ViroLab Virtual Laboratory: Data Access Services Prototype Manual

Project Start:	01/03/2006
Project Duration:	36 Months
Priority area	2.4.11
Contract No.:	INFSO-IST-027446
Website:	http://www.virolab.org

Due-Date:	31-08-2007
Delivery:	31-08-2007
Lead Partner:	USTUTT
Dissemination Level:	confidential
Status:	draft
Approved:	
Version:	0.4

Log of Document

Version	Date	Changes Summary	Authors
0.1	11/08/2007	Initial version of DAS manual	Matthias Assel
0.2	17/08/2007	First document reviewing	Matthias Assel
0.3	19/08/2007	Extensive proofreading and editing	Matthias Assel, Aenne Löhden
0.4	05/09/2007	Included latest DAS functionality	Matthias Assel

TABLE OF CONTENTS

This research is partly funded by the European Commission IST-2005-027446 Project “ViroLab”.

1. INTRODUCTION

The complexity of data management on a Grid arises from the scale, dynamism, autonomy, heterogeneity, and distribution of resources. To conceal these complexities of the underlying infrastructure, a sophisticated management system needs to be developed, which ensures that the resources appear transparent to their users. This could be achieved by hiding the different data resources and their internals behind a layer of virtualization services that guarantees data access in a consistent, data resource-independent way.

The Data Access Services (DAS) consist of a set of such virtualization services that provide interfaces for querying, updating, transforming and delivering data to various data resources via standard web services. The current version, which is described in detail by this document, allows users to collect any kind of (meta) information of an underlying resource including the used data management technology (database system), schema specification, and availiability of the resource itself. This information can then be used by clients to specify a query that accesses the corresponding database similarly as they would proceed within their local environment. Not only single data resources can be accessed but also queries to multiple databases can be simultaneously performed using specific interfaces provided by the DAS. There are currently some restrictions while sending requests to more than one resource but the DAS is in constant development and the functionalities are permanently extended.

This documentation is intended to support other developers working in the ViroLab project to connect their components and applications with the DAS in order to get a unified entry point for distributed data access. It describes the main steps to set up and use the services developed and offers a deeper insight into the implementation structure of the first prototype.

1.1. REFERENCES AND SOURCE CODE

The current DAS JavaDoc source code documentation can be found at the following web address:

http://www.hlrs.de/organization/ds/projects/virolab/dasapi/index.html

The latest source code of the DAS and appropriate test applications can be downloaded from the ViroLab SVN hosted by University of Stuttgart:

https://svn.gforge.hlrs.de/svn/virolab/trunk/modules/DataAccess/ViroLab_v.0.2/

The access to the repository is restricted only to members of the ViroLab consortium. An open source release of the DAS is planned to be provided at the end of the project.

2. PROTOTYPE USAGE

In order to deal with the DAS, there are some technical prerequisites for components and applications regarding hardware and software. In a typical Grid infrastructure, the DAS is usually installed at one central location where it can be accessed from any external machine. This section gives an overview on how to interact with the DAS from a particular machine and also explains basic functionalities and some advanced features.

2.1. RUNNING THE PROTOTYPE

2.1.1. Operating Requirements

To communicate with the DAS, which is based on standard web service interfaces, an user - could be an application or a component – has to ensure that the latest interface description (WSDL specification) is used. These descriptions are normally used to automatically generate so-called client stubs, a set of Java classes, which allow the interaction with the corresponding services in a smooth way.

The following sections shall provide setup information that is required to use the generated classes and to finally interact with the DAS.

2.1.1.1. Local Hardware Requirements

The hardware requirements for using the client stubs are not very high as any today's computer hardware able to run Java 5 software should be sufficient. An Internet connection is needed in order to call the DAS. As the services are quite communication dependent, the optimal configuration will have a low-latency Internet connection.

2.1.1.2. Local Software Requirements

The software may run on almost every 32-bit operating system like Windows XP, Vista or Server 2003 as well as on different Linux distributions like Ubuntu or SuSe Linux. It only requires a running installation of the Java 5 Runtime Environment, which can be directly downloaded free of charge from SUN's website (http://java.sun.com/j2se/1.5.0/).

