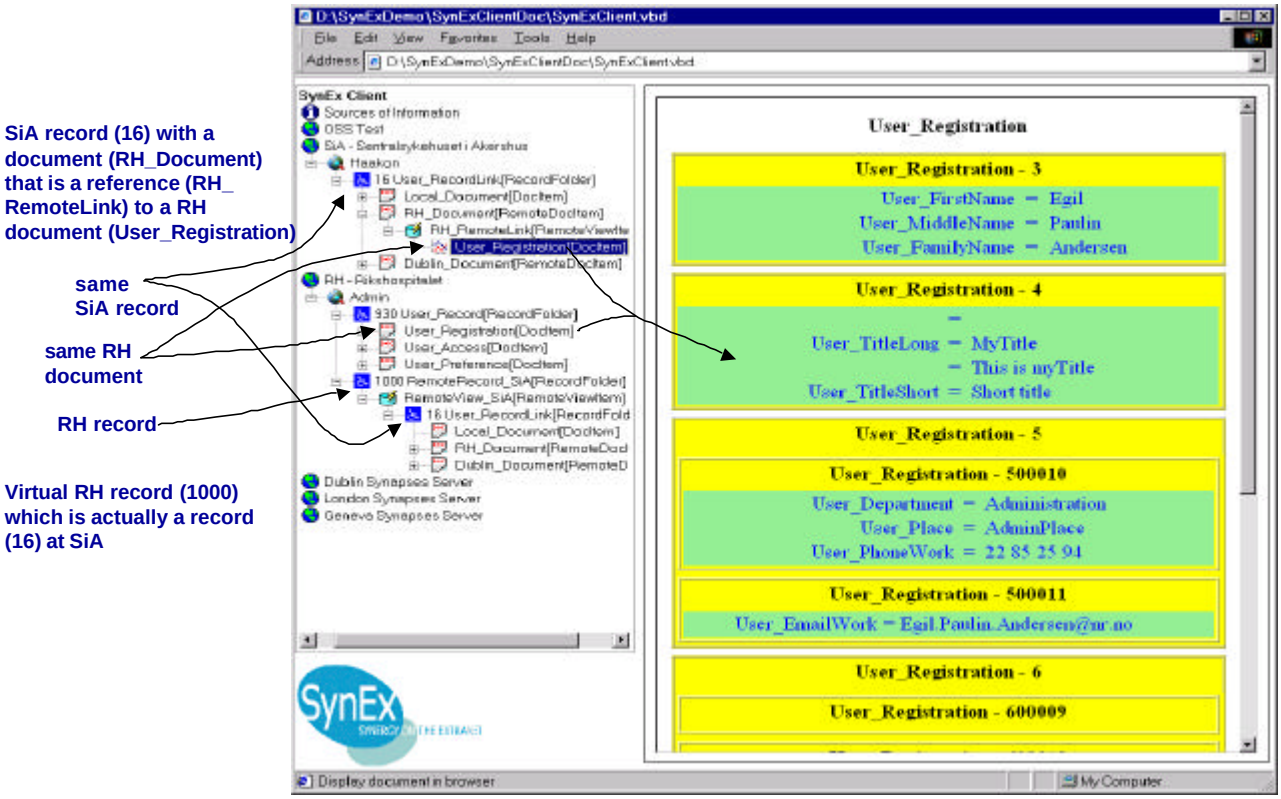


Figure 9. *Sharing healthcare records at SiA and RH.*

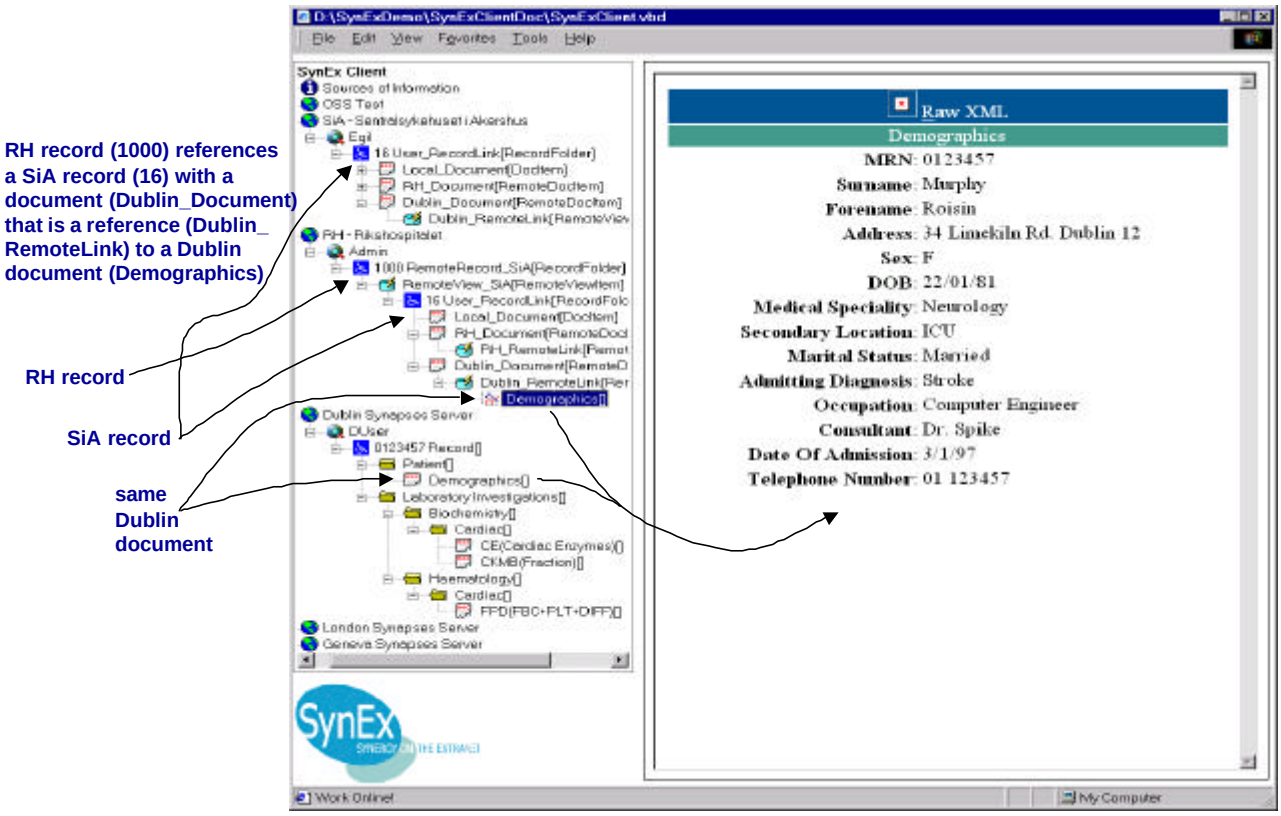


Figure 10. *Sharing healthcare records at SiA, RH and Dublin.*

PS: The GUI differences between figure 9 and 10 versus the other ones above are due to the latter examples being made in an earlier version of the client GUI components.

An important characteristic of this integration scenario is it's client-side processing, where the requests were made. The server's only responsibility is to maintain valid links to where remote parts of its records reside. Furthermore, the implementation of a particular Synapses server, and its legacy feeder systems, is irrelevant to the integration. For example, the Dublin Synapses server is based on a C++/CORBA environment connecting to various data sources via a generic database interface, while the Oslo Synapses server uses MTS (Microsoft Transaction Server) with COM (Component Object Model) components as the application layer, and SQL Server for the data store (see deliverable D2.2). Thus far no SynEx attempt has been made to standardise the web server interfaces. The Dublin web server interface is based on CGI scripts, while the Oslo web server uses ASP, but this is just a matter of implementation, both are equally suitable and could easily be exchanged or even replaced by a third one. A great benefit of basing the information exchange on XML is that it makes the technology offering this information transparent to the task of achieving seamless integration. It would have been much more cumbersome and time-consuming to achieve record integration based on a common protocol, e.g. (D)COM or CORBA (Common Object Request Broker Architecture) components.

3.5 Menus and User Functionality

In the SHS Demonstrator all user interaction takes place via pop-up menus attached to items in the tree-view control in the left HTML frame. The only exception is using a double-click to view a particular information source (under the "information" item). The following is a description of each of these menus.

3.5.1 Root Menu ("SynEx Client")

Refresh ▹ Refresh Information

The information sources listed under the "Information" item are read from a particular XML file (*"synex-information.xml"*) on the default server when starting the application. This menu selection implies a new read of this file. However, its purpose is mainly for offline use, since in that case a corresponding local file will be read ("C:\SynExClient\Common\synex-information.html").

Refresh ▹ Refresh Servers

The available servers are read from a particular XML file (*"synex-server-info.xml"*) on the default server when starting the application, and then presented as tree-view server items. This menu selection implies a new read of this file. However, its purpose is mainly for offline use, since in that case a corresponding local file will be read ("C:\SynExClient\Common\synex-server-info.html").

New ▹ New Server

Define a new server. Information required is its name (any unique name is valid), its type and its (web) address. The "type" of a server concerns how to connect to the server to retrieve SynExML information. Currently the demonstrator has prepared for four types ("Oslo", "Dublin", "London" and "Geneva"), but currently only two are ready, namely "Oslo" and "Dublin".

In addition you can specify "Web Information" as a web address with information on this server. Notice that the "Offline catalog" property of servers are **not** used in the current version.

Administration ▹ Work Offline/Work Online

This SynEx client is obviously meant for online work, either via an intranet, an extranet or internet itself, and initially it will be in online mode. However, by executing the "Work Offline" command it will also be possible to view record information based on files stored locally.

In offline mode then login is redundant and when presented with the login information box any user name and password will be accepted. When making a request (the "Record Search" command - see below) for a particular record, folder or document, this information must already exist in a file previously made by executing the "Save" command on a record, folder or document (see below).

The client uses a particular offline catalog on a local disk when working offline. The default catalog is *"C:\SynExClient\Offline"*, but this can be changed with the "Set Offline Catalog" command.

Administration ▹ Set Offline Catalog

Specify a new offline catalog; see also "Work Offline"/"Work Online" above.

Records, folders and documents that are saved to file will be stored in files under a catalog, within the offline catalog, with the same name as the server from which they are retrieved. See the "Save" command for RecordFolder, FolderRIC and ComRIC below.

Administration ▹ Set Default Server

The default server is used to retrieve information on SynEx information servers and available servers (see "refresh" above).

Administration ▹ Load Cache from File

A cache that is saved to file, i.e., via a set of ADO-XML files, can be reloaded with this command.

PS: *This command cannot be used in the current version since a complete update of the tree-view according to the new cache content has not been implemented!*

Administration ▷ Save Cache to File

The content of the client cache is saved to a set of ADO-XML files in a specified catalog on the local disk.

Debug ▷ Trace Information

When the client executes, errors, warnings and various kinds of notifications are sent to a Trace Manager component. This information can be viewed in the right HTML frame.

Debug ▷ Cache Information

This selection lets you view the current content of the cache (for debug purposes only).

About

Information on the demonstrator.

Exit

Exit the demonstrator.

3.5.2 Server Menu (a "world" icon)

Online Help

The right HTML frame will navigate to the "Web information" address of the selected server.

Properties

Various information on the selected server will be provided in the right HTML frame.

LogOn

Log on to this server by entering a user name and a password.

Edit Server

Edit the properties of this server.

Delete Server

Remove this server.

3.5.3 Login Menu (a "key" icon)

Properties

Various information on the selected login will be provided in the right HTML frame.

Record Search

Specify a record, folder or document to retrieve from the server to which this login applies.

Refresh Login

A login may time-out after a certain time period without user action; e.g. the Oslo Synapses Server will time-out after 20 minutes without client activity. This command can be used to refresh a particular login if needed.

LogOff

Log off - end this login.

3.5.4 RecordFolder Menu (a "patient" icon)

Properties

Various information on the selected record object will be provided in the right HTML frame.

Set Presentation

See the description of this command under the ComRIC menu.

Save

The specified record is saved to a file locally on the client. Notice that this implies a refresh of the cache, and notice that only the structure of a record is saved (its "shape", i.e., its documents are leaf nodes).

The record will be saved in a file with the following name:

```
record_<recordID>_shape.xml
```

where <recordID> is its unique RecordID. The file will be saved in the following catalog:

```
<offline catalog>\<server name>\<file name>
```

where <offline catalog> is the current offline catalog for the client (see "Work Offline" in the root menu), <server name> is the name of the server from which the record is retrieved, and <file name> is the file name composed as just explained.

Refresh

The current version of the specified record is removed from the cache, and then reloaded from the server.

Delete

The specified record is removed from the cache.

3.5.5 FolderRIC Menu (a "folder" icon)

Properties

Various information on the selected folder object will be provided in the right HTML frame.

Set Presentation

See the description of this command under the ComRIC menu.

Save

The specified folder is saved to a file locally on the client. Notice that this implies a refresh of the cache, and notice that only the structure of a folder is saved (its "shape", i.e., its documents are leaf nodes).

The folder will be saved in a file with the following name:

```
folder_<recordID>_<RCID>_shape.xml
```

where <recordID> and <RCID> is its unique identification within its Synapses server. The file will be saved in the following catalog:

```
<offline catalog>\<server name>\<file name>
```

where <offline catalog> is the current offline catalog for the client (see "Work Offline" in the root menu), <server name> is the name of the server from which the folder is retrieved, and <file name> is the file name composed as just explained.

Refresh

The current version of the specified folder is removed from the cache, and then reloaded from the server.

Delete

The specified folder is removed from the cache.

3.5.6 ComRIC Menu (a "document" icon)

Properties

Various information on the selected document object will be provided in the right HTML frame.

View Document

The document and its content is presented in the right frame according to the presentation format assigned to it; see "Set Presentation" below.

Cache Document

The document and all its content, i.e., all its containing RIC's and their RecordItem's, are loaded from the server and stored in the cache.

Notice that when requesting a particular record, or a folder or document within a record, with the "Record Search" command under a login, then the content of documents are not received from the server. That is, the documents are then leaf nodes in the structure of records received. Thus this command is used to retrieve the content of specific documents.

Set Presentation

The demonstrator supports two different techniques for presenting a document and its content after being received as a string of SynExML from the server, namely *XSL (eXtensible Stylesheet Language)* and *DHTML (Dynamic HTML)*.

Each document (ComRIC) can be assigned a preferred choice for presentation via this command. If no presentation (default) is assigned to a document, then the XSL specification *"default_docview.xsl"* in the catalog *"C:\SynExClient\Common"* will be used. You can replace the XSL within this file with your own default if you like. However, while the current default XSL is the same as the default XSL provided by the OSS (*"oss-document.xsl"*), the current version of SynEx Client is **not** able to utilize a default XSL specification referenced within the XML received from the server. Thus irrespective of whether the server references an XSL specification or not, if no other presentation is chosen for a particular document, via the use of this "Set Presentation" command, then *"default_docview.xsl"* will be used.

As an alternative to XSL, a HTML file containing DHTML can be assigned to a document as its presentation format. This can either be a DHTML file that includes an ActiveX control, responsible for the presentation logic, or alternatively a DHTML file utilising VBScript or JavaScript only. Chapter 7 describes how this use of DHTML works in further detail. SynEx Client's support for XSL and DHTML implies that any user can customise document presentations without having to change or recompile any components other than possibly new ones made by the user.

The presentation chosen for a particular document will be used when executing the "View Document" command above. The "Set Presentation" command is also available for folders and records. The consequence of this is that documents within this folder, or record, for which no presentation is defined, will use the presentation assigned to their closest folder, or alternatively the presentation assigned to the entire record.

Save

The specified document is saved to a file locally on the client. Notice that this implies a refresh of the cache, and notice that the entire content of the document is saved.

The document will be saved in a file with the following name:

```
document_<recordID>_<RCID>_all.xml
```

where <recordID> and <RCID> is its unique identification within its Synapses server. The file will be saved in the following catalog:

```
<offline catalog>\<server name>\<file name>
```

where <offline catalog> is the current offline catalog for the client (see "Work Offline" in the root menu), <server name> is the name of the server from which the document is retrieved, and <file name> is the file name composed as just explained.

Refresh

The current version of the specified document is removed from the cache, and then reloaded from the server. However, notice that this command does not reload the content of the specified document. For this an additional "Cache Document" command must be issued as explained above.

Delete

The specified document is removed from the cache.

3.5.7 ViewRIC2 Menu (a "pointer into" icon or a "database lightning" icon)

Properties

Various information on the selected ViewRIC2 object will be provided in the right HTML frame.

Retrieve Link Target

This command retrieves from the server the target of the specified ViewRIC2.

The target can be an entire record, a folder, a document, or also a RIC within a document. In the latter case the entire enclosing document will be retrieved since a document is the least unit of retrieval from a Synapses server.

Notice that if the target is a record or a folder then only their structure will be retrieved, i.e., their documents will be leaf nodes and "Cache Document" must be used to retrieve the content of these documents. If, on the other hand, a document or parts of a document is the target then the entire document with all its content will be retrieved and cached.

3.5.8 ViewRIC1 Menu (a "spreadsheet+database" icon)

Properties

Various information on the selected ViewRIC1 object will be provided in the right HTML frame.

3.5.9 DataRIC Menu (a "spreadsheet" icon)

Properties

Various information on the selected DataRIC object will be provided in the right HTML frame.

4. An Architectural Overview

4.1 Platform and Architecture

Figure 11 illustrates the layered architecture that has been designed and implemented in WP2. There is a client presentation and interaction layer, which is the topic of this document, a web server, an application layer, and a data layer. The web server is *IIS (Internet Information Server)* [12] with *ASP (Active Server Pages)* objects and scripts as its interface. The application layer consists of *COM (Component Object Model)* [10] components under the control of *MTS (Microsoft Transaction Server)* as the transaction server, and the data layer is an *SQL Server* database [9] containing healthcare record information. At the Oslo site the latter is the *Oslo Synapses Server (OSS)*.

Deliverable D2.2 describes this server-side platform in further detail.