Furthermore, to ensure a proper working with the DAS, a number of third-party Java libraries (compiled and zipped in *jar* files) is needed:

- activation.jar (http://java.sun.com/products/javabeans/jaf/index.jsp)
- addressing-1.0.jar (http://jakarta.apache.org/addressing/)
- axis.jar (http://ws.apache.org/axis/)
- commons-discovery.jar (http://jakarta.apache.org/commons/discovery/)
- commons-logging.jar (http://jakarta.apache.org/commons/logging/)
- jaxrpc.jar (http://ws.apache.org/axis/)
- log4j.jar (http://jakarta.apache.org/log4j/)
- saaj.jar (http://ws.apache.org/axis/)
- servlet.jar (http://jakarta.apache.org/tomcat/)
- wsdl4j.jar (http://sourceforge.net/projects/wsdl4j/)
- wss4j.jar (http://ws.apache.org/wss4j/)

- xalan.jar (http://xml.apache.org/xalan-j/)
- xercesImpl.jar (http://xml.apache.org/xerces2-j/)
- xml-apis.jar (http://xml.apache.org/xerces2-j/)
- xmlsec.jar (http://xml.apache.org/security/)

Principially, there are two ways to get the latest client stubs. Firstly, one can download them from the SVN repository (see part 1.1) or secondly one can create them dynamically everytime before using the DAS. The second option would be more flexible but requires additional effort from the developer. He needs to modify the source code to create the Java classes on-the-fly during program execution. The developer can for example use the Axis tool *WSDL2JAVA* available within the Axis library earlier mentioned or he can use the *CreateDASStubs* class shipped with the DAS, which is also available within the SVN repository.

2.1.1.3. Grid infrastructure requirements

The main requirement for using the DAS is that it has to be operational and that it publishes its web service interaces (WSDL) externally. Being unavailable, the user's requests will be rejected by the DAS.

If additional security between a client and the DAS is required due to some specific reasons, there is an optional feature that allows secure communication with the services - the messages are encrypted and signed. To turn these security functionalities on, the provider of the DAS needs to set some property values within the DAS configuration files and a valid X.509 certificate needs to be in place as well. The client also needs one of these certificates in order to encrypt and decrypt the corresponding messages. Both certificates must be trusted by the same certificate authority (CA), otherwise the integrity of the exchanged messages is not guaranteed.

Further information on security principles and mechanisms can be found at the following website:

http://gdp.globus.org/gt4-tutorial/multiplehtml/pt03.html

2.1.2. Step-by-Step User Setup

Step 1: Download the precompiled stubs from the SVN or create them on-the-fly using the following WSDL file:

http://angelina.hlrs.de:8080/wsrf/services/DataAccessService?wsdl

Step 2: Make sure to have a proper installation of the Java 5 Runtime Environment on the local site (simply search for java executable). If not, download and install it properly.

Step 3: Get all the third-party libraries of the software. Please check with *Local Software Requirements* section where to download these files.

Step 4: Use the stubs together with all the other libraries to include DAS interactions within your own source code. One can proceed like the following example where a connection to the DAS is established and different resource IDs are requested and simply printed out.

```
DataAccessServiceAddressingLocator locator = new DataAccessServiceAddressingLocator();
EndpointReferenceType endpoint = new EndpointReferenceType();
```

```
try
{
  String serviceURI = "http://angelina.hlrs.de:8080/wsrf/services/DataAccessService";
  String ogsaService = "http://csharp.hlrs.de:9090/wsrf/services/hospitals";
  endpoint.setAddress(new Address(serviceURI));
  DataAccessPortType dataService = locator.getDataAccessPortTypePort(endpoint);
  if(dataService != null)
  {
    System.out.println("Connecting to service at: "+serviceURI);
    DataResourceList result = dataService.getAvailableDataResources(ogsaService);
    if(result != null)
    {
      for(int i = 0; i < result.getResources().length; i++)
      {
        ResourceParams currentResource = result.getResources(i);
        if(currentResource != null)
        {
          System.out.println("Found resource: "+currentResource.getResourceID());
        }
      }
    }
  }
  else
  {
    System.out.println("Cannot connect to service at: "+serviceURI);
  }
}
catch(RemoteException e)
{
  e.printStackTrace();
}
```

Step 5: Compile the source code and execute the program.

2.2. BASIC OPERATIONS

The DAS currently offers different functionalities for querying distributed data resources. The basic features allow the interaction with underlying databases in a common way but also provide specific methods such as distributed queries, download of publicly available rule sets, and more. Details on the interfaces currently available are described on the following website:

http://www.hlrs.de/organization/ds/projects/virolab/dasapi/index.html

In order to give users an idea of how to use the basic functionalities and also in which order these operations must be invoked, the following parts shall simply list the single methods and explain their basic features. For more details on the implementation, please refer to section 5.

Part 1 will explain the main interfaces for accessing remote data resources. Part 2 will concentrate on the submission of distributed queries to various resources concurrently. Finally, part 3 presents one specifically implemented routine to download publicly available rule sets while part 4 describes the principle how to store specific application data.

Part 1: Connecting to and querying from remote databases

- Initialize the corresponding service and resource. The Initparams argument of the method requires two parameters, the service URL and the abstract resource ID (typically an unique name).

  boolean init(InitParams params) where params is used the following way:

  ```
  InitParams initParams = new InitParams();
  initParams.setServiceLocation("http://csharp.hlrs.de:9090/wsrf/services/hospitals");
  initParams.setResourceID("ROME");
  ```

- Collect the data resource information of a particular resource. This information includes the database technology (e.g. MySQL, Postgres) and a list of keywords indicating available tables - this is currently a prerequisite, refer to section 2.4.

  DataResourceInformation getDataResourceInformation(String resourceID) where resourceID is "ROME"

- Query for the schema definition. This principal is currently based on plain SQL statements. To request schemes from different databases, please refer to the according SQL statements.

  DataResult getDataFromQuery("Describe currentTableName") -> currentTableName obtained from previous request

- Perform a concrete query. To query different databases, please refer to the according SQL statements.

  DataResult getDataFromQuery("Select * From currentTableName") or whatever you want…

Part 2: Submitting distributed queries

Submitting a distributed query to multiple resources is currently restricted to one specific method:

DataResult submitDistributedQuery(java.lang.String queryString)

This method requires a real SQL query as input and then automatically performs the following actions:

- Checks which resources are available
- Requests data resource information of each available resource
- Compares the keywords to the tables given by the query
- If corresponding resources are found, each of them is queried using the input statement
- Finally, the results are merged and the resource ID is added to each new data row as an additional primary key

Part 3: Requesting publicly available rule sets

To deal with such rule sets, one specific method was implemented that handles all relevant activities (download, version checking, submission to requester) based on the requirements of the DRS application.

The method can be directly called by the DRS or any other component. The only necessary argument to be passed to the function is the type of rule sets wanted while the version argument is optional. These arguments are simple string types and must have the following input values.

- String type: *ANRS|HIVDB|REGA*
- String version: e.g. *4.1.0* (optional)

The result can either be a notification message (String) saying that the current rule set version is up-to-date ('Your current version is up-to-date'), or an XML document (String) including the latest rule set.

To point out the functionality, there is a simple test application named *AccessRulesetsClient.* It is a Java Swing application, which allows the user to input the relevant information (required by the above explained method *RequestRuleSets*) and then starts to process this input by downloading a new version or to find out that the current one is up-to-date. The test application is available in the SVN at

https://svn.gforge.hlrs.de/svn/virolab/trunk/modules/DataAccess/ViroLab_v.0.2/TestClients.

The application can be started by typing the following into a command line:

ant runRuleSetClient

The following window appears on the desktop.

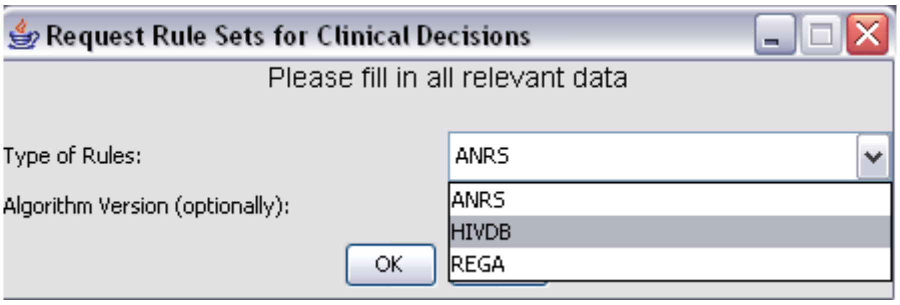
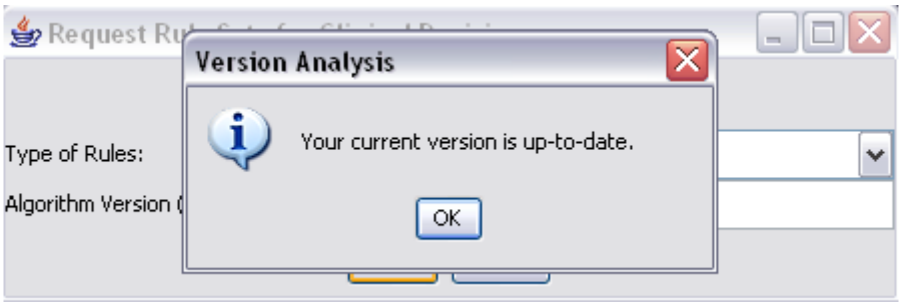