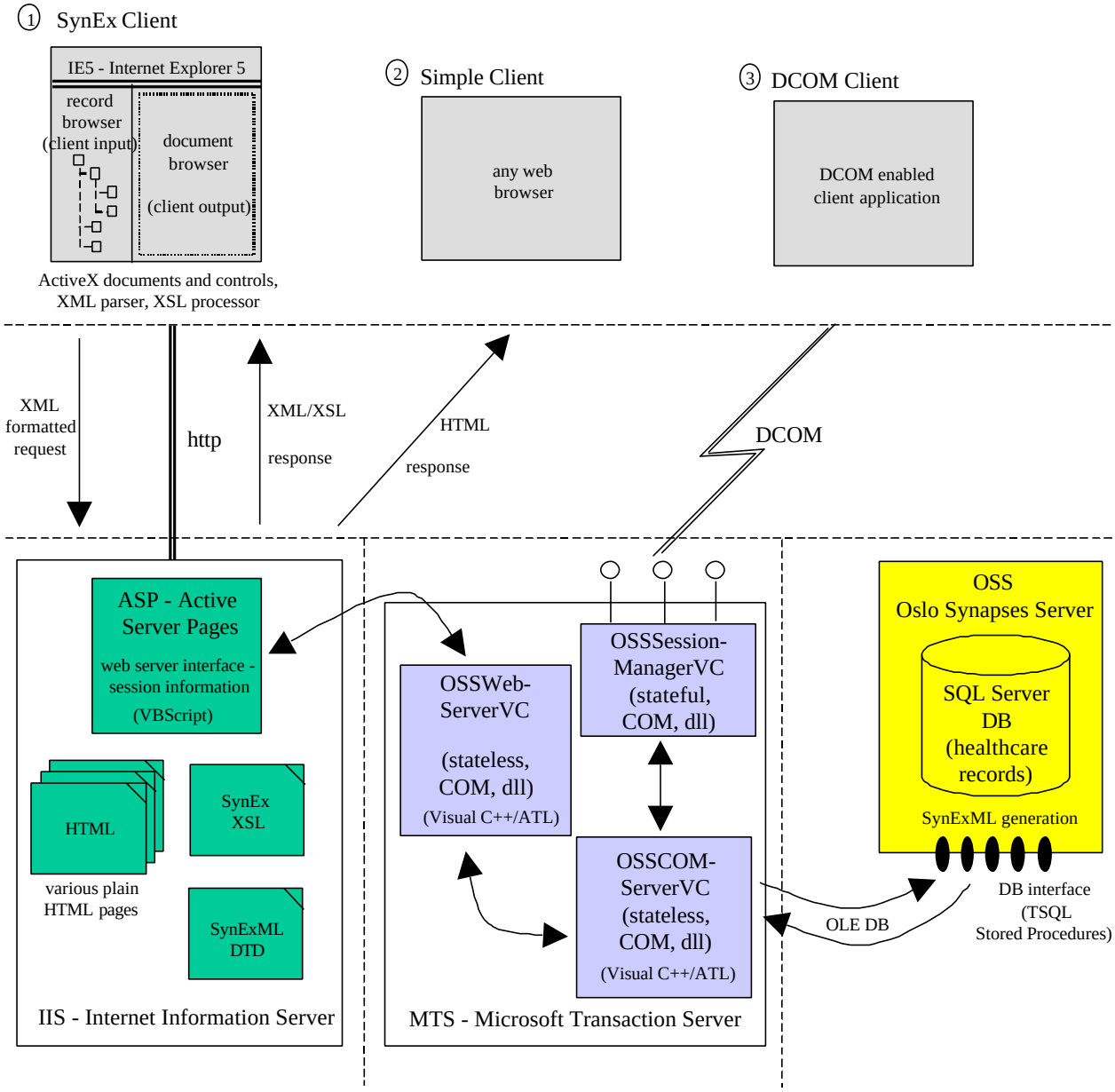


Figure 11. *The technical architecture of the SHS demonstrator..*

Client-side Components

There are a wide variety of different client types possible for a distributed information system. At one end there are highly specialised and domain dependent "thick" clients, including e.g. extensive caching, and flexible options for checking in and out information and then work on this information offline before updating the server with changes made. At the other end there are "thin" clients with common web browsers, or also ultra-light mobile devices with WAP/WML.

For reasons described below (XML+http=SOAP), clients can be made independent of the server-side technology. However, for clients based on Microsoft technology the following are some relevant examples, as illustrated in figure 11:

- *"Simple client"* - Any web browser will be able to log on to the server and then select, retrieve and update information.
- *"SynEx client"* - The client consists of a set of *ActiveX components* that execute within e.g. *Internet Explorer* as their container.
- *"DCOM client"* - The client accesses the server via *DCOM (Distributed COM)*.

The "simple client" corresponds to what can be seen as a "true" web-client, while the "SynEx client" is more like a traditional client application using web protocols for server interaction. The "simple client" may also include WAP based mobile devices with WML browsers. For reasons described below, DCOM is not a particularly interesting option unless there is a very close relationship between the client and the server with respect to whom has developed them, and which servers the clients will connect to. The "SynEx client" alternative has been chosen in WP2.

Network Protocol

To make record information available on an extranet using common internet technology, the internet protocol *http (HyperText Transfer Protocol)* is used for client-server interaction both ways.

XML for Exchanging Healthcare Record Information

XML [6] has the power to become the independent data exchange format of the future. The use of XML to exchange data between heterogenous systems provides support for hierarchically structured patient data, user defined tags and machine-understandable assertions for searching, reasoning and analysing healthcare information like federated healthcare record objects.

An XML DTD, called the *SynExML (SynEx Markup Language)*, has been defined within the SynEx project to be used for inter-site exchange of FHCR information. That is, SynExML is the basis for semantic interoperability between SynEx components that relate to FHCR information. The 2.1 beta 4 version of SynExML, which is likely to become the final version for the duration of the SynEx project, is included in appendix C of this document. SynExML is based upon the generic FHCR structure defined within the Synapses project [3][4][5], and most of its elements and attributes correspond one-to-one with record component concepts and properties defined within the Synapses Server specification.

Microsoft currently works on a specification called *SOAP (Simple Object Access Protocol)* [7][8] where the communication between a client and a server is formatted as XML over http both ways; i.e., as opposed to e.g. *DCOM (Distributed COM)* or *IIOP (Internet Inter-ORB Protocol* for *CORBA* [15]*)* also requests from a client to a server is formatted as XML (as functions with arguments) which can then be parsed by the server and acted upon. There are several advantages by this despite its functional simplicity relative to DCOM and IIOP.

http is a simple protocol with good coverage and few demands on the client, and, not the least, most firewalls are readily configured for common security options dealing with well known internet protocols and ports. This as opposed to DCOM or IIOP for which firewalls can pose a problem. In practice, the ability for remote machines to interact via DCOM and IIOP is more limited. That is, DCOM and IIOP can be well-suited for computers within e.g. a limited area, but not between "any" remote client and server on the internet.

Since XML amounts to strings of text it is well-suited for transmission via http, and the great benefit of SOAP's combined use of XML and http is that it makes the underlying client- and server-side technology transparent to each other. Thus similar to how component technology like COM provides for programming

language independence and technical interoperability locally, SOAP provides for platform independence and technical interoperability globally.

Furthermore, WML which is used for accessing information from e.g. WAP based mobile phones is itself XML, i.e., it is XML according to a particular XML schema definition. Thus information transmission with XML is well-suited for such mobile clients.

Since client requests received by a server is formatted as XML, instead of being received as e.g. low-level RPC's, a lack of complete client-server version compatibility can be handled more gracefully. For example, if an "old" client uses a "newer" server then the server may support a slightly different or extended set of requests than those requested by the "old" client. However, to the degree that the server is able to understand the "old" client's request, despite that it may not fully correspond to a request as expected by the "newer" server, the server may still be able to serve the "old" client. This is not possible with e.g. COM requests where e.g. a missing function argument leads to an immediate error.

4.2 Distribution and Integration

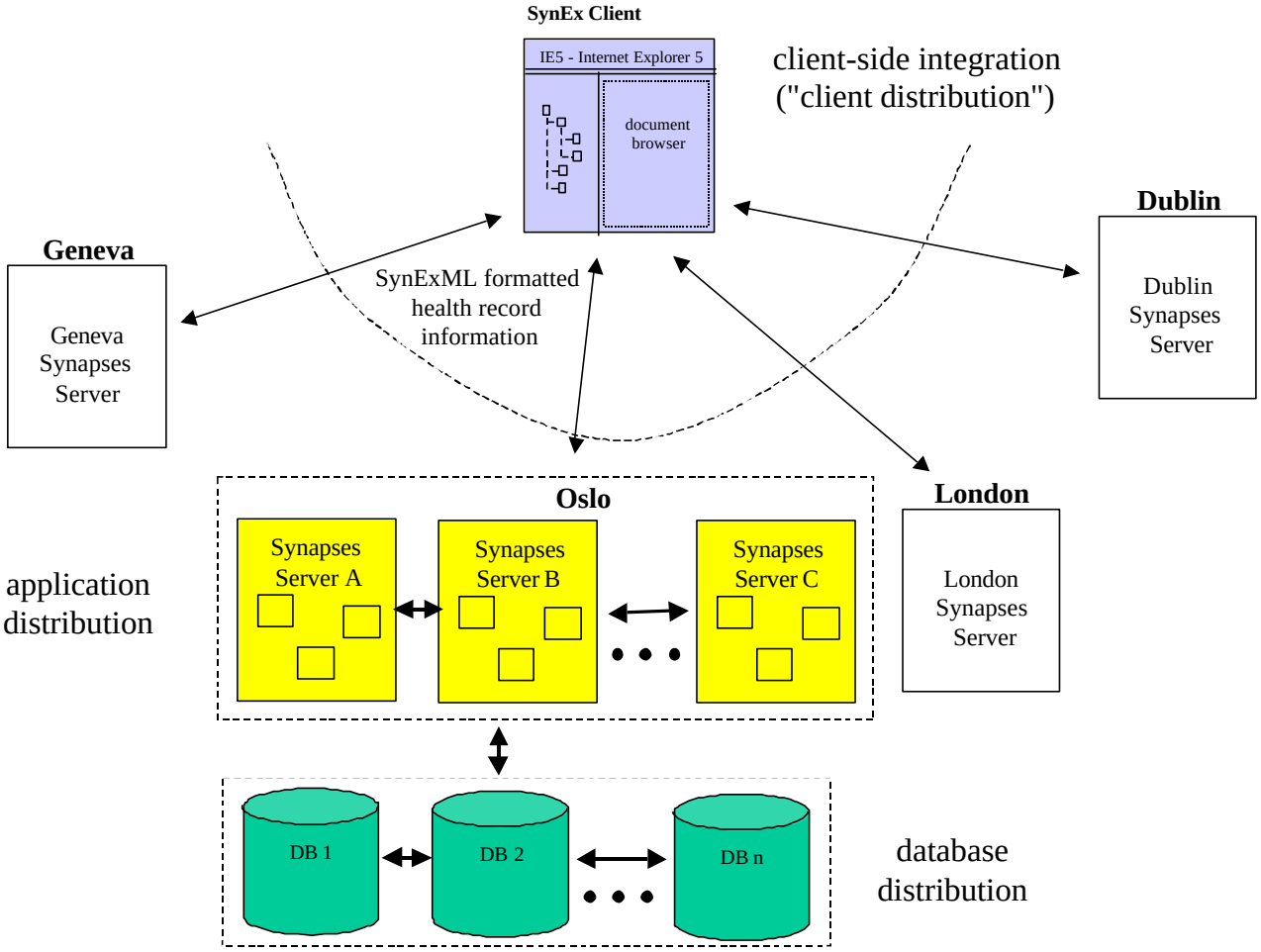


Figure 12. *Client versus Application versus Database Distribution.*

Shared Federated Healthcare Records in SynEx

The Oslo Synapses Server (OSS) and the Synapses Server specification does not rely on a central catalog service for retrieving information on where various parts of a particular patient record resides. Instead this information is distributed such that every record on a particular server has the information required to access other parts of it that exists on other servers. Thus the hyperlink capabilities of OSS and Synapses, together with XML and web technology as explained above, offer a good basis to realise shared federated healthcare records.

Client versus Application versus Database Distribution

We can distinguish between different kinds of distribution in an information system architecture, namely database-, application- and client distribution. Figure 12 illustrates this.

Database distribution utilises e.g. SQL Server distribution features, while distribution at the application layer is handled by MTS as a collaboration of MTS objects that reside on different computers. Distributed transactions in both MTS and SQL Server are handled by the same DTC (Distributed Transaction Controller).

By *"client distribution"* is here meant that distributed information is integrated at the client level, i.e., as the result of a client's request for related but distributed information (e.g. distributed information on a particular health record). The application and data layers are not involved in the distribution except that they may contain information on how and where to get access to related information that resides elsewhere.

Database distribution should be transparent to the application layer, while distribution at the application layer should be transparent to its clients. Typically, the database and application layer distribution are "local" distribution in the sense that the databases or application components exist on computers in physical proximity, or at least the same development organisation is in charge of configuring each participant in the distribution.

Client side integration may well be an integration of globally distributed information where the organisation offering the server side functionality and information may have no knowledge of who are the clients, and the client devices can be common web browsers, mobile phones, etc, making a connection to the server from anywhere in the world.

Synapses records are well-suited for this kind of client side integration since *we do not foresee any need for client-side transactions to span more than those parts of a record that reside in a single server*. That is, this is a constraint that in our experience will not severely hamper the users work with health record information, but it greatly simplifies the transaction handling of globally distributed health records.

5. SynEx Client Components

5.1 Implementation, Source Code and Project Files

All components that constitute the SynEx Client are implemented in Visual Basic. The reason for choosing Visual Basic for the client, as opposed to Visual C++ used for the server-side components (see deliverable D2.2), is its much better cost-effectiveness in development time relative to Visual C++. Performance requirements on the client-side does not mandate the use of Visual C++, and despite being more inflexible than Visual C++ it is sufficient for our purposes.

However, the interfaces of SynEx Client components are specified in IDL, but Visual Basic and IDL are not fully compatible. That is, the IDL specification is converted into a Microsoft type library with the MIDL compiler, and then this type library is used by Visual Basic via its "Implements" statement. There are two disadvantages, however. One is that outgoing interfaces(events) and connection points cannot be implemented in Visual Basic based on IDL specifications. Thus the correspondence between the IDL specification provided here and the Visual Basic implementation is not 100%. More specifically, events raised by the CacheManager component within the "VBClientCache.dll" module can only be received via a "WithEvents" reference to the CacheManager object itself, not via a reference typed according to the IDL specification (e.g. IsxcmanCacheManager). Furthermore, Visual Basic cannot implement IDL interface inheritance. Thus the inheritance relationships illustrated in the object model in figure 17 below cannot be implemented in Visual Basic. Instead the Visual Basic components will use several "Implements" statements to implement each of the involved interfaces separately.

Source Code

The source code for the current implementation of the SynEx Client comes as a zip file named:

*oss-client-wp2-v**n.m**.zip*

where the bold ***n.m*** states the version. When extracting the content of this file, the following catalogs are produced:

- *ClientFiles* : Contains four catalogs "Common", "DHTML", "HTMLapplication" and "Samples" that must exist on a client computer running SynExClient. See below for more details.
- *SynExClientIDL* : The IDL specification for the SynEx Client components. It exists as part of a Visual C++ project in order to use the MIDL compiler to generate a corresponding type library.
- *ClientTypeLib* : The type library generated by the Visual C++ project in the above SynExClientIDL catalog is placed in this catalog. The Visual Basic implementation of the components reference this type library.
- *VBTraceManager* : The Visual Basic project for the "VBTraceManager.exe" module.
- *VBSynExProvider* : The Visual Basic project for the "VBSynExProvider.dll" module.
- *VBFHCRProvider* : The Visual Basic project for the "VBFHCRProvider.dll" module.
- *VBClientCache* : The Visual Basic project for the "VBClientCache.dll" module.
- *VBSynExClient* : The Visual Basic project for the "VBSynExClient.ocx" ActiveX control.
- *VBInfoView* : The Visual Basic project for the "VBInfoView.ocx" ActiveX control.
- *VBDocView* : The Visual Basic project for the "VBDocView.ocx" ActiveX control

The projects should be compiled in the following sequence:

1. *SynExClientIDL* **Notice:** Only compile the IDL file (with the MIDL compiler) - not the project! Then copy the "SynExClientIDL.tlb" file to the ClientTypeLib catalog.
2. *VBTraceManager*
3. *VBSynExProvider*

4. *VBFHCRProvider*
5. *VBClientCache*
6. *VBSynExClient*
7. *VBInfoView*
8. *VBDocView* (this is an example for demonstration purposes only)

SynEx Client Files

When executing the SynEx Client it depends on a number of files under the root catalog *"C:\SynExClient"*. These files must be organised into the following four catalogs, corresponding to their organisation in the *"ClientFiles"* catalog of the above zip file for source code distribution:

- *C:\SynExClient\Common* : files required for offline operation
- *C:\SynExClient\DHTML* : files required for user output in the right frame
- *C:\SynExClient\HTMLapplication* : files for executing the SynEx Client application (*"synexclient-main.html"* is the application home page)
- *C:\SynExClient\Samples* : some DHTML demonstration files (uses the *VBDocView* control)

5.2 Client Component Architecture

5.2.1 An Outline

Figure 13 illustrates an outline of the principal architecture for the SynEx Client. Notice the difference between the application dependent versus the application independent components. The GUI components are specific to the SHS Demonstrator, and, as already mentioned, they have not been emphasised in this project (not production quality). The other components, i.e., a client cache, components for connecting to various Synapses webservers and components for parsing received XML, are application independent. They can be reused, as COM components, in healthcare information systems that require access to healthcare records. They are the main software deliverable in WP2 (for the client-side).

The application independent components are divided into three different kinds of components. One set of components implement a cache on the client for storing retrieved healthcare records. This makes it possible to work with the records in offline mode. A user can also at any time save the current cache to a set of files on the local file system, and then reload this cache later on. The current cache is implemented with *ADO (Microsoft Active Data Objects)* [11], but the implementation of the cache is transparent to other client components. Thus the cache can be implemented by the use of some other suitable technology, e.g. XML DOM object structures, without affecting other parts of the client.

The set of components named "Data Provider"'s are responsible for communication with different kinds of webservers offering record information formatted in SynExML. For example, connecting to the OSS webserver (IIS with ASP and SOAP) is different from connecting to the Dublin webserver (CGI-scripts). Webserver requests have also not been standardised (yet) within SynEx. Thus a different Data Provider component is made for each type of webserver, and one Data Provider for offline mode (accessing local storage instead of a webserver).

Finally there is a set of XML parser components, one component for each kind of XML that can be received. These are responsible for parsing XML received via a Data Provider, and then inserting the results into the cache. For example, there is one parser component for SynExML, one for information on available Synapses servers, and one for information on SynEx information sources.

The purpose of such a categorisation of components is to allow one component to change without this having unforeseeable consequences on other components. For example, minor changes in the SynExML, that does not affect information content, can be carried out in the SynExML parser component without affecting other parts of the client.

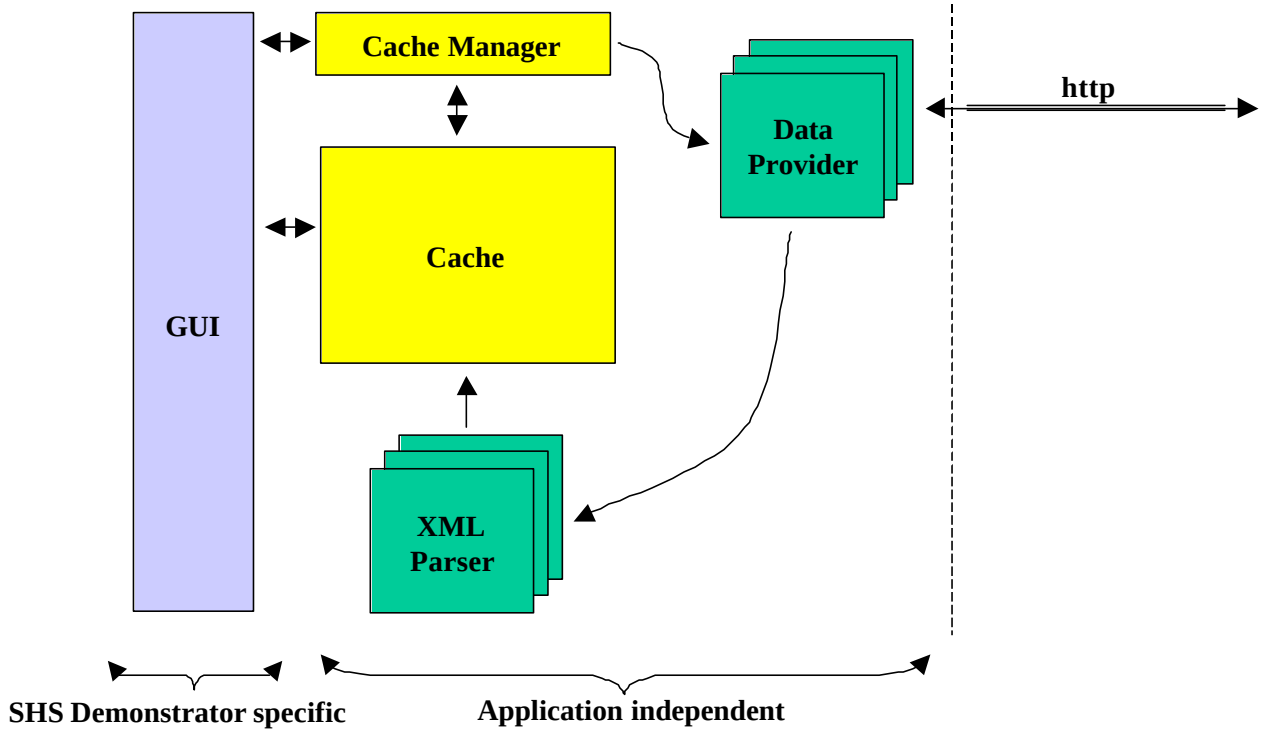


Figure 13. *An outline of the SynEx Client component architecture.*

5.2.2 The SynEx Client GUI

Figure 14 illustrates the organisation of the GUI components, which amounts to a set of HTML pages with ActiveX controls. These are specific to the SHS Demonstrator. The application start page is *"C:\SynExClient\-HTMLapplication\synexclient-main.html"*, which contains two frames. The left frame (frame1) is always *"C:\SynExClient\HTMLapplication\synexclient-frame1.html"*, which contains the ActiveX control *"VBSynExClient.ocx"*. As illustrated in chapter 3, this is a tree-view control through which user interaction takes place.