Request Rule Sets for Clinical Decisions
Please fill in all relevant data
Type of Rules:
ANRS
Algorithm Version (optionally):
ANRS
HIVDB
REGA
OK

Figure 2.1: Selecting the type of rule sets

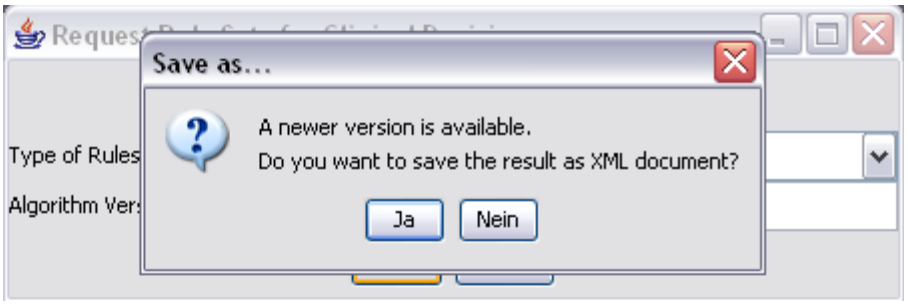
When the version is up-to-date, the following notification will be shown.



Your current version is up-to-date.

Figure 2.2: Received notification message

If a newer rule set version is available, a message box will pop up and ask whether to save the newer version or not (it will be saved as an XML file).



A newer version is available.
Do you want to save the result as XML document?

Figure 2.3: Save the newly available rule sets

Part 4: Storing relevant application data in a specific database

In order to track specific experiment results and user inputs related to these experiments, every application should store their input as well as output data in a corresponding database. In the current version, the DAS provides one particular method that allows to store appropriate application data:

```
boolean storeApplicationData(AppStoringParams appStoringparams)
```

It should be called whenever an application wants to store its data. The input argument required by the function consists of two parameters, which should be of the following input format:

- AppType appType: *DRS|RegaAlignment|RegaHIVSubType*
- String[] values: an array of values to be stored

Internally, the method firstly analyzes the application type and based on type value, the particular application-dependent function is called and all passed values are stored in the corresponding database (refer to section 5.3).

2.3. ADVANCED FEATURES

One advanced feature in the current release of the DAS is the integration of the services together with the authorization principle of Shibboleth. It is in an early stage of development so that the final user authorization is more or less implemented in a static way, meaning that as long as a user carries a specific attribute such as a role, institution, etc. he might be allowed to access a particular resource. There is not a real dynamic procedure for access control available yet but the future releases of the DAS will also contain such a dynamic authorization model where a so-called Policy Decision Point (PDP) can be asked whether a user has the necessary attributes to enable his access to a particular resource. For more information on Shibboleth and the principle used in ViroLab, please refer to the deliverables D2.2 and D3.2 submitted in month 12 and 9 respectively.

2.4. KNOWN PROBLEMS

The current version of the DAS software is in its first release but most of the basic functionality is considered stable. There are actually no known bugs but this might change while the software is in daily usage. Nevertheless, the DAS has got some limitations and yet unimplementd features, which shall be roughly listed and explained, and which will be covered in future releases.

1. Yet unimplemented features:
    - Dynamic user authorization: Using a PDP and user-defined policies (usually defined and managed by data providers themselves) to control the access to different resources
    - Meta query language to simplify communication with services: Developing a higher-level language that allows application users as well as developers to query resources without using real SQL statements but rather common well-known terms
    - Parallelization of distributed queries: Develop an algorithm that enables parallel processing of a distributed query – currently done in a sequential order - to increase performance and reliability of user queries
    - Automatic registration of newly available resources: Design a wizard for data providers that facilitates automatic registration of their resources at the DAS
    - Functionalities that enable easy and efficient requesting of schema specifications
    - Application-specific transformations that transform data or data formats for specific needs of ViroLab applications

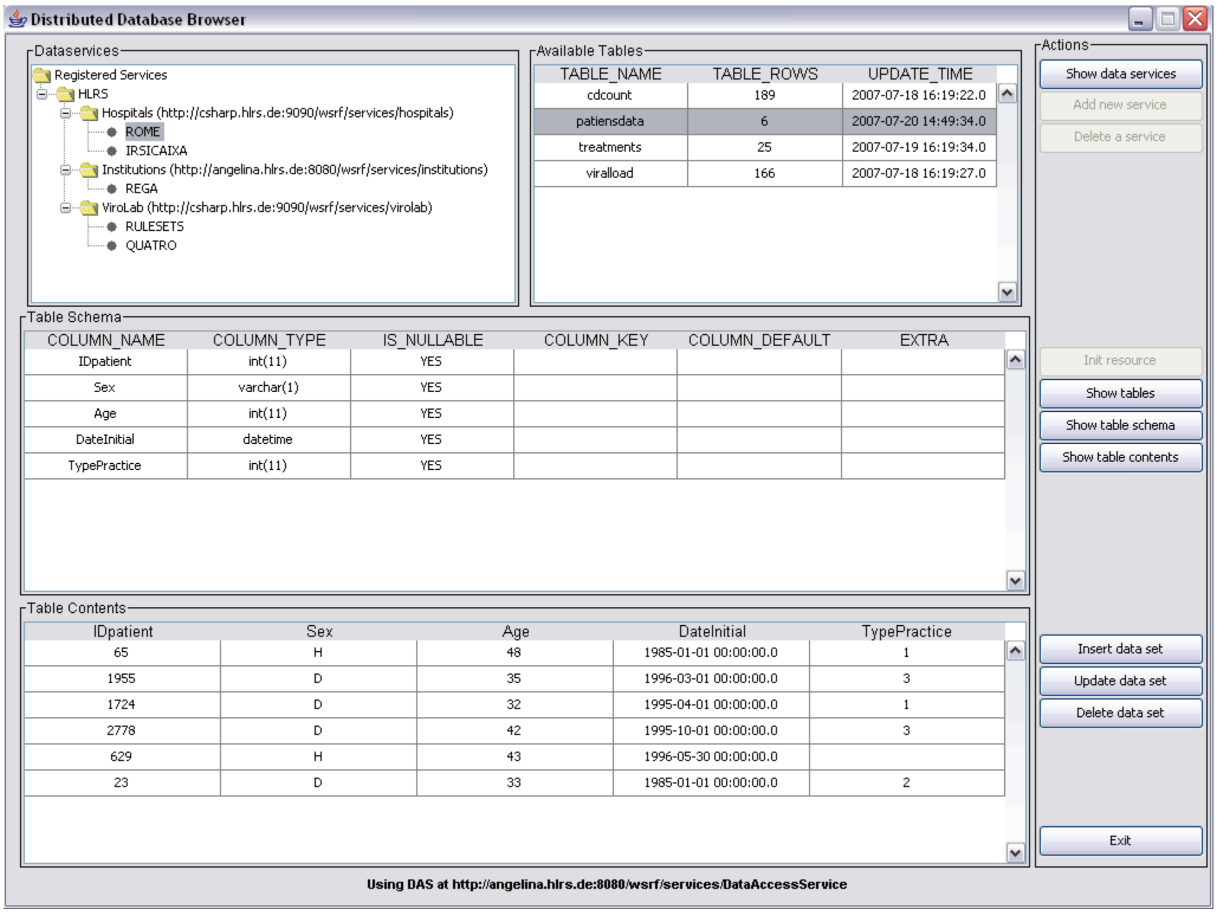
2. Current limitations
    - Submission of distributed queries:
        - All tables must have the same schema
        - The database technologies used should be the same
        - The keywords identifying the content of a data resource must be equal to the database table names (requirement for OGSA-DAI only)
        - The results contain a new primary column: the resource ID
    - Schema retrieval: Schema specifications can only be requested using corresponding SQL statements like *Describe table* for MySQL databases