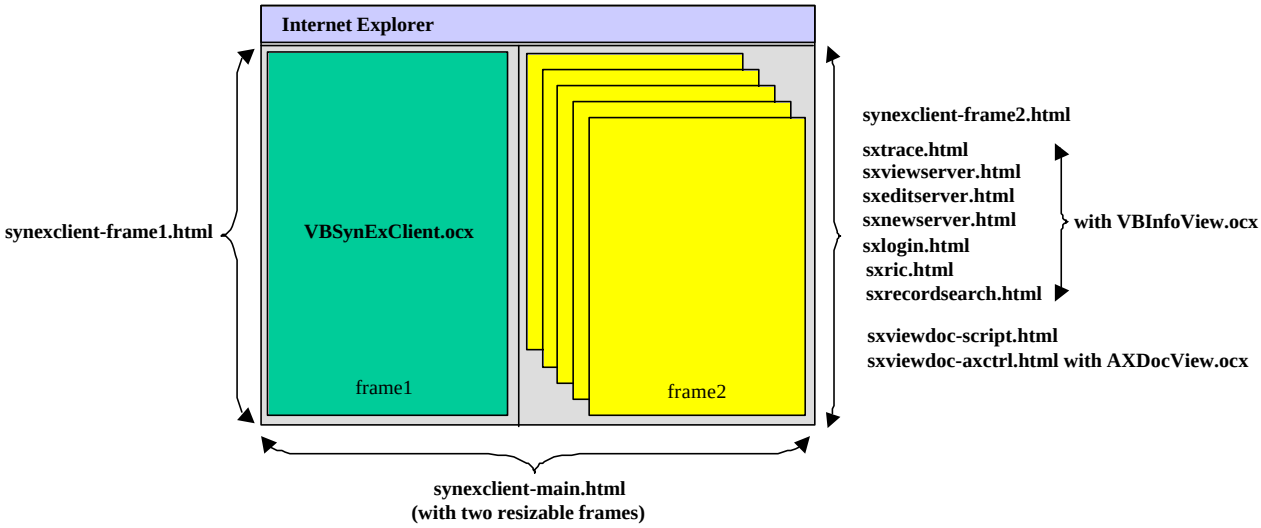


Figure 14. *The SynEx Client graphical user interface, which is specific to the SHS Demonstrator.*

During application start-up then the right frame (frame2) is *"C:\SynExClient\HTMLapplication\synexclient-frame2.html"*. However, once the *"VBSynExClient.ocx"* control is loaded and ready, it will take control of the right frame and navigate to various other pages either locally or on the web. Thus the right frame can be seen as a regular browser window under the control of the ActiveX control in the left frame.

Most of the output provided in the right frame will be based on DHTML files residing in the catalog *"C:\SynExClient\DHTML\sx<....>.html"* (where <.....> are different names like "trace", "viewserver", etc). These files contain the ActiveX control *"VBInfoView.ocx"*, which interacts with the *"VBSynExClient.ocx"* control in the left frame. By designing new XSL specifications and/or new DHTML pages with or without new ActiveX controls (or Java Applets), it is possible for a client to completely control presentations in the right frame without changing any of the existing GUI components. Chapter 7 describes this in further detail.

5.2.3 SynEx Client Modules and Components

Figure 15 illustrates a more detailed overview of the modules and components that constitute the SynExML Client, and a reference to the COM interfaces that they implement. Chapter 6 provides a more detailed description of each of these interfaces, and the object model that they constitute.

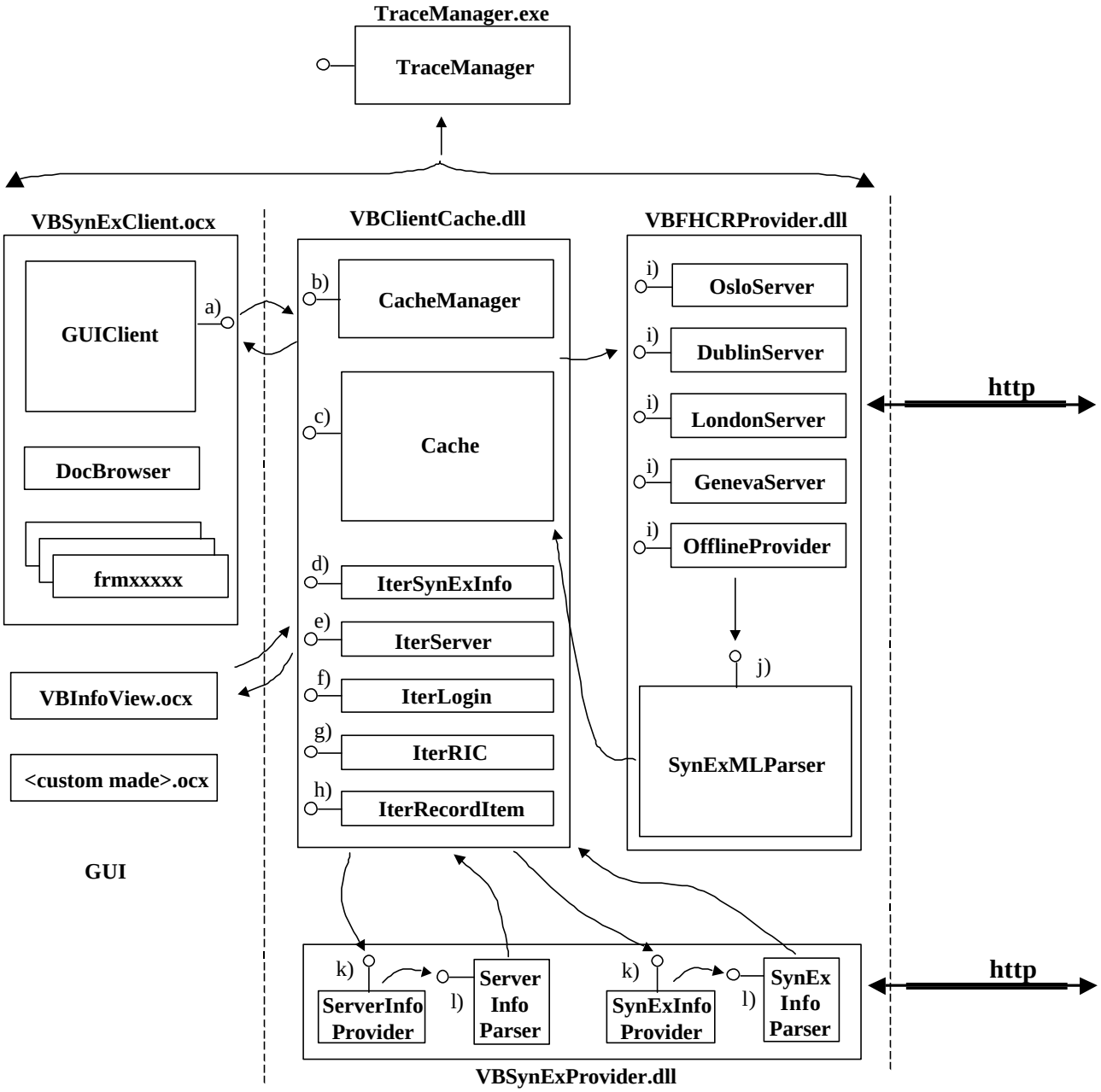
Notice that we will use the term *"module"* for dll, ocx (ActiveX controls) and exe files that contain a set of COM components, and the term *"component"* for COM co-classes, VB class modules, VC++ classes, etc, from which objects can be instantiated.

The *"VBSynExClient.ocx"* GUI ActiveX control, residing in the left frame, contains the *GUIClient* component which handles most of the user interaction via its tree-view control. It also contains the *DocBrowser* component which is responsible for information output in the right frame. That is, DocBrowser interacts with other GUI ActiveX controls like *"VBInfoView.ocx"* and *"<custom made>.ocx"* (e.g. *"VBDocInfo.ocx"*) as will be explained in chapter 7 below. In addition it contains a number of Visual Basic forms, *"frmxxxxx"*, for various user interaction.

The *"VBClientCache.dll"* module contains the components *CacheManager* and *Cache* which together constitute the client cache. The cache is currently implemented by a set of ADO Recordsets, and the next section below describes its implementation in further detail. The cache also provides a number of iterator components for objects that a client will use to traverse and access cached information.

The *"VBFHCRProvider.dll"* module contains a set of data provider components and an XML parser component for SynExML. That is, relative to figure 13, data providers and the XML parser components that they need are implemented within the same module in this case since we do not foresee any need to distribute them independently as two separate modules. A data provider is made for each of the SynEx server types; i.e., "Oslo", "Dublin", "London" and "Geneva". At the moment only the "Oslo" and the "Dublin" servers are available. In addition there is a separate data provider for use in offline mode. This provider accesses files on the local file system instead of making requests to a webserver. SynExML formatted XML received by one of these data providers, or read from file by the OfflineProvider, are delivered to the *SynExMLParser* component. This component is responsible for parsing the XML and inserting the record information into the cache.

The *"VBSynExProvider.dll"* module is similar to the *"VBFHCRProvider.dll"* in the sense that it contains two data provider components and two XML parser components. The *ServerInfoProvider* component is used to retrieve information on available servers, and the XML that it receives is parsed by the *ServerInfoParser* component. Similarly, the *SynExInfoProvider* component is used to retrieve information on SynEx information sources, and *SynExInfoParser* is used to parse the XML received for this.



a) IsxguiLoginInfo

b) IsxcmanCacheManager

c) IsxcshCreateCacheObjects
IsxcshCacheSearch

d) IsxcshSynExInfo
IsxcshCollection

e) IsxcshLogin
IsxcshCollection

f) IsxcshServer
IsxcshCollection

g) IsxcshRICShape
IsxcshRICInformation
IsxcshRICOperation
IsxcshRecordFolder
IsxcshFolderRIC
IsxcshComRIC
IsxcshViewRIC1
IsxcshViewRIC2
IsxcshDataRIC
IsxcshCollection

h) IsxcshRIShape
IsxcshRIInformation
IsxcshRIOperation
IsxcshRecordItem
IsxcshCollection

i) IsxprvFHCRLogin
IsxprvFHCRInformation

j) IsxxmlFHCRParser

k) IsxprvInformation

l) IsxxmlParser

Figure 15. *A detailed overview of SynEx Client modules and components.*

Read-Only Information Access

The current version of both client- and server-side components in the WP2 deliverable is prepared just for read-only access to Synapses healthcare record information. The reason for this is that within SynEx we have not (yet) standardised on write access to record information. It will be simple, technically, to extend access to the Oslo Synapses Server to include write access; including both record updates as well as the creation of new records and documents. This just amounts to invoking existing functions for this in the OSS. The only limitation regarding write access is that, as explained in section 4.2, no transaction should span more than those parts of a record that reside in a single server. This is also a constraint that applies to the current version of OSS itself.

5.3 Client Cache

The client cache is implemented by a set of *ADO recordset's*, as illustrated in figure 16. That is, each entity "rs<....>" in the diagram represents a particular recordset. All the attributes of each recordset are listed within the entity, but associations are also illustrated in order to make it easier to understand their relationships (but these are all implemented by the attributes listed!). Sets of attributes in bold constitute unique (candidate) keys for each recordset.

For performance reasons these recordsets are not fully normalised; e.g. the ServerID attribute of rsRICShape can be derived from its LoginID attribute, and so on. Furthermore, a different recordset is used for each of the different kinds of RIC's, and both RIC's and RecordItem's are split into a structural part ("shape") and their information carrying part. This was done to avoid too large recordsets, but practice seems to indicate that there are no performance problems (with respect to recordset search) for the number of rows involved here. Due to the encapsulation of the VBClientCache module, any changes to the cache implementation can be performed without affecting other parts of the client.

Most of the recordsets and their attributes correspond one-to-one with the Synapses healthcare record specification (or in this case, the SynExML). *rsServer* and *rsLogin* stores servers and logins, respectively, while *rsSynExInfo* stores information on SynEx information sources. However, *rsLoginRootRIC* is a recordset for implementation reasons only. It is used to identity RIC's that are the root of a particular part of a record that is cached. For example, if an entire record is cached then its RecordFolder RIC will be the root, while if only a particular folder or document within a record is cached, not the entire record, then this FolderRIC or ComRIC will be a "root RIC" in this sense. The purpose of identifying such "root RIC's" is that this simplifies updates to the tree-view control after new records or parts of records are cached.

Finally, while most attributes are also self-explanatory relative to the Synapses healthcare record specification, the *status* attribute of the RIC and RecordItem recordsets only exist for implementation purposes. The cache does not support concurrent multi-user access, but it must be able to recover from situations where there are errors in the XML received. That is, during the parsing of received XML the XML parser component will insert record information into the cache via a number of insert functions (see chapter 6). If an error occurs then it must be possible to rollback any changes made, and the *status* attribute is used for this. When a new RIC or RecordItem is inserted it is first stored with status=*2* (new), and if the same RIC already exists then all RIC's underneath it, and all its RecordItem's, have their status changed from 0 (active) to 1 (old). If the overall cache operation is eventually committed then all RIC's and RecordItem's with status=1 (old) are removed, while all with status=2 (new) are changed into 0 (active). Alternatively, if the overall operation is rolled back then those with status=1 (old) are changed into 0 (active), while those with status=2 (new) are removed.

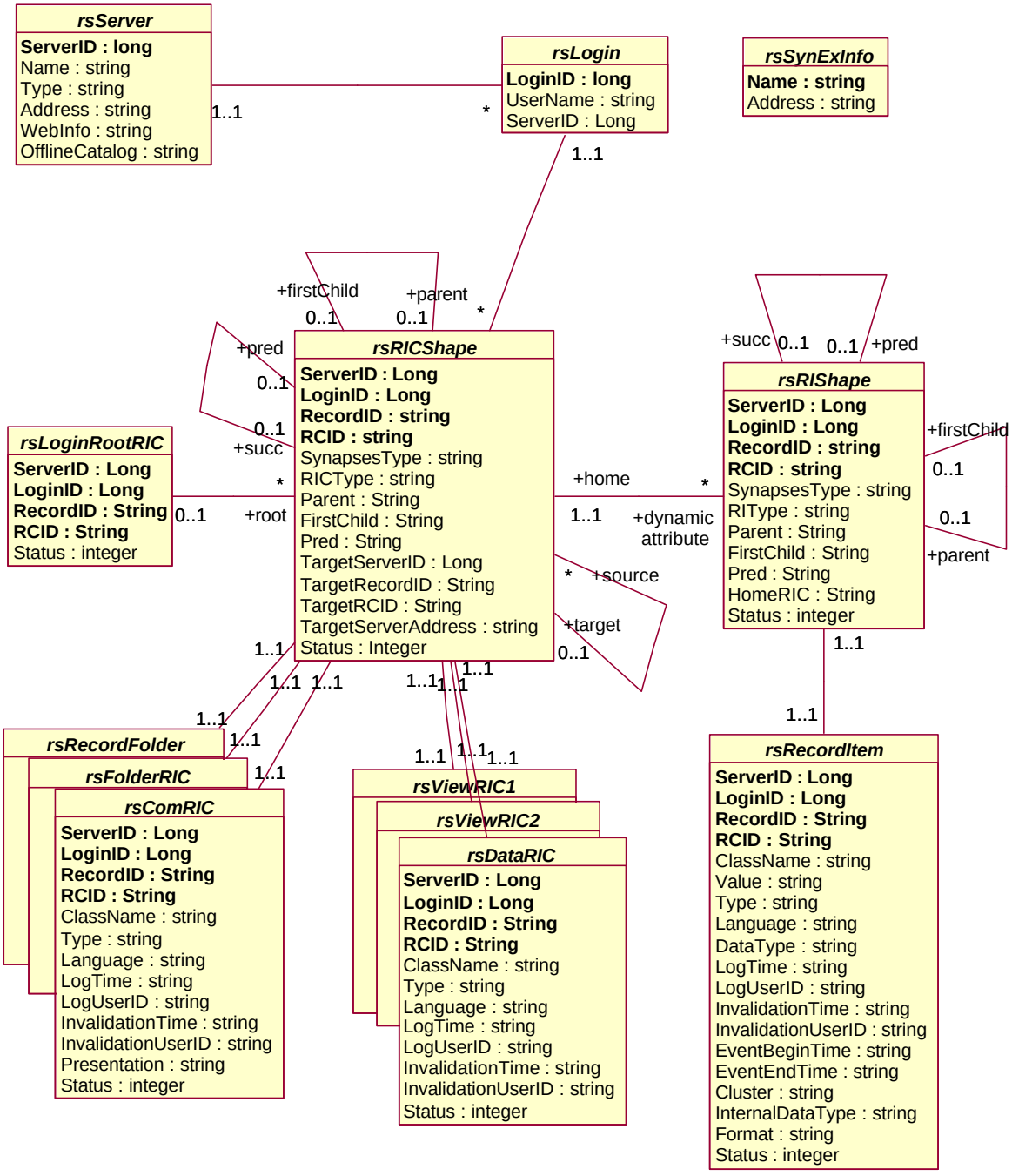


Figure 16. *ADO recordset's in the cache.*

6. Client Object Model

6.1 Cache and CacheManager

Figure 17 illustrates the object model provided by the cache and the cache manager components, and the following is a brief description of each of the interfaces involved. Notice that many of the functions in these interfaces correspond one-to-one to menu commands in the tree-view control. These are already described in section 3.5.

As mentioned above, Visual Basic cannot implement IDL inheritance relationships. Thus the inheritance relationships in this object model are not actually implemented by our Visual Basic components. Instead the iterator components in figure 15 implements the inheriting interfaces individually.

IsxcmanCacheManager

This interface is implemented by the CacheManager component, and its functions are already described in the above. The boolean result of the *SaveCache*, *LoadCache*, *LoadServerInfo* and *LoadSynExInfo* functions indicate success or failure.