- Authorization: Authorisation is more or less static, granting either full access or no access to all service methods (access denied!)
- Query language: DAS interfaces must be queried using concrete SQL statements instead of an abstract query language
- Application data storage: The current implementation is limited to one specific ViroLab application – Drug Ranking System (DRS) – but further applications will follow

3. INTERFACE REFERENCE GUIDE

Since the current release of the DAS provides standard web service interfaces, one typically needs a client to interactively test its functionality. Therefore, a distributed database browser was implemented, which allows users not only to browse the resource's content like its schema and data but also to manipulate its entries.

The GUI of the 'Distributed Database Browser', which can be seen in Fig. 3.1, is a simple but very helpful application to quickly overview available resources and their contents. It presents on one screen all information about the selected resource including available tables, table schemes, and table contents. One can perform specific selections on the tables and if a user has permission to insert, update, or delete data sets, this can also be directly done via this application.



- Registered Services
  - HLRS
    - Hospitals (http://csharp.hlrs.de:9090/wsrf/services/hospitals)
      - ROME
      - IRSICAIXA
    - Institutions (http://angelina.hlrs.de:8080/wsrf/services/institutions)
      - REGA
    - ViroLab (http://csharp.hlrs.de:9090/wsrf/services/virolab)
      - RULESETS
      - QUATRO

TABLE_NAME	TABLE_ROWS	UPDATE_TIME
cdcount	189	2007-07-18 16:19:22.0
patiensdata	6	2007-07-20 14:49:34.0
treatments	25	2007-07-19 16:19:34.0
viralload	166	2007-07-18 16:19:27.0

COLUMN_NAME	COLUMN_TYPE	IS_NULLABLE	COLUMN_KEY	COLUMN_DEFAULT	EXTRA
IDpatient	int(11)	YES			
Sex	varchar(1)	YES			
Age	int(11)	YES			
DateInitial	datetime	YES			
TypePractice	int(11)	YES			

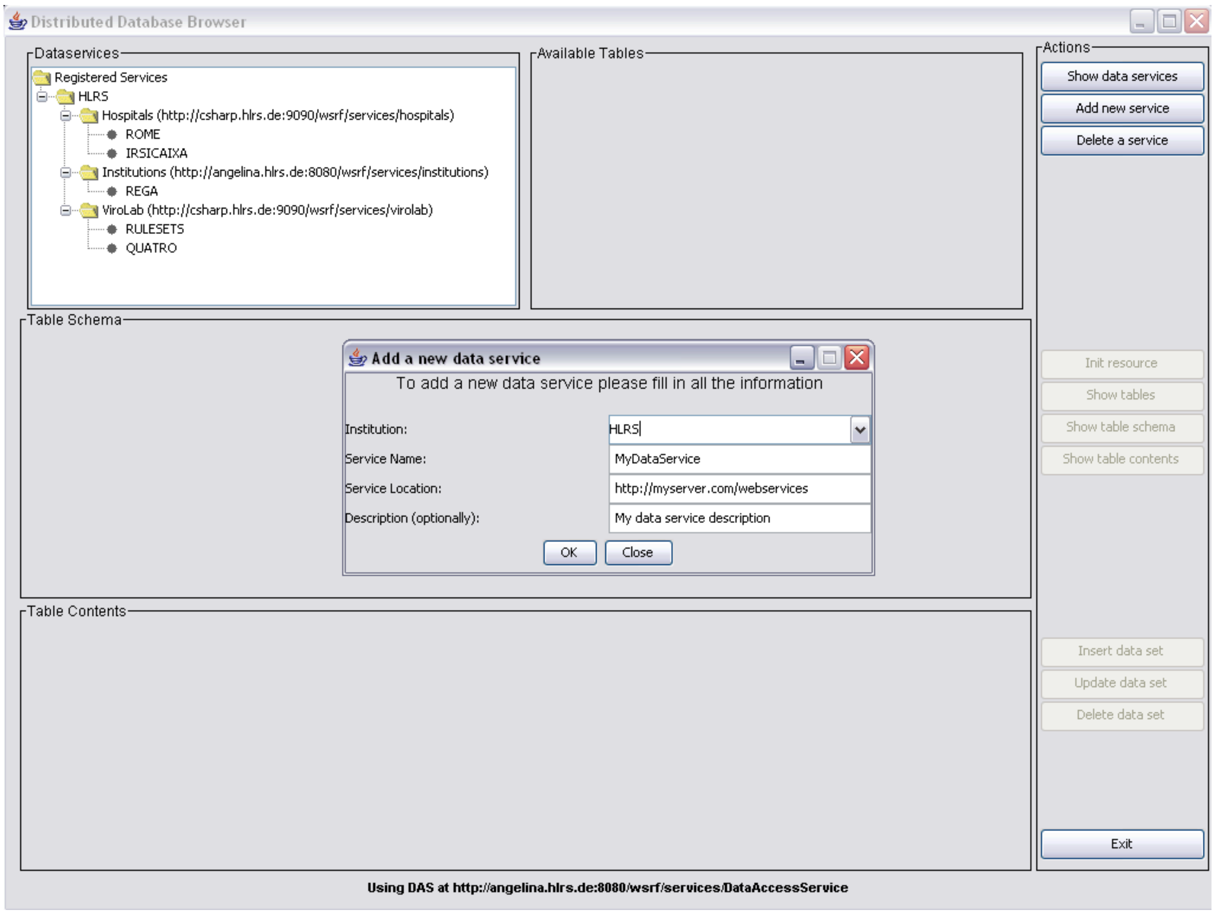
IDpatient	Sex	Age	DateInitial	TypePractice
65	H	48	1985-01-01 00:00:00.0	1
1955	D	35	1996-03-01 00:00:00.0	3
1724	D	32	1995-04-01 00:00:00.0	1
2778	D	42	1995-10-01 00:00:00.0	3
629	H	43	1996-05-30 00:00:00.0	
23	D	33	1985-01-01 00:00:00.0	2