IsxcshCreateCacheObjects

Functions within this interface are used to fill the cache with SynExML formatted record information. For Synapses servers that does not provide the RecordID attribute (which is optional in the SynExML) then the *GenerateNewRecordID* function can be used to generate a RecordID unique within the current cache. The RecordID of such records, or record parts, can not be used to later identity this record at its original server, however.

The *PostRecordCaching* function must be called after a caching operation to indicate success/commit (inAction="Commit") or failure/rollback (inAction="Rollback") of the entire operation.

IsxcshCacheSearch

This interface contains functions to search for particular objects, or collections of objects, in the cache.

IsxcshCollection

This is an interface with functions to traverse a collection of objects.

Notice that each of the iterator components, e.g. *IterRIC*, can be used to represent both a single particular object, e.g. a particular RIC, and also a collection of such objects, simultaneously. This because the iterator components all implement this interface beside its other corresponding iterator interface, e.g. IsxcshRICShape.

A function that returns a collection of objects will never return "null", i.e., it will always return an iterator representing the collection, but if the collection is empty then its *MoveFirst* function returns "false".

Notice also that in the diagram, the type of the objects that participate in a particular collection are written in square brackets after the *IsxcshCollection* interface.

IsxcshSynExInfo

Access to properties of SynEx information sources.

IsxcshServer

Properties and functions of available servers.

IsxcshLogin

Properties and functions of logins. The Cache and Get functions use the corresponding functions of the data provider component that correspond to the login's server type.

IsxcshRICShape

This interface can be used to traverse a structure of RIC objects. When an iterator object represents a particular RIC object, then after invoking its *GoParent* function it will represent the parent RIC object of this object, if any exists. The boolean return value indicates whether the traversal succeeded or failed. In case of failure the iterator will represent the same object as before the function call.

The *RecordItems* function returns a collection of all RecordItems attached to the RIC object. The *Sources* function returns a collection of all ViewRIC2 objects, if any, that references this RIC object as its target. For ViewRIC2 objects then the *Target* function will return an iterator for its target RIC object.

Notice that when using the *Clone* function then if the iterator objects also represents a collection of objects, the new iterator object will not copy this collection. It will only be a new iterator for the single RIC object represented by the iterator object being cloned.

The *SynapsesType* property returns the RIC object's basic Synapses SynOM type, e.g. "RecordFolder" or "ViewRIC2", while its *RICType* property may return an OSS specific specialisation of these basic Synapses SynOM types.

IsxcshRICInformation

This interface contains common RIC properties.

The *Type* property corresponds to the *RICType* property of IsxcshRICShape.

IsxcshRICOperation

This interface contains common functions for RecordFolder, FolderRIC and ComRIC objects. They correspond to the menu commands available for corresponding tree-view items, as described in section 3.5.

IsxcshRecordFolder

This interface contains functions concerning the presentation of documents within a record (RecordFolder); see section 3.5.

IsxcshFolderRIC

This interface contains functions concerning the presentation of documents within a folder (FolderRIC); see section 3.5.

IsxcshComRIC

The *Get/SetPresentation* functions concern the presentation of the document, as described in section 3.5. The *CacheContent* function can be used to cache the content of the document. The *GetAttributes* function returns a collection of every RecordItem of a particular class (specified by the *inClassName* argument) within the document. Notice that this function returns RecordItems attached to any RIC object within the document, not just those attached to the document root ComRIC object.

IsxcshViewRIC1

This interface has no functions.

IsxcshViewRIC2

This interface contains properties and functions applicable to ViewRIC2 objects.

The *CacheTarget* function will cache the targeted record, folder or document, and the boolean result indicates whether this succeeded or not.

IsxcshDataRIC

This interface has no functions.

IsxcshRIShape

This interface is similar to the IsxcshRICShape interface, and can be used to traverse a structure of RecordItem objects.

The *HomeRIC* function returns an iterator for the RIC object to which the RecordItem object belongs.

IsxcshRIInformation

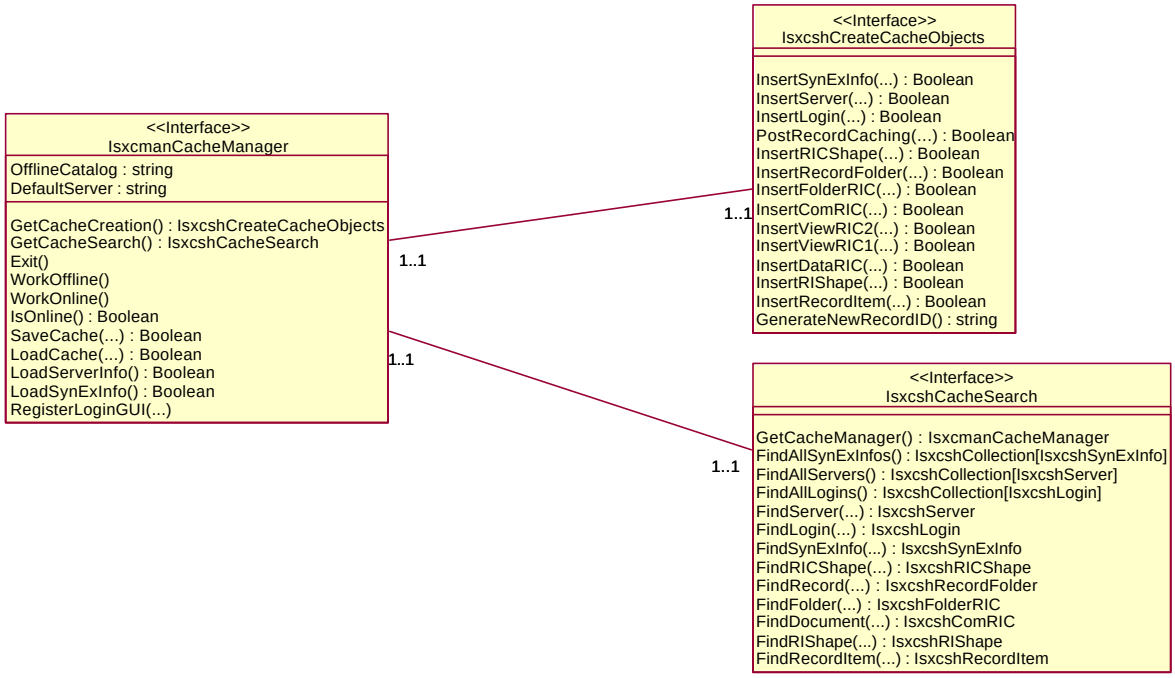
This interface contains common RecordItem properties.

IsxcshRIOperation

This interface has no functions.

IsxcshRecordItem

This interface has no functions.



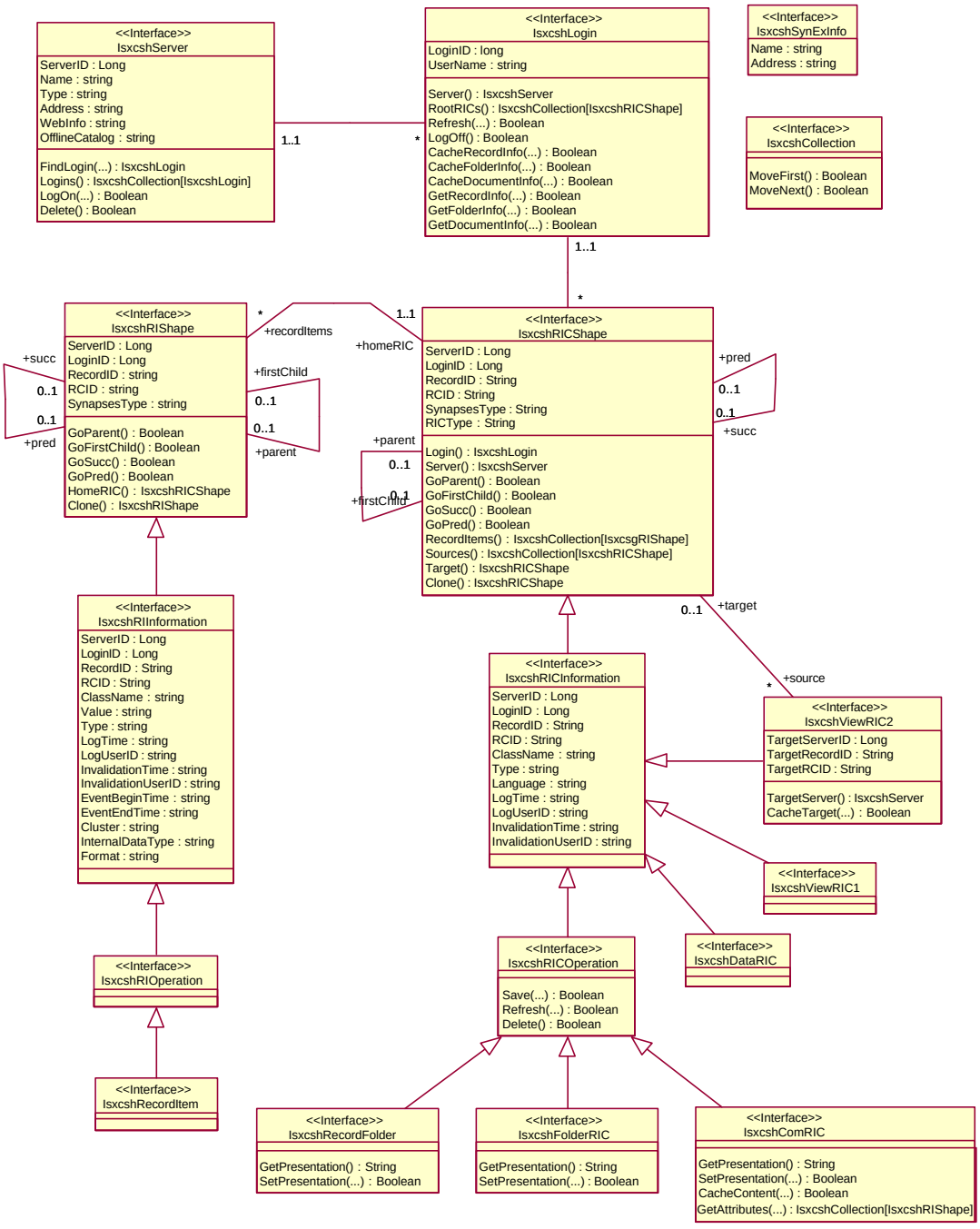


Figure 17. *The object model provided by the cache and the cache manager components.*

Outgoing Interfaces/Events

As mentioned above, Visual Basic does not support the implementation of outgoing interfaces specified in IDL. Thus the *IsxoutEvents* interface in the IDL specification (in appendix A) is not actually implemented, but instead the *CacheManager* component itself provides the corresponding events along a "WithEvents" reference typed as *CacheManager* (instead of being typed *IsxoutEvents*).

Notice that the *RICAdded/RICRefreshed/RICRemoved* events only trigger for the *"root RIC"* of a particular record, or part of a record, as explained in section 5.3. They do **not** trigger for each individual RIC added/refreshed/removed.

IsxguiLoginInfo

This is like an outgoing interface from the *CacheManager* component in the sense that it requires its client, which in this case is the *GUIClient* component (figure 15), to implement this interface. A client of CacheManager must provide a reference to its *IsxguiLoginInfo* interface by calling the *IsxcmanCacheManager* function *RegisterLoginGUI*. *CacheManager* uses this interface to make a request for login information; i.e., user name and password.

6.2 Data Provider and XML Parser Components

In addition there are the following interfaces supported by the data providers and the XML parsers as illustrated in figure 15.

IsxprvFHCRLogin

This interface is implemented by each of the SynExML data providers in figure 15; i.e., the offline mode provider, and each of the different server type providers. It has functions for logging on and off a particular webserver address.

IsxprvFHCRInformation

This interface is also implemented by each of the SynExML data providers in figure 15. It has functions used by its clients to retrieve SynExML on a particular record, folder or document. The difference between the Cache versus the Get functions is that the Cache functions are used to store the retrieved information in the cache, via the use of the SynExML parser component, while the Get functions return the result as an XML string (inResponse="xml") or an HTML string (inResponse="html"). If inRetrieval="all" then document contents are also included, while inRetrieval="shape" returns only the structure of records/folders/documents and not document content.

Notice that the *SetContextInfo* function must be called before using the Cache functions in order to provide the provider with a reference to the cache, and information on to which login and server the information belongs.

IsxprvInformation

This interface is implemented by the *ServerInfoProvider* and the *SynExInfoProvider* components. The *CacheInformation* function will retrieve the corresponding information and store it in the cache.

The *SetContextInfo* function has the same purpose as in the *IsxprvFHCRInformation* interface, and must be called before the *CacheInformation* function.

IsxxmlFHCRParser

This interface is implemented by the *SynExML* parser component. Its *ParseAndCache* function receives a XML DOM reference and the result is inserted into the cache identified by first calling its *SetContextInfo* function.

IsxxmlParser

This interface is similar to the *IsxxmlFHCRParser* interface except that its *SetContextInfo* function only requires information on which cache to insert the results.

7. Customising Document Presentations

As already mentioned under the "Set Presentation" command in section 3.5, the demonstrator supports two different techniques for presenting a document and its content after being received as a string of SynExML from the server, namely *XSL (eXtensible Stylesheet Language)* and *DHTML (Dynamic HTML)*. By designing new XSL specifications and/or new DHTML pages with or without ActiveX controls[1], it is possible for a user to completely control presentations in the right frame without changing any of the existing GUI components.

XSL Presentations

The support for XSL is fairly straightforward. That is, with the exception that the SynEx Client cannot use, "as is", a reference to an XSL specification within the XML received from the server. Either an XSL specification, available on the local file system, must be explicitly assigned to a document with the "Set Presentation" command, or if no such assignment is made then the *"C:\SynExClient\Common\default_docview.xsl"* specification will be used. Of course, by changing the content of this file you can define a new default XSL specification to use as default.

Notice: An important deficiency in the current version is that document presentation assignments are not made persistent. Assignments made during a session will not be available when you start your next session!

DHTML Presentations

DHTML can be used as an alternative to XSL for document presentation. The advantage of DHTML over XSL is that DHTML is more explicit when it comes to organising the presentation layout.

DHTML presentations are assigned to documents the same way XSL specifications are assigned; i.e., by using the "Set Presentation" command and selecting a DHTML file. However, the creation of new DHTML files for presentation requires more intimate knowledge of the SynEx Client components than the XSL specifications (which require knowledge on the SynExML format). Thus before considering DHTML for document presentation, first consider the DHTML that is used for other kinds of information presentation.

Most information that is presented to a user in the right frame is made in DHTML; e.g. when presenting server information, doing record search, viewing the client execution trace in case of problems, etc. The DHTML files that are used for this are located under *"C:\SynExClient\DHTML"*; e.g. *"sxviewserver.html"*, *"sxrecordsearch.html"*, *"sxtrace.html"*, and so on. They all contain the ActiveX control *"VBInfoView.ocx"* which is responsible for inserting information from the cache into the HTML.

Figure 18 illustrates the (rather elaborate) communication between the components involved. Initially, when first starting the application, then the *GUIClient* component in the left HTML frame (see figure 14) gets hold of a reference to the window in the right HTML frame, and hands this over to the *DocBrowser* component (1). The purpose of *DocBrowser* is only let a single object be responsible for presentation issues. Later on, when e.g. the user requests information on a particular server by executing its "Properties" menu command, then *GUIClient* receives this request via its tree-view control. It then calls the *"prepareViewServer"* method of *DocBrowser* (2), after which *DocBrowser* navigates, in the *right* frame window, to *"C:\SynExClient\DHTML\sxviewserver.html"* (3). This DHTML file contains the *"VBInfoView.ocx"* control, and when the control is ready and loaded it gets hold of the *"VBSynExClient.ocx"* control in the left frame, and thus access to the *GUIClient* component within this control (*DocBrowser* is made a private component within *"VBSynExClient.ocx"*. *"VBSynExClient.ocx"* (i.e., its *ViewControl* component) then sends the message (method) *"presentationControlIsReady"* to *GUIClient*, and *GUIClient* just forwards this to *DocBrowser* which then invokes the *"ViewServer"* method of *ViewControl* within *"VBSynExClient.ocx"*, and provides it with the server information required. Finally the *"ViewServer"* method displays the information within the HTML of *"C:\SynExClient\DHTML\sxviewserver.html"*.

[1] or Java Applets - provided they are able to communicate with COM components.

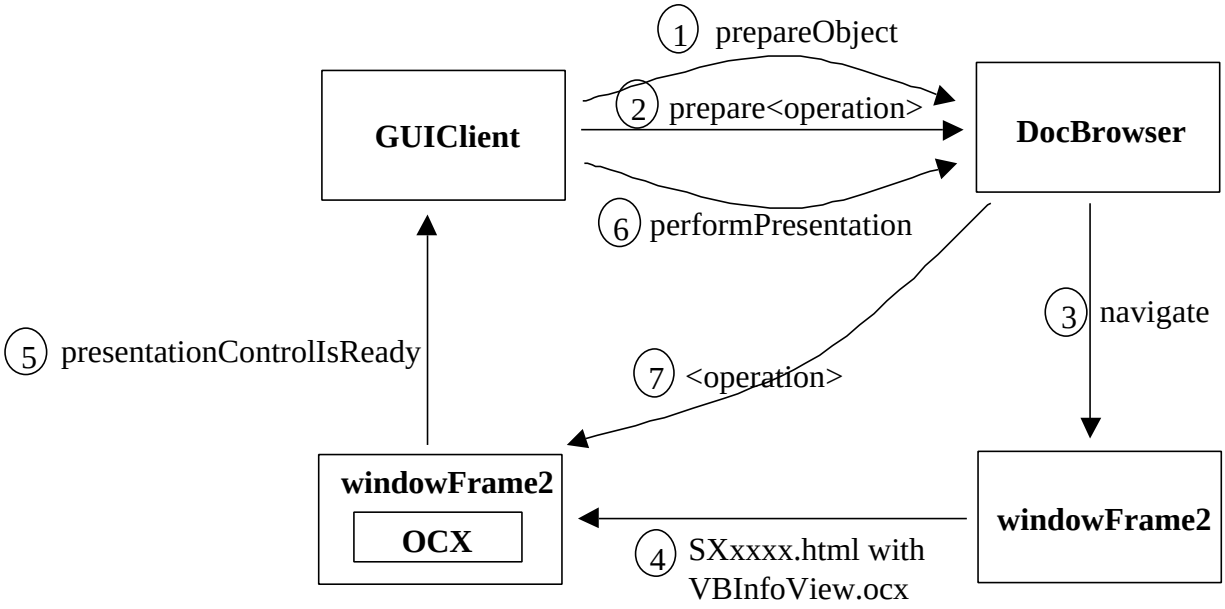


Figure 18. *Interaction sequence between the interaction control ("VBSynExClient.ocx") in the left frame, and the ActiveX control in charge of performing the document presentation in the DHTML page in the right frame.*

Indeed this may seem a little awkward, but the use of DHTML and its ActiveX control for server information, record requests, etc, was made for "experimentation" purposes when implementing this demonstrator. It would be much simpler to just use Visual Basic forms from the *"VBSynExClient.ocx"* control like e.g. those used for login information, offline catalog selection, etc.