Using DAS at http://angelina.hlrs.de:8080/wsrf/services/DataAccessService

Figure 3.1: The main 'Distributed Database Browser' window

In order to facilitate the work with the application, the main functionalities shall be roughly described:

- *Show data services*: Visualizes all available data services and their corresponding resources based on the information in the central data service repository (see upper left part)
- *Add new service*: Registers a new data service in the central repository used by the DAS – see Fig. 3.2
- *Delete a service*: Deletes a data service from the central repository



Registered Services
- HLRS
  - Hospitals (http://csharp.hlrs.de:9090/wsrf/services/hospitals)
    - ROME
    - IRSICAIXA
  - Institutions (http://angelina.hlrs.de:8080/wsrf/services/institutions)
    - REGA
  - ViroLab (http://csharp.hlrs.de:9090/wsrf/services/virolab)
    - RULESETS
    - QUATRO

To add a new data service please fill in all the information

Institution:	HLRS
Service Name:	MyDataService
Service Location:	http://myserver.com/webservices
Description (optionally):	My data service description

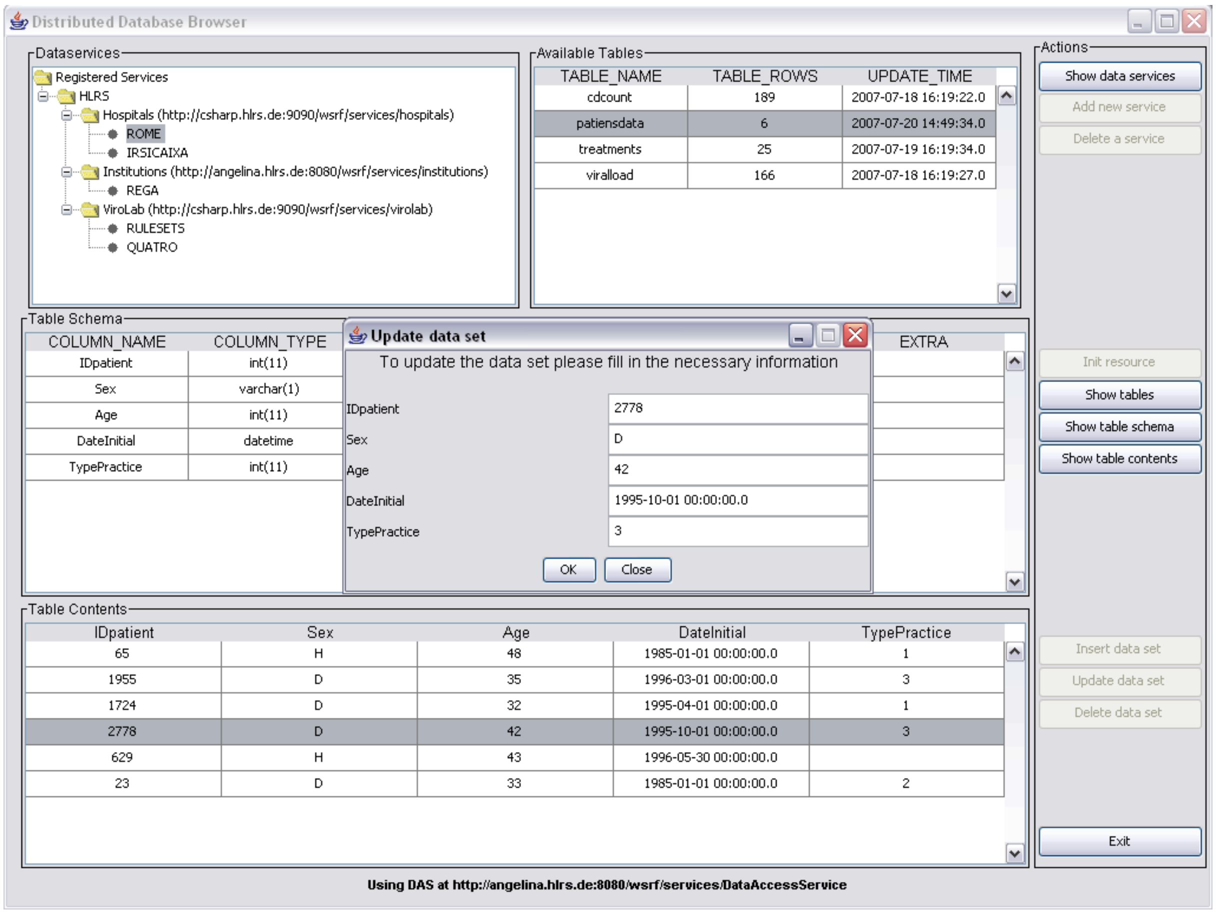
Using DAS at http://angelina.hlrs.de:8080/wsrf/services/DataAccessService

Figure 3.2: Pop-up window for adding a new data service instance

- *Init resource*: Initializes the selected resource according to the service URL provided
- *Show tables*: Displays the available tables including the number of rows and the latest update time (upper right part)
- *Show table schema*: Displays the table schema (central part).