Now back to document presentations. DHTML used for document presentation need not contain an ActiveX control. It may use an ActiveX control for its presentation logic, but it can also be made entirely in HTML and VBScript or JavaScript. An example of both is provided with the SynEx Client source code. In the *"C:\SynExClient\Samples"* catalog there is a DHTML file *"sxviewdoc-script.html"* with just HTML and VBScript, and also a similar file *"sxviewdoc-axctrl.html"* which uses the ActiveX control *"VBDocInfo.ocx"* for its presentation logic. Their differences are just that the VBScript code in the former is programmed within a COM component in the latter (in this case in Visual Basic, but any COM supporting language could have been used). You can see how these work by selecting them with the "Set Presentation" command.

When using DHTML for document presentations the component communication is not exactly as in figure 18. To illustrate this, see the listing of *"sxviewdoc-script.html"* below. The VBScript code, or the code in an ActiveX control for the same purpose, does not use the *"presentationControlIsReady"* method in figure 18. Instead it uses the *GUIClient* method *"DHTMLDocumentViewInformation"* to receive information on which document to display, and, not the least, a reference to the cache (*CacheManager*). Hence if you want to create your own DHTML document presentation, beside creating the HTML for this, all you need to do is change the VBScript code within the *"DisplayDocument"* routine below (and similarly in the code for *"VBDocInfo.ocx"*).

File: ***"C:\SynExClient\Samples\sxviewdoc-script.html"*** (an example of DHTML in VBScript)

```
<HTML>
<HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
    <TITLE>Example DHTML page, with only VBScript, for presenting a Synapses healthcare
           document (ComRIC)</TITLE>

    <SCRIPT LANGUAGE="VBScript">
      ' Variables and object references
      Dim refCacheManager
      Dim mServerID
      Dim mLoginID
      Dim mRecordID
      Dim mRCID
```

```
      ' Disable the context menu - event triggered by context selection
      Sub Document_OnContextMenu()
         Window.Event.ReturnValue = False
      End Sub

      ' Initialize variables and object references
      Sub Window_OnLoad()
         On Error Resume Next

         Dim otherWnd
         Dim refOtherControl

         ' Find the window of the other frame (the left frame)
         Set otherWnd = document.parentWindow.Top.frames("AXcontrolFrame")
         If (otherWnd Is Nothing) Then
            MsgBox "Unable to find the window in the left frame!", , "Error"
         Else
            Set refOtherControl = otherWnd.document.All("TVcontrol")
            If (refOtherControl Is Nothing) Then
               MsgBox "Unable to find ActiveX control in the left frame!", , "Error"
            Else
               Set refCacheManager = refOtherControl.DHTMLDocumentViewInformation(
                                                     mServerID, _
                                                     mLoginID, _
                                                     mRecordID, _
                                                     mRCID)
               If (refCacheManager Is Nothing) Then
                  MsgBox "Unable to retrieve the Cache Manager!", , "Error"
               Else
                  Call DisplayDocument()
               End If
            End If
         End If

         If Err.Number <> 0 Then
            MsgBox "Error: " & Err.Description, , "Error"
            Err.Clear
         End If
      End Sub

      ' Routine for displaying the document identification
      Sub DisplayDocument()
         document.getElementById("SXserverid").innerText = CStr(mServerID)
         document.getElementById("SXloginid").innerText = CStr(mLoginID)
         document.getElementById("SXrecordid").innerText = mRecordID
         document.getElementById("SXrcid").innerText = mRCID

         ' NOTICE: This example does not utilise the refCacheManager, but, in general,
         '         since this reference is available any cache information can be
         '         retrieved, from its IsxcmanCacheManager interface, and displayed
         '         in a DHTML file like this!
      End Sub

      ' Release object references
      Sub Window_OnUnload()
         Set refCacheManager = Nothing
      End Sub
   </SCRIPT>
</HEAD>

<BODY ID="SXdocument" TEXT="#000000" BGCOLOR="#ffff00">

    <B><FONT FACE="Comic Sans MS" SIZE=4>
       <P ALIGN="CENTER">DHTML page with VBScript for document presentation</P>
    </B></FONT>
```

```
    <P>This is an example of how you can create DHTML pages for customised document
       (ComRIC) presentation. This page only displays the identification of the document,
       i.e., its ServerID, LoginID, RecordID and RCID. However, since you have access to
       the cache, via the Cache Manager reference (IsxcmanCacheManager), you can retrieve
       document information and present it any way you like within your (D)HTML.</P>

    <P ALIGN="LEFT"><B>Document identification:</B></P>

    <TABLE CELLSPACING=0 BORDER=0 WIDTH=300 ALIGN="LEFT">
        <TR><TD WIDTH="50%" ALIGN="RIGHT"><P>ServerID:</TD>
            <TD ID="SXserverid" BGCOLOR="white" ALIGN="LEFT"> </TD></TR>

        <TR><TD WIDTH="50%" ALIGN="RIGHT"><P>LoginID:</TD>
            <TD ID="SXloginid" BGCOLOR="white" ALIGN="LEFT"> </TD></TR>

        <TR><TD WIDTH="50%" ALIGN="RIGHT"><P>RecordID:</TD>
            <TD ID="SXrecordid" BGCOLOR="white" ALIGN="LEFT"> </TD></TR>

        <TR><TD WIDTH="50%" ALIGN="RIGHT"><P>RCID:</TD>
            <TD ID="SXrcid" BGCOLOR="white" ALIGN="LEFT"> </TD></TR>
    </TABLE>

   <BR CLEAR="LEFT"/><BR/>

   <P>This page is made with VBScript only. In general, page functionality can be
      programmed in both VBScript or JavaScript, or alternatively you can include an
      ActiveX control or dll within the page to do the work. Visual Basic v.6 includes
      a DHTML application wizard that you may find useful.</P>
</BODY>
</HTML>
```

8. Security

Providing secure access to information is a key issue for healthcare information systems.

Authentication means that the server must be able to verify that the client is who he claims to be (e.g. via password like mechanisms, smart-cards, etc), and also that no other person can take over a client's session on the server without the server (and the client) being aware of this. The authentication mechanisms for the WP2 platform is described in the deliverable D2.2.

Authorisation and access control means that a client can only access or update information for which he is authorised to do such operations. Authorisation and access control should preferably be considered an inherent part of the overall system- and information modelling. This to make it possible to assign domain specific read and write authorisations to various levels of granularity, e.g. read access to a particular document, but write access only to a particular field in this document, and also to be flexible with respect to dynamically changing authorisations. Often there will be a trade-off between flexible access control versus performance. Thus authorisation and access control mechanisms should be taken into consideration right from the start of the analysis/design phase. The Oslo Synapses Server has already built-in such mechanisms that are very flexible and fine-grained; see deliverable D2.2.

Security relating to encryption of transferred information, client download of applets or ActiveX components, and more, are also important security topics, but there has not been an in-depth consideration of these issues within WP2 beyond what are common techniques for this.

9. Concluding Remarks

What XML Is and Isn't

XML plays a key role in our architecture for transmission of information server-to-client and client-to-server (and server-to-server). Thus to reconsider very briefly what XML is and isn't.

XML is a string of text formatted according to certain rules, where some of these rules are common to every XML string (well-formed), while others *may* be defined in an accompanying schema definition (valid). There are several different kinds of XML schema definitions, e.g. *DTD (Document Type Definition)*, *XML Schema*, and more. An XML string can easily be saved to a file (a plain text file), but XML is *not* meant for storing persistent information. That is, it is not principally different from how any text file can be used for that purpose and XML is no alternative to databases for information storage. On the contrary, text based information in XML, as opposed to other more specialised data formats, is very well-suited for database storage such that the XML itself can instead be created on demand. When using XML in an information system as we do then the XML strings may never exist in any files. More important is that when being transferred e.g. from the server to the client the XML is a text string, while when being created and when being received by the client it can be accessed and operated on as a data-structure (an object structure) with an interface of functions (including events) as any other e.g. COM object. That latter is possible via the *DOM (Document Object Model)* offered by XML parsers, as illustrated in figure 19.

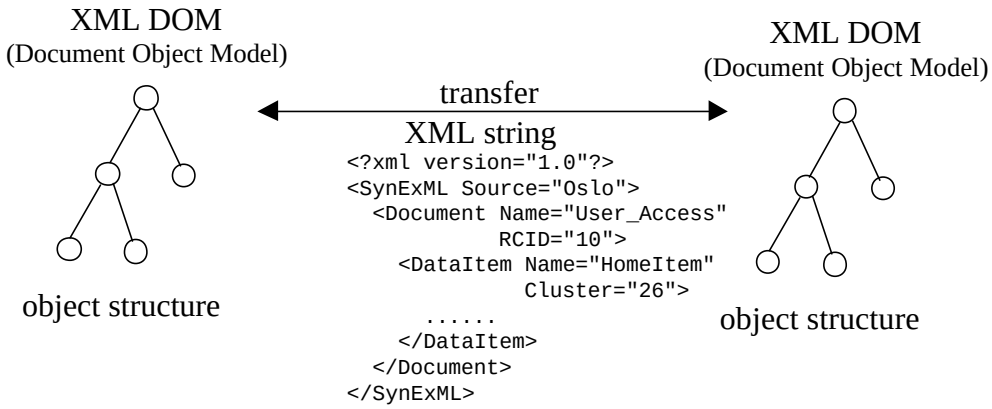


Figure 19. *XML as a string of text and also as a structure of objects.*

One lesson from SynEx, regarding what XML is not, is that despite that there are languages like DTD's and others for defining XML schemas, XML is ***not*** a modelling language and therefore ***should not be used as such***.

SynExML is based on the Synapses healthcare record specification, but due to the very generic nature of Synapses it has been a very time-consuming process for every site involved to agree on a common XML format. For each of the numerous changes made (and many more are expected and required if SynExML will live on) (at least) the XML parsing code had to be changed. We believe that these changes could have been avoided by basing the information and object modelling on a proper modelling method, and instead use XML at a "lower level" of the overall system development process.

UML (Unified Modelling Language) [13], and tools like e.g. Rational Rose [14], is becoming much of a standard for object-oriented and relational modelling, analysis and design. An important benefit of using UML is for communication purposes, both between software developers, but also between software developers and domain experts since it is relatively easy to enable domain experts to understand what is expressed by a UML model.

Healthcare information systems, and healthcare record systems in particular, are *open and generic* information systems in the sense that the kind of information that they must be able to manage is not fully known at design-time. They will evolve over time, and they must be designed for this. UML can be a useful tool not only for designing the information system itself, but also for making it more generic and more adaptable to changes and extensions over time. For example, in order for a client and a server to interact and communicate in a meaningful way they must have a common understanding regarding which requests can be

made by the client, and what kind of information will be returned from the server. Thus any XML that is transferred between clients and servers could be based on one out of two different kinds of XML formats, namely

- The schema of a UML model (its classes, structural and behavioural properties, etc)
- Information on object instances of this schema, and their relationships and properties

Client requests may then correspond to methods on various UML classes, and the information returned will be information on a particular set of objects instantiated from particular classes in the UML model. An important benefit of this is that regardless of which model changes and extensions are later made, e.g. in a healthcare record specification, the foundation for the common understanding necessary between a client and a server will remain the same, and the infrastructure for client-server communication will remain unchanged. Application developers will also be presented with object models and object interfaces that all adhere to certain commonly agreed UML conventions, and which only differ regarding which UML model they apply to. That is, different sub-systems and sub-models of an overall information system will be defined by different UML models to avoid a single, huge and unmanageable model.

Information Presentation

XML in combination with XSL makes it possible to provide many different kinds of presentations of the same XML formatted information. For example, the same XML string can be presented as WML in the WML browser of a mobile phone, as one kind of HTML in Netscape, as a slightly different HTML in Internet Explorer, as computer generated speech for a blind person, and so on.

However, particularly for very generic data structures like the Synapses healthcare record specification, and correspondingly SynExML formatted XML, then it is a very time-consuming and demanding task to produce high quality XSL specifications. Thus we believe that for healthcare record systems then XSL may well be used for certain ad-hoc, on-the-fly presentations, but most of the information presentation should be performed more "traditionally"; e.g. as with the Visual C++ presentation module in the current Oslo Synapses Server production version.

Record Distribution, Integration and Sharing

The work of WP2 has demonstrated that with state-of-the-art web technology it is relatively simple, technically, to achieve sharing and integration of distributed electronic patient records.

However, the content of an electronic patient record may be distributed globally, to any number of sites, and its content is also expected to be available "forever". The latter implies an important problem and challenge, also from a technical perspective, that has not been addressed within SynEx WP2. The remote record links in Synapses are "hard-coded" in the sense that they contain a particular webserver address, and in addition enough information to uniquely identity the relevant record parts on this server. However, over time webservers, databases, access rights are moved, renamed, changed, etc. Thus a more robust, long-life solution will be required to assure that distribution targets remain available.

10. References

1. SynEx Homepage, http://www.gesi.it/synex/
2. B.Jung, E.P.Andersen, J.Grimson; *Using XML for Seamless Integration of Distributed Electronic Patient Records*; the XML 2000 Scandinavia conference; http://www.xml2000.org/program/index.html
3. Synapses Homepage, http://www.cs.tcd.ie/synapses/public/
4. P.Hurlen, K.Skifjeld, E.P.Andersen, *The Basic Principles of the Synapses Federated Healthcare Record Server*, International Journal of Medical Informatics, Vol. 52, Nr. 1-3, 1998
5. W.Grimson, D.Berry, J.Grimson, G.Stephens, E.Felton, P.Given, R.O'Moore, *Federated Healthcare Record Server - the Synapses Paradigm*, International Journal of Medical Informatics, 1998
6. World Wide Web Consortium; *XML*; http://www.w3.org/XML
7. *SOAP specification*, http://msdn.microsoft.com/xml/general/SOAP_V09.asp
8. D.Box; *A Young Person's Guide to The Simple Object Access Protocol: SOAP Increases Interoperability Across Platforms and Languages*; http://msdn.microsoft.com/msdnmag/issues/0300/soap/soap.asp
9. Microsoft, *SQL Server*, http://www.microsoft.com/sql
10. Microsoft, *COM/DCOM*, http://www.microsoft.com/com
11. Microsoft, *UDA/OLE DB/ADO*, http://www.microsoft.com/data
12. Microsoft, *IIS/ASP*, http://www.microsoft.com/iis
13. Object Management Group (OMG), *UML*, http://www.omg.org/uml
14. Rational Rose, *UML Resource Center*, http://www.rational.com/uml
15. Object Management Group (OMG), *CORBA*, http://www.omg.org/corba

A. IDL (Interface Definition Language) Specification for Client Components

SynExClient.IDL

Source: */SynExClientIDL/SynExClient.idl*

```
// SynExClient.idl : IDL source for SynExClient.dll
//
// This file will be processed by the MIDL tool to
// produce the type library (SynExClient.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";

[
    uuid(EFF435BC-19C1-11D4-9639-0060979B4844),
    version(1.0),
    helpstring("SynExClient 1.0 Type Library")
]
library SYNEXCLIENTLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    // TLib : Microsoft XML, version 2.0 : {D63E0CE2-A0A2-11D0-9C02-00C04FC99C8E}
    importlib("msxml.dll");

    // Forward declarations
    interface IsxprvFHCRLogin;
    interface IsxprvFHCRInformation;
    interface IsxprvInformation;
    interface IsxxmlParser;
    interface IsxxmlFHCRParser;
    interface IsxcmanCacheManager;
    interface IsxguiLoginInfo;
    interface IsxcmanEventSubscription;
    interface IsxoutEvents;
    interface IsxcshCreateCacheObjects;
    interface IsxcshCacheSearch;
    interface IsxcshSynExInfo;
    interface IsxcshServer;
    interface IsxcshLogin;
    interface IsxcshRICShape;
    interface IsxcshRICInformation;
    interface IsxcshRICOperation;
    interface IsxcshRecordFolder;
    interface IsxcshFolderRIC;
    interface IsxcshComRIC;
    interface IsxcshViewRIC2;
    interface IsxcshDataRIC;
    interface IsxcshViewRIC1;
    interface IsxcshRIShape;
    interface IsxcshRIInformation;
    interface IsxcshRIOperation;
    interface IsxcshRecordItem;
    interface IsxcshCollection;


    //------------ IsxprvFHCRLogin

    [
        object,
        uuid(CFBB8611-19C0-11d4-9639-0060979B4844),
        oleautomation,
        dual,
```

```
    helpstring("IsxprvFHCRLogin interface"),
    pointer_default(unique)
]
interface IsxprvFHCRLogin : IDispatch
{
    [id(1), helpstring("Function LogOn")]
    HRESULT LogOn([in] BSTR inAddress,
                  [in] BSTR inUserName,
                  [in] BSTR inPassword,
                  [out,retval] VARIANT_BOOL* bResult);

    [id(2), helpstring("Function LogOff")]
    HRESULT LogOff([in] BSTR inAddress,
                   [in] BSTR inUserName,
                   [out,retval] VARIANT_BOOL* bResult);
};


//------------ IsxprvFHCRInformation

[
    object,
    uuid(CFBB8612-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxprvFHCRInformation interface"),
    pointer_default(unique)
]
interface IsxprvFHCRInformation : IDispatch
{
    [id(1), helpstring("Function SetContextInfo")]
    HRESULT SetContextInfo([in] IsxcshCreateCacheObjects* inRefCache,
                           [in] long inLoginID,
                           [in] long inServerID);

    [id(2), helpstring("Function CacheRecordInfo")]
    HRESULT CacheRecordInfo([in] BSTR inAddress,
                            [in] BSTR inUserName,
                            [in] BSTR inRecordID,
                            [in] BSTR inRetrieval,
                            [in, out] VARIANT_BOOL* bLoginOK,
                            [out,retval] VARIANT_BOOL* bResult);

    [id(3), helpstring("Function CacheFolderInfo")]
    HRESULT CacheFolderInfo([in] BSTR inAddress,
                            [in] BSTR inUserName,
                            [in] BSTR inRecordID,
                            [in] BSTR inRCID,
                            [in] BSTR inRetrieval,
                            [in, out] VARIANT_BOOL* bLoginOK,
                            [out,retval] VARIANT_BOOL* bResult);

    [id(4), helpstring("Function CacheDocumentInfo")]
    HRESULT CacheDocumentInfo([in] BSTR inAddress,
                              [in] BSTR inUserName,
                              [in] BSTR inRecordID,
                              [in] BSTR inRCID,
                              [in] BSTR inRetrieval,
                              [in, out] VARIANT_BOOL* bLoginOK,
                              [out,retval] VARIANT_BOOL* bResult);

    [id(5), helpstring("Function GetRecordInfo")]
    HRESULT GetRecordInfo([in] BSTR inAddress,
                          [in] BSTR inUserName,
                          [in] BSTR inRecordID,
                          [in] BSTR inRetrieval,
```

```
                            [in] BSTR inResponse,
                            [in, out] BSTR* outResponse,
                            [in, out] VARIANT_BOOL* bLoginOK,
                            [out,retval] VARIANT_BOOL* bResult);

        [id(6), helpstring("Function GetFolderInfo")]
        HRESULT GetFolderInfo([in] BSTR inAddress,
                            [in] BSTR inUserName,
                            [in] BSTR inRecordID,
                            [in] BSTR inRCID,
                            [in] BSTR inRetrieval,
                            [in] BSTR inResponse,
                            [in, out] BSTR* outResponse,
                            [in, out] VARIANT_BOOL* bLoginOK,
                            [out,retval] VARIANT_BOOL* bResult);

        [id(7), helpstring("Function GetDocumentInfo")]
        HRESULT GetDocumentInfo([in] BSTR inAddress,
                              [in] BSTR inUserName,
                              [in] BSTR inRecordID,
                              [in] BSTR inRCID,
                              [in] BSTR inRetrieval,
                              [in] BSTR inResponse,
                              [in, out] BSTR* outResponse,
                              [in, out] VARIANT_BOOL* bLoginOK,
                              [out,retval] VARIANT_BOOL* bResult);
    };


    //------------ IsxprvInformation

    [
        object,
        uuid(CFBB8613-19C0-11d4-9639-0060979B4844),
        oleautomation,
        dual,
        helpstring("IsxprvInformation interface"),
        pointer_default(unique)
    ]
    interface IsxprvInformation : IDispatch
    {
        [id(1), helpstring("Function SetContextInfo")]
        HRESULT SetContextInfo([in] IsxcshCreateCacheObjects* inRefCache);

        [id(2), helpstring("Function CacheInformation")]
        HRESULT CacheInformation([out,retval] VARIANT_BOOL* bResult);
    };


    //------------ IsxxmlParser

    [
        object,
        uuid(CFBB8633-19C0-11d4-9639-0060979B4844),
        oleautomation,
        dual,
        helpstring("IsxxmlParser interface"),
        pointer_default(unique)
    ]
    interface IsxxmlParser : IDispatch
    {
        [id(1), helpstring("Function ParseAndCache")]
        HRESULT ParseAndCache([in] IXMLDOMDocument* inRefXMLDoc,
                            [out,retval] VARIANT_BOOL* bResult);

        [id(2), helpstring("Function SetContextInfo")]
```

```
    HRESULT SetContextInfo([in] IsxcshCreateCacheObjects* inRefCache);
};


//------------ IsxxmlFHCRParser

[
    object,
    uuid(026962B1-1B71-11d4-963B-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxxmlFHCRParser interface"),
    pointer_default(unique)
]
interface IsxxmlFHCRParser : IDispatch
{
    [id(1), helpstring("Function ParseAndCache")]
    HRESULT ParseAndCache([in] IXMLDOMDocument* inRefXMLDoc,
                          [out,retval] VARIANT_BOOL* bResult);

    [id(2), helpstring("Function SetContextInfo")]
    HRESULT SetContextInfo([in] IsxcshCreateCacheObjects* inRefCache,
                           [in] long inLoginID,
                           [in] BSTR inServerType,
                           [in] long inServerID);
};


//------------ IsxcmanCacheManager

[
    object,
    uuid(CFBB8634-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcmanCacheManager interface"),
    pointer_default(unique)
]
interface IsxcmanCacheManager : IDispatch
{
    [propget, id(1), helpstring("property OffLineCatalog")]
    HRESULT OffLineCatalog([out, retval] BSTR *pOffLineCatalog);

    [propput, id(1), helpstring("property OffLineCatalog")]
    HRESULT OffLineCatalog([in] BSTR newOffLineCatalog);

    [propget, id(2), helpstring("property DefaultServer")]
    HRESULT DefaultServer([out, retval] BSTR *pDefaultServer);

    [propput, id(2), helpstring("property DefaultServer")]
    HRESULT DefaultServer([in] BSTR newDefaultServer);

    [id(3), helpstring("Function GetCacheCreation")]
    HRESULT GetCacheCreation([out,retval] IsxcshCreateCacheObjects** refCache);

    [id(4), helpstring("Function GetCacheSearch")]
    HRESULT GetCacheSearch([out,retval] IsxcshCacheSearch** refCache);

    [id(5), helpstring("Function Exit")]
    HRESULT Exit();

    [id(6), helpstring("Function WorkOffline")]
    HRESULT WorkOffline();

    [id(7), helpstring("Function WorkOnline")]
```

```
    HRESULT WorkOnline();

    [id(8), helpstring("Function IsOnline")]
    HRESULT IsOnline([out,retval] VARIANT_BOOL* bResult);

    [id(9), helpstring("Function SaveCache")]
    HRESULT SaveCache([in] BSTR inCatalog,
                      [out,retval] VARIANT_BOOL* bResult);

    [id(10), helpstring("Function LoadCache")]
    HRESULT LoadCache([in] BSTR inCatalog,
                      [out,retval] VARIANT_BOOL* bResult);

    [id(11), helpstring("Function LoadServerInfo")]
    HRESULT LoadServerInfo([out,retval] VARIANT_BOOL* bResult);

    [id(12), helpstring("Function LoadSynExInfo")]
    HRESULT LoadSynExInfo([out,retval] VARIANT_BOOL* bResult);

    [id(13), helpstring("Function RegisterLoginGUI")]
    HRESULT RegisterLoginGUI([in] IsxguiLoginInfo* inRefLoginInfo);
};


//------------ IsxguiLoginInfo

[
    object,
    uuid(CFBB8635-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxguiLoginInfo interface"),
    pointer_default(unique)
]
interface IsxguiLoginInfo : IDispatch
{
    [id(1), helpstring("Function GetLoginInformation")]
    HRESULT GetLoginInformation([in] BSTR inServerName,
                                [in, out] BSTR* inoutUserName,
                                [in, out] BSTR* outPassword,
                                [out,retval] VARIANT_BOOL* bResult);
};


//------------ IsxcmanEventSubscription

[
    object,
    uuid(CFBB8616-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcmanEventSubscription interface"),
    pointer_default(unique)
]
interface IsxcmanEventSubscription : IDispatch
{
    [id(1), helpstring("Function Subscribe")]
    HRESULT Subscribe([in] IsxoutEvents* inRefReceiver,
                      [in, out] long* outSubscrID,
                      [out,retval] VARIANT_BOOL* bResult);

    [id(2), helpstring("Function EndSubscribe")]
    HRESULT EndSubscribe([in] long inSubscrID,
                         [out,retval] VARIANT_BOOL* bResult);
};
```

```
//------------ IsxoutEvents

[
    object,
    uuid(CFBB8617-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxoutEvents interface"),
    pointer_default(unique)
]
interface IsxoutEvents : IDispatch
{
    [id(1), helpstring("Function Notification")]
    HRESULT Notification([in] BSTR inMessage);

    [id(2), helpstring("Function SynExInfoAdded")]
    HRESULT SynExInfoAdded([in] IsxcshSynExInfo* inRefSynExInfo);

    [id(3), helpstring("Function SynExInfoRemoved")]
    HRESULT SynExInfoRemoved([in] BSTR inSynExInfoName);

    [id(4), helpstring("Function ServerAdded")]
    HRESULT ServerAdded([in] IsxcshServer* inRefServer);

    [id(5), helpstring("Function ServerRemoved")]
    HRESULT ServerRemoved([in] long inServerID,
                          [in] BSTR inServerName);

    [id(6), helpstring("Function LoginAdded")]
    HRESULT LoginAdded([in] IsxcshLogin* inRefLogin);

    [id(7), helpstring("Function LoginRemoved")]
    HRESULT LoginRemoved([in] long inLoginID,
                         [in] BSTR inUserName,
                         [in] long inServerID,
                         [in] BSTR inServerName);

    [id(8), helpstring("Function RICAdded")]
    HRESULT RICAdded([in] IsxcshRICShape* inRefRICShape);

    [id(9), helpstring("Function RICRefreshed")]
    HRESULT RICRefreshed([in] IsxcshRICShape* inRefRICShape);

    [id(10), helpstring("Function RICRemoved")]
    HRESULT RICRemoved([in] long inServerID,
                       [in] long inLoginID,
                       [in] BSTR inRecordID,
                       [in] BSTR inRCID,
                       [in] BSTR inUserName,
                       [in] BSTR inServerName);
};


//------------ IsxcshCreateCacheObjects

[
    object,
    uuid(CFBB8618-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshCreateCacheObjects interface"),
    pointer_default(unique)
]
```

```
interface IsxcshCreateCacheObjects : IDispatch
{
    [id(1), helpstring("Function InsertSynExInfo")]
    HRESULT InsertSynExInfo([in] BSTR inName,
                            [in] BSTR inAddress,
                             [out,retval] VARIANT_BOOL* bResult);

    [id(2), helpstring("Function InsertServer")]
    HRESULT InsertServer([in] BSTR inName,
                         [in] BSTR inType,
                         [in] BSTR inAddress,
                         [in] BSTR inWebInfo,
                         [in] BSTR inOffLineCatalog,
                         [out,retval] VARIANT_BOOL* bResult);

    [id(3), helpstring("Function InsertLogin")]
    HRESULT InsertLogin([in] BSTR inUserName,
                        [in] long inServerID,
                        [in, out] long* outLoginID,
                        [out,retval] VARIANT_BOOL* bResult);

    [id(4), helpstring("Function PostRecordCaching")]
    HRESULT PostRecordCaching([in] BSTR inRecordID,
                              [in] BSTR inRCID,
                              [in] long inLoginID,
                              [in] long inServerID,
                              [in] BSTR inAction,
                              [out,retval] VARIANT_BOOL* bResult);

    [id(5), helpstring("Function InsertRICShape")]
    HRESULT InsertRICShape([in] long inServerID,
                           [in] long inLoginID,
                           [in] BSTR inRecordID,
                           [in] BSTR inRCID,
                           [in] BSTR inSynapsesType,
                           [in] BSTR inRICType,
                           [in] BSTR inParentRCID,
                           [in] BSTR inFirstChildRCID,
                           [in] BSTR inPredRCID,
                           [in] BSTR inTargetServerAddress,
                           [in] BSTR inTargetRecordID,
                           [in] BSTR inTargetRCID,
                           [out,retval] VARIANT_BOOL* bResult);

    [id(6), helpstring("Function InsertRecordFolder")]
    HRESULT InsertRecordFolder([in] long inServerID,
                               [in] long inLoginID,
                               [in] BSTR inRecordID,
                               [in] BSTR inRCID,
                               [in] BSTR inClassName,
                               [in] BSTR inType,
                               [in] BSTR inLanguage,
                               [in] BSTR inLogTime,
                               [in] BSTR inLogUserID,
                               [in] BSTR inInvalidationTime,
                               [in] BSTR inInvalidationUserID,
                               [out,retval] VARIANT_BOOL* bResult);

    [id(7), helpstring("Function InsertFolderRIC")]
    HRESULT InsertFolderRIC([in] long inServerID,
                            [in] long inLoginID,
                            [in] BSTR inRecordID,
                            [in] BSTR inRCID,
                            [in] BSTR inClassName,
                            [in] BSTR inType,
                            [in] BSTR inLanguage,
```

```
                        [in] BSTR inLogTime,
                        [in] BSTR inLogUserID,
                        [in] BSTR inInvalidationTime,
                        [in] BSTR inInvalidationUserID,
                        [out,retval] VARIANT_BOOL* bResult);

    [id(8), helpstring("Function InsertComRIC")]
    HRESULT InsertComRIC([in] long inServerID,
                         [in] long inLoginID,
                         [in] BSTR inRecordID,
                         [in] BSTR inRCID,
                         [in] BSTR inClassName,
                         [in] BSTR inType,
                         [in] BSTR inLanguage,
                         [in] BSTR inLogTime,
                         [in] BSTR inLogUserID,
                         [in] BSTR inInvalidationTime,
                         [in] BSTR inInvalidationUserID,
                         [out,retval] VARIANT_BOOL* bResult);

    [id(9), helpstring("Function InsertViewRIC2")]
    HRESULT InsertViewRIC2([in] long inServerID,
                           [in] long inLoginID,
                           [in] BSTR inRecordID,
                           [in] BSTR inRCID,
                           [in] BSTR inClassName,
                           [in] BSTR inType,
                           [in] BSTR inLanguage,
                           [in] BSTR inLogTime,
                           [in] BSTR inLogUserID,
                           [in] BSTR inInvalidationTime,
                           [in] BSTR inInvalidationUserID,
                           [out,retval] VARIANT_BOOL* bResult);

    [id(10), helpstring("Function InsertDataRIC")]
    HRESULT InsertDataRIC([in] long inServerID,
                          [in] long inLoginID,
                          [in] BSTR inRecordID,
                          [in] BSTR inRCID,
                          [in] BSTR inClassName,
                          [in] BSTR inType,
                          [in] BSTR inLanguage,
                          [in] BSTR inLogTime,
                          [in] BSTR inLogUserID,
                          [in] BSTR inInvalidationTime,
                          [in] BSTR inInvalidationUserID,
                          [out,retval] VARIANT_BOOL* bResult);

    [id(11), helpstring("Function InsertViewRIC1")]
    HRESULT InsertViewRIC1([in] long inServerID,
                           [in] long inLoginID,
                           [in] BSTR inRecordID,
                           [in] BSTR inRCID,
                           [in] BSTR inClassName,
                           [in] BSTR inType,
                           [in] BSTR inLanguage,
                           [in] BSTR inLogTime,
                           [in] BSTR inLogUserID,
                           [in] BSTR inInvalidationTime,
                           [in] BSTR inInvalidationUserID,
                           [out,retval] VARIANT_BOOL* bResult);

    [id(12), helpstring("Function InsertRIShape")]
    HRESULT InsertRIShape([in] long inServerID,
                          [in] long inLoginID,
                          [in] BSTR inRecordID,
```

```
                            [in] BSTR inRCID,
                            [in] BSTR inSynapsesType,
                            [in] BSTR inRIType,
                            [in] BSTR inParent,
                            [in] BSTR inFirstChild,
                            [in] BSTR inPred,
                            [in] BSTR inHomeRIC,
                            [out,retval] VARIANT_BOOL* bResult);

    [id(13), helpstring("Function InsertRecordItem")]
    HRESULT InsertRecordItem([in] long inServerID,
                             [in] long inLoginID,
                             [in] BSTR inRecordID,
                             [in] BSTR inRCID,
                             [in] BSTR inClassName,
                             [in] BSTR inValue,
                             [in] BSTR inType,
                             [in] BSTR inLanguage,
                             [in] BSTR inDataType,
                             [in] BSTR inLogTime,
                             [in] BSTR inLogUserID,
                             [in] BSTR inInvalidationTime,
                             [in] BSTR inInvalidationUserID,
                             [in] BSTR inEventBeginTime,
                             [in] BSTR inEventEndTime,
                             [in] BSTR inCluster,
                             [in] BSTR inInternalDataType,
                             [in] BSTR inFormat,
                             [out,retval] VARIANT_BOOL* bResult);

    [id(14), helpstring("Function GenerateNewRecordID")]
    HRESULT GenerateNewRecordID([out,retval] BSTR* outRecordID);
};


//------------ IsxcshCacheSearch

[
    object,
    uuid(CFBB8615-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshCacheSearch interface"),
    pointer_default(unique)
]
interface IsxcshCacheSearch : IDispatch
{
    [id(1), helpstring("Function GetCacheManager")]
    HRESULT GetCacheManager([out,retval] IsxcmanCacheManager** refCacheManager);

    [id(2), helpstring("Function FindAllSynExInfos")]
    HRESULT FindAllSynExInfos([out,retval] IsxcshCollection** outRefSynExInfos);

    [id(3), helpstring("Function FindAllServers")]
    HRESULT FindAllServers([out,retval] IsxcshCollection** outRefServers);

    [id(4), helpstring("Function FindAllLogins")]
    HRESULT FindAllLogins([out,retval] IsxcshCollection** outRefLogins);

    [id(5), helpstring("Function FindServer")]
    HRESULT FindServer([in] long inServerID,
                       [out,retval] IsxcshServer** outRefServer);

    [id(6), helpstring("Function FindLogin")]
    HRESULT FindLogin([in] long inLoginID,
```

```
                        [out,retval] IsxcshLogin** outRefLogin);

    [id(7), helpstring("Function FindSynExInfo")]
    HRESULT FindSynExInfo([in] BSTR inInfoName,
                          [out,retval] IsxcshSynExInfo** outRefSynExInfo);

    [id(8), helpstring("Function FindRICShape")]
    HRESULT FindRICShape([in] long inServerID,
                         [in] long inLoginID,
                         [in] BSTR inRecordID,
                         [in] BSTR inRCID,
                         [out,retval] IsxcshRICShape** outRefRICShape);

    [id(9), helpstring("Function FindRecord")]
    HRESULT FindRecord([in] long inServerID,
                       [in] long inLoginID,
                       [in] BSTR inRecordID,
                       [out,retval] IsxcshRecordFolder** outRefRecordFolder);

    [id(10), helpstring("Function FindFolder")]
    HRESULT FindFolder([in] long inServerID,
                       [in] long inLoginID,
                       [in] BSTR inRecordID,
                       [in] BSTR inRCID,
                       [out,retval] IsxcshFolderRIC** outRefFolderRIC);

    [id(11), helpstring("Function FindDocument")]
    HRESULT FindDocument([in] long inServerID,
                         [in] long inLoginID,
                         [in] BSTR inRecordID,
                         [in] BSTR inRCID,
                         [out,retval] IsxcshComRIC** outRefComRIC);

    [id(12), helpstring("Function FindRIShape")]
    HRESULT FindRIShape([in] long inServerID,
                        [in] long inLoginID,
                        [in] BSTR inRecordID,
                        [in] BSTR inRCID,
                        [out,retval] IsxcshRIShape** outRefRIShape);

    [id(13), helpstring("Function FindRecordItem")]
    HRESULT FindRecordItem([in] long inServerID,
                           [in] long inLoginID,
                           [in] BSTR inRecordID,
                           [in] BSTR inRCID,
                           [out,retval] IsxcshRecordItem** outRefRecordItem);
};


//------------ IsxcshSynExInfo

[
    object,
    uuid(CFBB8619-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshSynExInfo interface"),
    pointer_default(unique)
]
interface IsxcshSynExInfo : IDispatch
{
    [propget, id(1), helpstring("property Name")]
    HRESULT Name([out, retval] BSTR *pName);

    [propget, id(2), helpstring("property Address")]
```

```
    HRESULT Address([out, retval] BSTR *pAddress);

    [propput, id(2), helpstring("property Address")]
    HRESULT Address([in] BSTR newAddress);
};


//------------ IsxcshServer

[
    object,
    uuid(CFBB861B-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshServer interface"),
    pointer_default(unique)
]
interface IsxcshServer : IDispatch
{
    [propget, id(1), helpstring("property ServerID")]
    HRESULT ServerID([out, retval] long *pServerID);

    [propget, id(2), helpstring("property Name")]
    HRESULT Name([out, retval] BSTR *pName);

    [propget, id(3), helpstring("property Type")]
    HRESULT Type([out, retval] BSTR *pType);

    [propget, id(4), helpstring("property Address")]
    HRESULT Address([out, retval] BSTR *pAddress);

    [propput, id(4), helpstring("property Address")]
    HRESULT Address([in] BSTR newAddress);

    [propget, id(5), helpstring("property WebInfo")]
    HRESULT WebInfo([out, retval] BSTR *pWebInfo);

    [propput, id(5), helpstring("property WebInfo")]
    HRESULT WebInfo([in] BSTR newWebInfo);

    [propget, id(6), helpstring("property OffLineCatalog")]
    HRESULT OffLineCatalog([out, retval] BSTR *pOffLineCatalog);

    [propput, id(6), helpstring("property OffLineCatalog")]
    HRESULT OffLineCatalog([in] BSTR newOffLineCatalog);

    [id(7), helpstring("Function FindLogin")]
    HRESULT FindLogin([in] BSTR inUserName,
                      [out,retval] IsxcshLogin** outRefLogin);

    [id(8), helpstring("Function Logins")]
    HRESULT Logins([out,retval] IsxcshCollection** outRefLogins);

    [id(9), helpstring("Function LogOn")]
    HRESULT LogOn([in] BSTR inUserName,
                  [in] BSTR inPassword,
                  [in, out] long* outLoginID,
                  [out,retval] VARIANT_BOOL* bResult);

    [id(10), helpstring("Function Delete")]
    HRESULT Delete([out,retval] VARIANT_BOOL* bResult);
};


//------------ IsxcshLogin
```

```
[
    object,
    uuid(CFBB861C-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshLogin interface"),
    pointer_default(unique)
]
interface IsxcshLogin : IDispatch
{
    [id(1), helpstring("Function LoginID")]
    HRESULT LoginID([out,retval] long* outLoginID);

    [id(2), helpstring("Function UserName")]
    HRESULT UserName([out,retval] BSTR* outUserName);

    [id(3), helpstring("Function Server")]
    HRESULT Server([out,retval] IsxcshServer** outRefServer);

    [id(4), helpstring("Function RootRICs")]
    HRESULT RootRICs([out,retval] IsxcshCollection** outRefRICShapes);

    [id(5), helpstring("Function Refresh")]
    HRESULT Refresh([in] BSTR inPassword,
                    [out,retval] VARIANT_BOOL* bResult);

    [id(6), helpstring("Function LogOff")]
    HRESULT LogOff([out,retval] VARIANT_BOOL* bResult);

    [id(7), helpstring("Function CacheRecordInfo")]
    HRESULT CacheRecordInfo([in] BSTR inRecordID,
                            [in] BSTR inRetrieval,
                            [in, out] VARIANT_BOOL* bLoginOK,
                            [out,retval] VARIANT_BOOL* bResult);

    [id(8), helpstring("Function CacheFolderInfo")]
    HRESULT CacheFolderInfo([in] BSTR inRecordID,
                            [in] BSTR inRCID,
                            [in] BSTR inRetrieval,
                            [in, out] VARIANT_BOOL* bLoginOK,
                            [out,retval] VARIANT_BOOL* bResult);

    [id(9), helpstring("Function CacheDocumentInfo")]
    HRESULT CacheDocumentInfo([in] BSTR inRecordID,
                              [in] BSTR inRCID,
                              [in] BSTR inRetrieval,
                              [in, out] VARIANT_BOOL* bLoginOK,
                              [out,retval] VARIANT_BOOL* bResult);

    [id(10), helpstring("Function GetRecordInfo")]
    HRESULT GetRecordInfo([in] BSTR inRecordID,
                          [in] BSTR inRetrieval,
                          [in] BSTR inResponse,
                          [in, out] BSTR* outResponse,
                          [in, out] VARIANT_BOOL* bLoginOK,
                          [out,retval] VARIANT_BOOL* bResult);

    [id(11), helpstring("Function GetFolderInfo")]
    HRESULT GetFolderInfo([in] BSTR inRecordID,
                          [in] BSTR inRCID,
                          [in] BSTR inRetrieval,
                          [in] BSTR inResponse,
                          [in, out] BSTR* outResponse,
                          [in, out] VARIANT_BOOL* bLoginOK,
```

```
                            [out,retval] VARIANT_BOOL* bResult);

        [id(12), helpstring("Function GetDocumentInfo")]
        HRESULT GetDocumentInfo([in] BSTR inRecordID,
                                [in] BSTR inRCID,
                                [in] BSTR inRetrieval,
                                [in] BSTR inResponse,
                                [in, out] BSTR* outResponse,
                                [in, out] VARIANT_BOOL* bLoginOK,
                                [out,retval] VARIANT_BOOL* bResult);

        [id(13), helpstring("Function TransformXMLwithXSL")]
        HRESULT TransformXMLwithXSL([in] BSTR inXMLString,
                                    [in] BSTR inXSLAddress,
                                    [in, out] BSTR* outResultString,
                                    [out,retval] VARIANT_BOOL* bResult);
    };


    //------------ IsxcshRICShape

    [
        object,
        uuid(CFBB861D-19C0-11d4-9639-0060979B4844),
        oleautomation,
        dual,
        helpstring("IsxcshRICShape interface"),
        pointer_default(unique)
    ]
    interface IsxcshRICShape : IDispatch
    {
        [id(1), helpstring("Function ServerID")]
        HRESULT ServerID([out,retval] long* outServerID);

        [id(2), helpstring("Function LoginID")]
        HRESULT LoginID([out,retval] long* outLoginID);

        [id(3), helpstring("Function RecordID")]
        HRESULT RecordID([out,retval] BSTR* outRecordID);

        [id(4), helpstring("Function RCID")]
        HRESULT RCID([out,retval] BSTR* outRCID);

        [id(5), helpstring("Function SynapsesType")]
        HRESULT SynapsesType([out,retval] BSTR* outSynapsesType);

        [id(6), helpstring("Function RICType")]
        HRESULT RICType([out,retval] BSTR* outRICType);

        [id(7), helpstring("Function Login")]
        HRESULT Login([out,retval] IsxcshLogin** outRefLogin);

        [id(8), helpstring("Function Server")]
        HRESULT Server([out,retval] IsxcshServer** outRefServer);

        [id(9), helpstring("Function Parent")]
        HRESULT GoParent([out,retval] VARIANT_BOOL* bResult);

        [id(10), helpstring("Function FirstChild")]
        HRESULT GoFirstChild([out,retval] VARIANT_BOOL* bResult);

        [id(11), helpstring("Function Succ")]
        HRESULT GoSucc([out,retval] VARIANT_BOOL* bResult);

        [id(12), helpstring("Function Pred")]
```

```
    HRESULT GoPred([out,retval] VARIANT_BOOL* bResult);

    [id(13), helpstring("Function RecordItems")]
    HRESULT RecordItems([out,retval] IsxcshCollection** outRefRecordItems);

    [id(14), helpstring("Function Sources")]
    HRESULT Sources([out,retval] IsxcshCollection** outRefSources);

    [id(15), helpstring("Function Target")]
    HRESULT Target([out,retval] IsxcshRICShape** outRefTarget);

    [id(16), helpstring("Function Clone")]
    HRESULT Clone([out,retval] IsxcshRICShape** outRICShape);
};


//------------ IsxcshRICInformation

[
    object,
    uuid(CFBB861E-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshRICInformation interface"),
    pointer_default(unique)
]
interface IsxcshRICInformation : IDispatch
{
    [id(1), helpstring("Function ServerID")]
    HRESULT ServerID([out,retval] long* outServerID);

    [id(2), helpstring("Function LoginID")]
    HRESULT LoginID([out,retval] long* outLoginID);

    [id(3), helpstring("Function RecordID")]
    HRESULT RecordID([out,retval] BSTR* outRecordID);

    [id(4), helpstring("Function RCID")]
    HRESULT RCID([out,retval] BSTR* outRCID);

    [id(5), helpstring("Function ClassName")]
    HRESULT ClassName([out,retval] BSTR* outClassName);

    [id(6), helpstring("Function Type")]
    HRESULT Type([out,retval] BSTR* outType);

    [id(7), helpstring("Function Language")]
    HRESULT Language([out,retval] BSTR* outLanguage);

    [id(8), helpstring("Function LogTime")]
    HRESULT LogTime([out,retval] BSTR* outLogTime);

    [id(9), helpstring("Function LogUserID")]
    HRESULT LogUserID([out,retval] BSTR* outLogUserID);

    [id(10), helpstring("Function InvalidationTime")]
    HRESULT InvalidationTime([out,retval] BSTR* outInvalidationTime);

    [id(11), helpstring("Function InvalidationUserID")]
    HRESULT InvalidationUserID([out,retval] BSTR* outInvalidationUserID);
};


//------------ IsxcshRICOperation
```

```
[
    object,
    uuid(CFBB8621-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshRICOperation interface"),
    pointer_default(unique)
]
interface IsxcshRICOperation : IDispatch  // preferably - : IsxcshRICInformation
{
    [id(13), helpstring("Function Save")]
    HRESULT Save([in] BSTR inRetrieval,
                 [in] BSTR inResponse,
                 [in, out] VARIANT_BOOL* bLoginOK,
                 [out,retval] VARIANT_BOOL* bResult);

    [id(14), helpstring("Function Refresh")]
    HRESULT Refresh([in, out] VARIANT_BOOL* bLoginOK,
                    [out,retval] VARIANT_BOOL* bResult);

    [id(15), helpstring("Function Delete")]
    HRESULT Delete([out,retval] VARIANT_BOOL* bResult);
};


//------------ IsxcshRecordFolder

[
    object,
    uuid(CFBB861F-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshRecordFolder interface"),
    pointer_default(unique)
]
interface IsxcshRecordFolder : IDispatch  // preferably - : IsxcshRICOperation
{
    [id(16), helpstring("Function GetPresentation")]
    HRESULT GetPresentation([out,retval] BSTR* outFileName);

    [id(17), helpstring("Function SetPresentation")]
    HRESULT SetPresentation([in] BSTR inFileName,
                            [out,retval] VARIANT_BOOL* bResult);
};


//------------ IsxcshFolderRIC

[
    object,
    uuid(CFBB8620-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshFolderRIC interface"),
    pointer_default(unique)
]
interface IsxcshFolderRIC : IDispatch  // preferably - : IsxcshRICOperation
{
    [id(16), helpstring("Function GetPresentation")]
    HRESULT GetPresentation([out,retval] BSTR* outFileName);

    [id(17), helpstring("Function SetPresentation")]
    HRESULT SetPresentation([in] BSTR inFileName,
                            [out,retval] VARIANT_BOOL* bResult);
};
```

```
//------------ IsxcshComRIC

[
    object,
    uuid(CFBB8622-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshComRIC interface"),
    pointer_default(unique)
]
interface IsxcshComRIC : IDispatch  // preferably - : IsxcshRICOperation
{
    [id(16), helpstring("Function GetPresentation")]
    HRESULT GetPresentation([out,retval] BSTR* outFileName);

    [id(17), helpstring("Function SetPresentation")]
    HRESULT SetPresentation([in] BSTR inFileName,
                            [out,retval] VARIANT_BOOL* bResult);

    [id(18), helpstring("Function CacheContent")]
    HRESULT CacheContent([in, out] VARIANT_BOOL* bLoginOK,
                         [out,retval] VARIANT_BOOL* bResult);

    [id(19), helpstring("Function GetAttributes")]
    HRESULT GetAttributes([in] BSTR inClassName,
                          [out,retval] IsxcshCollection** outRefRecordItems);
};


//------------ IsxcshViewRIC2

[
    object,
    uuid(CFBB8623-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshViewRIC2 interface"),
    pointer_default(unique)
]
interface IsxcshViewRIC2 : IDispatch  // preferably - : IsxcshRICInformation
{
    [id(16), helpstring("Function TargetServerID")]
    HRESULT TargetServerID([out, retval] long* outTargetServerID);

    [id(17), helpstring("Function TargetRecordID")]
    HRESULT TargetRecordID([out, retval] BSTR* outTargetRecordID);

    [id(18), helpstring("Function TargetRCID")]
    HRESULT TargetRCID([out, retval] BSTR* outTargetRCID);

    [id(19), helpstring("Function TargetServer")]
    HRESULT TargetServer([out,retval] IsxcshServer** outRefServer);

    [id(20), helpstring("Function CacheTarget")]
    HRESULT CacheTarget([in, out] VARIANT_BOOL* bLoginOK,
                        [out,retval] VARIANT_BOOL* bResult);
};


//------------ IsxcshDataRIC

[
    object,
```

```
    uuid(CFBB8624-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshDataRIC interface"),
    pointer_default(unique)
]
interface IsxcshDataRIC : IDispatch  // preferably - : IsxcshRICInformation
{
};


//------------ IsxcshViewRIC1

[
    object,
    uuid(CFBB8625-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshViewRIC1 interface"),
    pointer_default(unique)
]
interface IsxcshViewRIC1 : IDispatch  // preferably - : IsxcshRICInformation
{
};


//------------ IsxcshRIShape

[
    object,
    uuid(CFBB8626-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshRIShape interface"),
    pointer_default(unique)
]
interface IsxcshRIShape : IDispatch
{
    [id(1), helpstring("Function ServerID")]
    HRESULT ServerID([out,retval] long* outServerID);

    [id(2), helpstring("Function LoginID")]
    HRESULT LoginID([out,retval] long* outLoginID);

    [id(3), helpstring("Function RecordID")]
    HRESULT RecordID([out,retval] BSTR* outRecordID);

    [id(4), helpstring("Function RCID")]
    HRESULT RCID([out,retval] BSTR* outRCID);

    [id(5), helpstring("Function SynapsesType")]
    HRESULT SynapsesType([out,retval] BSTR* outSynapsesType);

    [id(6), helpstring("Function Parent")]
    HRESULT GoParent([out,retval] VARIANT_BOOL* bResult);

    [id(7), helpstring("Function FirstChild")]
    HRESULT GoFirstChild([out,retval] VARIANT_BOOL* bResult);

    [id(8), helpstring("Function Succ")]
    HRESULT GoSucc([out,retval] VARIANT_BOOL* bResult);

    [id(9), helpstring("Function Pred")]
    HRESULT GoPred([out,retval] VARIANT_BOOL* bResult);
```

```
    [id(10), helpstring("Function HomeRIC")]
    HRESULT HomeRIC([out,retval] IsxcshRICShape** outRefHomeRIC);

    [id(11), helpstring("Function Clone")]
    HRESULT Clone([out,retval] IsxcshRIShape** outRIShape);
};


//------------ IsxcshRIInformation

[
    object,
    uuid(CFBB8627-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshRIInformation interface"),
    pointer_default(unique)
]
interface IsxcshRIInformation : IDispatch
{
    [id(1), helpstring("Function ServerID")]
    HRESULT ServerID([out,retval] long* outServerID);

    [id(2), helpstring("Function LoginID")]
    HRESULT LoginID([out,retval] long* outLoginID);

    [id(3), helpstring("Function RecordID")]
    HRESULT RecordID([out, retval] BSTR* outRecordID);

    [id(4), helpstring("Function RCID")]
    HRESULT RCID([out, retval] BSTR* outRCID);

    [id(5), helpstring("Function ClassName")]
    HRESULT ClassName([out, retval] BSTR* outClassName);

    [id(6), helpstring("Function Value")]
    HRESULT Value([out, retval] BSTR* outValue);

    [id(7), helpstring("Function Type")]
    HRESULT Type([out, retval] BSTR* outType);

    [id(8), helpstring("Function LogTime")]
    HRESULT LogTime([out, retval] BSTR* outLogTime);

    [id(9), helpstring("Function LogUserID")]
    HRESULT LogUserID([out, retval] BSTR* outLogUserID);

    [id(10), helpstring("Function InvalidationTime")]
    HRESULT InvalidationTime([out, retval] BSTR* outInvalidationTime);

    [id(11), helpstring("Function InvalidationUserID")]
    HRESULT InvalidationUserID([out, retval] BSTR* outInvalidationUserID);

    [id(12), helpstring("Function EventBeginTime")]
    HRESULT EventBeginTime([out, retval] BSTR* outEventBeginTime);

    [id(13), helpstring("Function EventEndTime")]
    HRESULT EventEndTime([out, retval] BSTR* outEventEndTime);

    [id(14), helpstring("Function Cluster")]
    HRESULT Cluster([out, retval] BSTR* outCluster);

    [id(15), helpstring("Function InternalDataType")]
    HRESULT InternalDataType([out, retval] BSTR* outInternalDataType);
```

```
    [id(16), helpstring("Function Format")]
    HRESULT Format([out, retval] BSTR* outFormat);
};


//------------ IsxcshRIOperation

[
    object,
    uuid(CFBB8628-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshRIOperation interface"),
    pointer_default(unique)
]
interface IsxcshRIOperation : IDispatch  // preferably - : IsxcshRIInformation
{
};


//------------ IsxcshRecordItem

[
    object,
    uuid(CFBB8632-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshRecordItem interface"),
    pointer_default(unique)
]
interface IsxcshRecordItem : IDispatch  // preferably - : IsxcshRIOperation
{
};


//------------ IsxcshCollection

[
    object,
    uuid(CFBB8629-19C0-11d4-9639-0060979B4844),
    oleautomation,
    dual,
    helpstring("IsxcshCollection interface"),
    pointer_default(unique)
]
interface IsxcshCollection : IDispatch
{
    [id(1), helpstring("Function MoveFirst")]
    HRESULT MoveFirst([out, retval] VARIANT_BOOL* bResult);

    [id(2), helpstring("Function MoveNext")]
    HRESULT MoveNext([out, retval] VARIANT_BOOL* bResult);

    // If collection would be encapsulated VB Collection
    // [id(1), helpstring("Function Item")]
    // HRESULT Item([in] long idx,
    //              [out, retval] IDispatch** ppunk);
    //
    // [id(2), helpstring("Function Count")]
    // HRESULT Count([out, retval] long* pi4);
    //
    // [propget, id(3), helpstring("property NewEnum")]
    // HRESULT NewEnum([out, retval] IDispatch** ppunk);
    //
    // [id(4), helpstring("Function Add")]
```

```
        // HRESULT Add([in] VARIANT* Item,
        //             [in, optional] VARIANT* Key,
        //             [in, optional] VARIANT* Before,
        //             [in, optional] VARIANT* After);
        //
        // [id(5), helpstring("Function MoveNext")]
        // HRESULT Remove([in] VARIANT* Index);
    };

};

//-------------- EOF
```

B. Web Server Access

Client Requests

The following XML DTD defines the client request format that is expected by the WP2 platform:

```
<!-- XML DTD for requests accepted by the Oslo Synapses Server -->
<!ELEMENT OSSrequest ( Function+ ) >

<!ELEMENT Function ( Arg* ) >
<!ATTLIST Function  Name  CDATA  #REQUIRED>

<!ELEMENT Arg (#PCDATA)>
<!ATTLIST Arg  Name  CDATA  #REQUIRED>
```

Notice: The current implementation of the WP2 platform complies with SOAP technically by using XML over http, but it does **not** comply with the SOAP protocol with respect to the SOAP XML DTD for client requests.

No validation will be performed against this DTD when parsing it on the server side (if invalid then a more informative error message will be returned to the client), but the DTD will be available on WP2 web-servers in a file named *"oss-client-request.dtd"*.

The following is a list of functions currently supported. Notice that arguments in bold (**Arg**) indicates mandatory arguments, and in those cases where there is a predefined set of alternative argument values then the value in bold indicates the default value if this argument is not provided.

```
<Function Name="LogOn">
    <Arg Name="User">....</Arg>
        <Arg Name="Password">....</Arg>
    <Arg Name="ResponseType">..{html | xml | wml}..</Arg>
</Function>

<Function Name="LogOff">
    <Arg Name="User">....</Arg>
</Function>

<Function Name="RecordInfo">
    <Arg Name="User">....</Arg>
        <Arg Name="RecordID">....</Arg>
    <Arg Name="Retrieval">..{shape | all}..</Arg>
        <Arg Name="ResponseType">..{html | xml | wml}..</Arg>
</Function>

<Function Name="FolderInfo">
    <Arg Name="User">....</Arg>
        <Arg Name="RecordID">....</Arg>
    <Arg Name="RCID">....</Arg>
        <Arg Name="Retrieval">..{shape | all}..</Arg>
        <Arg Name="ResponseType">..{html | xml | wml}..</Arg>
</Function>

<Function Name="DocumentInfo">
    <Arg Name="User">....</Arg>
        <Arg Name="RecordID">....</Arg>
    <Arg Name="RCID">....</Arg>
        <Arg Name="Retrieval">..{shape | all}..</Arg>
    <Arg Name="ResponseType">..{html | xml | wml}..</Arg>
</Function>
```

The "ResponseType" argument is explained below (the two defaults are due to this being client browser dependent).

The "Retrieval" argument can be either *"shape"* or *"all"*. "Shape" means that only the structure of a record, folder or document will be returned, not information within a document (no RIC's within a ComRIC except the ComRIC itself, in Synapses terms).

The reason why the user name must be provided with each request is that a particular *client* accessing the information can be logged on to the same server as several *users* during the same *session*. In a later version a default (the most recent logon) will be provided for this argument since in most cases a client will be logged on as a single user for the duration of a session.

According to the DTD, several functions can be combined into a single request, e.g. the following request:

```
<OSSrequest>
      <Function Name="LogOn">
            <Arg Name="User">onordmann</Arg>
            <Arg Name="Password">xyz</Arg>
            <Arg Name="ResponseType">xml</Arg>
      </Function>
      <Function Name="RecordInfo">
      <Arg Name="User">onordmann</Arg>
            <Arg Name="RecordID">93003449</Arg>
            <Arg Name="Retrieval">all</Arg>
      </Function>
</OSSrequest>
```

but notice that (in the current implementation) results will only be provided to the client for the last of the functions in the request. Thus there is no use in combining several record/folder/document requests.

Sending Client Requests

The current implementation of the WP2 platform supports three different alternatives for sending a request from a client to the web server. Two of them for "production" use, and one only for simple "demonstration" purposes.

1. *QueryString - http GET command*

   The http GET command, as a socalled QueryString, means that the XML formatted request from the client is added to the web address as follows:

   ```
   http://citroen.nr.no/synexdemo/oss.asp?<OSSrequest><Function Name="LogOn">
                        <Arg Name="User">admin</Arg><Arg Name="Password">x</Arg>
                        <Arg Name="ResponseType">xml</Arg></Function></OSSrequest>
   ```

   This can be useful for demonstration purposes to make things explicit, but there are also several disadvantages; e.g. there is a limit to the length of GET commands so parts of it may be truncated, and the requests will be visible to "anyone" (e.g. in logs).

2. *HTML Forms - http POST command*

   Using an HTML Form to send a POST command is a better solution than the above GET command. There are no (at least practically important) limitations to the length of the XML request within a POST command. The following is an example of making such a request:

   ```
   <FORM METHOD="POST" ACTION="http://citroen.nr.no/synexdemo/oss2.asp">
         <INPUT TYPE="hidden" NAME="XMLRequest"
                              VALUE='<OSSrequest><Function Name="LogOn">
                                    <Arg Name="User">emil</Arg>
                                    <Arg Name="Password">x</Arg>
                                    <Arg Name="ResponseType">xml</Arg>
                                    </Function></OSSrequest>'/>
         <INPUT TYPE="submit" VALUE="Log On"/>
   </FORM>
   ```

   Notice: The name of the input/form field with the XML formatted request must be "XMLRequest" to be accepted/found by the current WP2 implementation.

3. *XMLHttpRequest ActiveX control - http POST command*

A http POST command can alternatively be sent by using an ActiveX control like *XMLHttpRequest*, which is available within the Micrsoft XML v.2.0 parser *"msxml.dll"*. The following is an example of how to use this control from Visual Basic:

```
Dim refHttp          As MSXML.XMLHTTPRequest
Dim refXMLDoc        As MSXML.DOMDocument
Dim strServerAddress As String
Dim strXMLrequest    As String

strServerAddress = "http://citroen.nr.no/synexdemo/oss.asp"
strXMLrequest = "<OSSrequest><Function Name=" & Chr(34) & "LogOn" & Chr(34) & _
                "><Arg Name=" & Chr(34) & "User" & Chr(34) & _
                ">admin</Arg><Arg Name=" & Chr(34) & "Password" & Chr(34) & _
                ">x</Arg><Arg Name=" & Chr(34) & "ResponseType" & Chr(34) & _
                ">xml</Arg></Function></OSSrequest>"

Set refHttp = New MSXML.XMLHTTPRequest

Call refHttp.Open("POST", strServerAddress, False)

' NB! To distinguish this POST command from the Forms POST command
Call refHttp.SetRequestHeader("XMLRequest", "XMLHttpRequest")

Set refXMLDoc = New MSXML.DOMDocument
refXMLDoc.Async = False
refXMLDoc.ValidateOnParse = False
If (Not refXMLDoc.LoadXML(strXMLrequest)) Then
    ...error in XML request...
End If
Call refHttp.Send(refXMLDoc)

...result available in refHttp.responseXML
```

Notice: To be able to distinguish this POST command from the other Forms POST command, a request header variable named *"XMLRequest"* is defined with the value *"XMLHttpRequest"*. Without this variable defined and set, a WP2 server will not be able to get the XML request sent.

The SynEx client uses the *XMLHttpRequest* component in its implementation.

SynExML Server Response

The server response to a valid request for record, folder or document information will be XML valid according to the *SynExML*, which again is based on the generic FHCR structure defined by the Synapses Server specification [3][4]. That is, such XML, and including a reference to a default XSL specification, will be returned to the client provided that *either* the client makes an explicit request for *"xml"* via the "ResponseType" argument, *or* no "ResponseType" argument is provided *and* the client browser is Internet Explorer v.5.0.

If the client specifies *"html"* for the "ResponseType" argument, *or* no "ResponseType" argument is provided *and* the client uses any other browser than Internet Explorer v.5.0, then the XML generated for the information requested will be transformed into HTML on the server-side via the use of an XSL specification.

Of course, requesting HTML instead of XML is only relevant in those cases where the client only wants to browse the information received, and the browser is unable to transform XML into e.g. HTML via XSL; either the default XSL provided from the server, or some other XSL that the client has access to.

WML formatted information is not available in the current version.

Other Server Responses

Notice that the server does not respond with XML valid according to SynExML for LogOn and LogOff requests, nor if an error occurs when processing a request (e.g. trying to retrieve information without being logged on, or information for which the user lacks authorization).

The following DTD defines alternative server responses for a successful LogOn, a successful LogOff, or an error situation:

```
<!-- XML DTD for non-SynExML responses returned by the Oslo Synapses Server -->

<!ELEMENT OSSresponse ( Success | Failure )* >

<!ELEMENT Success ( Function )>
<!ELEMENT Function (#PCDATA)>

<!ELEMENT Failure ( Error* ) >
<!ELEMENT Error ( Source, Number, Description )>
<!ELEMENT Source (#PCDATA)>
<!ELEMENT Number (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
```

This DTD will be available on WP2 web-servers in a file named *"oss-server-response.dtd"*.

C. SynExML DTD (Document Type Definition)

```
<!-- ============================================================ -->
<!--    Name: SynExML                                             -->
<!--    Version: 2.1 beta 4                                       -->
<!--    Date: 04/02/2000                                          -->
<!--    Copyright: SynEx                                          -->
<!--                                                              -->
<!--    Editor:                                                   -->
<!--    Benjamin JUNG (TCD, <benjamin.jung@cs.tcd.ie>)            -->
<!--                                                              -->
<!--    Contributing editor:                                      -->
<!--    Tony AUSTIN (UCL, <t.austin@chime.ucl.ac.uk>)             -->
<!--                                                              -->
<!--    Contributing authors:                                     -->
<!--    Jose ANDANY (HUG, <jose.andany@dim.hcuge.ch>)             -->
<!--    Egil P. ANDERSEN (SHS, <egil.paulin.andersen@nr.no>)      -->
<!--    Stephane SPAHNI (HUG, <stephane.spahni@dim.hcuge.ch>)     -->
<!--    Yigang XU (Broussais, <xu@hbroussais.fr>)                 -->
<!--    Vladimir YURPALOV (IBEX, <vdy@ibex.ch>)                   -->
<!--    Andrei EMELIANENKO (IBEX, <ave@ibex.ch>)                  -->
<!--    Dipak KALRA (UCL, <d.kalra@chime.ucl.ac.uk>)              -->
<!-- ============================================================ -->
<!-- ============================================================ -->
<!--      GENERAL COMMENTS                                        -->
<!--      It is recommended to use the ISO Date/Time format to    -->
<!--      express Times and Dates in ELEMENT content and          -->
<!--      ATTRIBUTE values.                                       -->
<!-- ============================================================ -->

<!-- ============================================================ -->
<!--      % RICattributes                                         -->
<!--      Attributes common to every RIC in the Synapses Server   -->
<!--      specification; i.e., attributes defined in class        -->
<!--      Record Component and class RIC in the Synapses Object   -->
<!--      View.                                                   -->
<!--                                                              -->
<!--      Class RIC inherits class Record Component in the        -->
<!--      Synapses Server specification.                          -->
<!--                                                              -->
<!--      Record Component is the root class in the Synapses      -->
<!--      Object View. The Object View contains the classes       -->
<!--      from which objects constituting actual healthcare       -->
<!--      records are instantiated. The Synapses Class View       -->
<!--      contains classes from which objects constituting        -->
<!--      healthcare record classes are instantiated              -->
<!-- ============================================================ -->

<!ENTITY % RICattributes "ClassName          CDATA    #REQUIRED
                          RCID               ID       #REQUIRED
                          RecordID           CDATA    #IMPLIED
                          LogUserID          CDATA    #IMPLIED
                          LogTime            CDATA    #IMPLIED
                          InvalidationUserID CDATA    #IMPLIED
                          InvalidationTime   CDATA    #IMPLIED">

<!-- ============================================================ -->
<!--      % RIAttributes                                          -->
<!--      Attributes common to every RecordItem in the Synapses   -->
<!--      Server specification; i.e., attributes defined in       -->
<!--      class Record Component and class RecordItem in the      -->
<!--      Synapses Object View.                                   -->
<!--                                                              -->
<!--      Class RecordItem inherits class Record Component in     -->
<!--      the Synapses Server specification.                      -->
```

```
<!-- ========================================================== -->

<!ENTITY % RIattributes "ClassName          CDATA    #REQUIRED
                         RCID               ID       #REQUIRED
                         RecordID           CDATA    #IMPLIED
                         LogUserID          CDATA    #IMPLIED
                         LogTime            CDATA    #IMPLIED
                         InvalidationUserID CDATA    #IMPLIED
                         InvalidationTime   CDATA    #IMPLIED
                         EventBeginTime     CDATA    #IMPLIED
                         EventEndTime       CDATA    #IMPLIED">

<!-- ========================================================== -->
<!--      % CommonRICAttributes                                 -->
<!--      CommonRICAttributes are attributes of RIC's that are  -->
<!--      not defined in the Synapses specification, but which  -->
<!--      all sites agree to add to this DTD.                   -->
<!--                                                            -->
<!--      The Language attribute is used to specify the language-->
<!--      used for terms within the element to which it belongs.-->
<!-- ========================================================== -->

<!ENTITY % CommonRICAttributes "Type      CDATA  #IMPLIED
                                Language  CDATA  #IMPLIED">

<!-- ========================================================== -->
<!--      % CommonRIAttributes                                  -->
<!--      CommonRIAttributes are attributes of RecordItem's     -->
<!--      that are not defined in the Synapses specification,   -->
<!--      but which all sites agree to add to this DTD.         -->
<!--                                                            -->
<!--      The Language attribute is used to specify the         -->
<!--      language used for terms within the element to which   -->
<!--      it belongs.                                           -->
<!--                                                            -->
<!--      The DataType attribute is used to specify type of     -->
<!--      data value carried by the RecordItem to which it      -->
<!--      belongs.                                              -->
<!-- ========================================================== -->

<!ENTITY % CommonRIAttributes "Type      CDATA  #IMPLIED
                               Language  CDATA  #IMPLIED
                               DataType  CDATA  #IMPLIED">

<!-- ========================================================== -->
<!--      SynExML (SynEx Markup Language)                       -->
<!--      A SynExML file can contain a set of RecordFolder's,   -->
<!--      FolderRIC's and ComRIC's in any sequence.             -->
<!--                                                            -->
<!--      Source specifies from where the XML is produced       -->
<!-- ========================================================== -->

<!ELEMENT SynExML (RecordFolder | FolderRIC | ComRIC)*>
<!ATTLIST SynExML Version CDATA  #REQUIRED
                  Source  CDATA  #REQUIRED>

<!-- ========================================================== -->
<!--      RCproperty                                            -->
<!--      RCproperties are (name,value) pairs.                  -->
<!--      They are not part of the Synapses Server              -->
<!--      specification, but they are included to support       -->
<!--      site-specific attributes. That is, conceptually they  -->
<!--      should be considered a site-specific addition to the  -->
<!--      ATTLIST for a particular element (e.g. the            -->
<!--      RecordFolder), and they are only included as nested   -->
<!--      elements within e.g. RecordFolder for DTD-technical   -->
```

```
<!--      reasons. For this reason they must always be the      -->
<!--      first elements within the element to which they       -->
<!--      belong (when parsing e.g. RecordFolder its attributes -->
<!--      should be known).                                     -->
<!-- ========================================================== -->

<!ELEMENT RCproperty (#PCDATA)>
<!ATTLIST RCproperty Name CDATA #REQUIRED>

<!-- ========================================================== -->
<!--      RecordFolder (a healthcare record)                    -->
<!--      In Synapses every healthcare record is rooted in a    -->
<!--      single RecordFolder object, and the structure of a    -->
<!--      HCR is seen as a tree-structure of RIC's with         -->
<!--      hyperlinks (ViewRIC2's) between them.                 -->
<!--      The elements that can be nested within a RecordFolder -->
<!--      element is as specified in the Synapses Server        -->
<!--      specification; i.e., either a single ViewRIC2, or a   -->
<!--      set of ComRIC's and/or FolderRIC's in any sequence.   -->
<!--      In Synapses RecordItem's are used to represent data   -->
<!--      values (as "dynamic attributes") attached to a        -->
<!--      particular RIC (a RIC as a structural element in a    -->
<!--      HCR). Thus beside its RIC children, a RecordFolder    -->
<!--      can also contain a set of RecordItem's.               -->
<!-- ========================================================== -->

<!ELEMENT RecordFolder
          (RCproperty*,
      ( (ComRIC | FolderRIC | RecordItem)* |
        (ViewRIC2, RecordItem*) ) )>
<!ATTLIST RecordFolder %CommonRICAttributes;
                       %RICattributes;>

<!-- ========================================================== -->
<!--      FolderRIC (a healthcare folder)                       -->
<!--      The elements that can be nested within a FolderRIC    -->
<!--      element is as specified in the Synapses Server        -->
<!--      specification (the same as for a RecordFolder -       -->
<!--      RecordFolder is a specialisation of FolderRIC in      -->
<!--      Synapses).                                            -->
<!-- ========================================================== -->

<!ELEMENT FolderRIC
          (RCproperty*,
      ( (ComRIC | FolderRIC | RecordItem)* |
        (ViewRIC2, RecordItem*) ) )>
<!ATTLIST FolderRIC %CommonRICAttributes;
                    %RICattributes;>

<!-- ========================================================== -->
<!--      ComRIC (a healthcare document)                        -->
<!--      The elements that can be nested within a ComRIC       -->
<!--      element is as specified in the Synapses Server        -->
<!--      specification; i.e., a set of DataRIC's, ViewRIC1's   -->
<!--      and/or ViewRIC2's in any sequence.                    -->
<!--      In addition it can contain a set of RecordItem's      -->
<!--      representing data values (as "dynamic attributes")    -->
<!--      attached to this ComRIC.                              -->
<!-- ========================================================== -->

<!ELEMENT ComRIC
          ( RCproperty*,
            (DataRIC | ViewRIC1 | ViewRIC2 | RecordItem)* )>
<!ATTLIST ComRIC %CommonRICAttributes;
                 %RICattributes;>
```

```
<!-- ========================================================== -->
<!--      DataRIC (a "field" within a healthcare document)      -->
<!--      The elements that can be nested within a DataRIC      -->
<!--      element is as specified in the Synapses Server        -->
<!--      specification; i.e., a set of more DataRIC's,         -->
<!--      ViewRIC1's and/or ViewRIC2's in any sequence.         -->
<!--      In addition it can contain a set of RecordItem's      -->
<!--      representing data values (as "dynamic attributes")    -->
<!--      attached to this DataRIC.                             -->
<!-- ========================================================== -->

<!ELEMENT DataRIC
          ( RCproperty*,
            (DataRIC | ViewRIC1 | ViewRIC2 | RecordItem)* )>
<!ATTLIST DataRIC %CommonRICAttributes;
                  %RICattributes;>

<!-- ========================================================== -->
<!--      ViewRIC1 (a "computed field" within a healthcare      -->
<!--                document)                                   -->
<!--      In Synapses a ViewRIC1 is similar to a DataRIC except -->
<!--      that its RecordItem's (its data values as "dynamic    -->
<!--      attributes") are computed on demand.                  -->
<!-- ========================================================== -->

<!ELEMENT ViewRIC1 (RCproperty*, RecordItem*)>
<!ATTLIST ViewRIC1 %CommonRICAttributes;
                   %RICattributes;>

<!-- ========================================================== -->
<!--      ViewRIC2 (a hyperlink between RIC's in two healthcare -->
<!--                records)                                    -->
<!--      A ViewRIC2 specifies a link either to another RIC     -->
<!--      within the same record, to a RIC within another record-->
<!--      at the same server, or to a RIC within another record -->
<!--      at another server. The Destination element specifies  -->
<!--      the link target.                                      -->
<!-- ========================================================== -->

<!ELEMENT ViewRIC2 (RCproperty*, Destination?, RecordItem*)>
<!ATTLIST ViewRIC2 %CommonRICAttributes;
                   %RICattributes;>

<!ELEMENT Destination EMPTY>
<!ATTLIST Destination ServerID CDATA #REQUIRED
                      RecordID CDATA #REQUIRED
                      RCID     CDATA #REQUIRED>

<!-- ========================================================== -->
<!--      RecordItem                                            -->
<!--      RecordItem is defined within the Synapses Server      -->
<!--      specification, but it is not defined with any content -->
<!--      (DataItem, as a specialisation of RecordItem, is just -->
<!--      included as an example (page 6 in the computational   -->
<!--      viewpoint)). An implementation of a Synapses Server   -->
<!--      is therefore free to define the content of            -->
<!--      RecordItem's as suits it best. However, their purpose -->
<!--      are as "dynamic attributes" to RIC's; i.e. RIC's      -->
<!--      define the structure of HCR while RI's contain the    -->
<!--      data values attached to them. Therefore, to make      -->
<!--      their "value" explicit, a Value element is added to   -->
<!--      their DTD definition.                                 -->
<!--      To allow for RecordItem's to define tree-structures   -->
<!--      of values, "RecordItem*" is added to the DTD          -->
<!--      specification.                                        -->
<!--      As for the RIC's defined above, RCproperty* is only   -->
```

```
<!--      meant to be used for extending the ATTLIST with       -->
<!--      site-specific attributes.                             -->
<!--      It is recommended to attach childelements of          -->
<!--      RecordItem in the following order: RCproperty,        -->
<!--      ElementItem, LinkItem, RecordItem, #PCDATA. It is     -->
<!--      also also recommended to keep the #PCDATA in a single -->
<!--      'data-island'.                                        -->
<!-- ========================================================== -->

<!ELEMENT RecordItem
          ( #PCDATA | RCproperty | ElementItem | LinkItem |
            RecordItem )*>

<!ATTLIST RecordItem %CommonRIAttributes;
                     %RIattributes;>
<!-- ==== END OF SynExML v.2.1 beta 3 ========================= -->
```