- *Show table contents*: Displays the table contents based on the query provided (lower central part)
- *Insert data set*: Adds a new data set to the current table (if permitted)
- *Update data set*: Updates a selected data set (if permitted) – see Fig. 3.3
- *Delete data set*: Deletes a selected data set (if permitted)
- *Exit*: Closes the application

The application is currently in an early stage of development so that not all planned functionalities of the DAS have been implemented so far. A new prototype is considered for the end of 2007, which then will include also the capabilities of submitting queries to multiple resources at the same time.

In parallel, a second user interface is developed that almost provides the same functionalities but implemented as a portlet for the project portal, which is based on the Google Web Toolkit (GWT) instead of being a stand-alone Java application.



Registered Services
- HLRS
  - Hospitals (http://csharp.hlrs.de:9090/wsrf/services/hospitals)
    - ROME
    - IRSICAIXA
  - Institutions (http://angelina.hlrs.de:8080/wsrf/services/institutions)
    - REGA
  - ViroLab (http://csharp.hlrs.de:9090/wsrf/services/virolab)
    - RULESETS
    - QUATRO

TABLE_NAME	TABLE_ROWS	UPDATE_TIME
cdcount	189	2007-07-18 16:19:22.0
patiensdata	6	2007-07-20 14:49:34.0
treatments	25	2007-07-19 16:19:34.0
viralload	166	2007-07-18 16:19:27.0

COLUMN_NAME	COLUMN_TYPE	EXTRA
IDpatient	int(11)	
Sex	varchar(1)	
Age	int(11)	
DateInitial	datetime	
TypePractice	int(11)	

To update the data set please fill in the necessary information

IDpatient	2778
Sex	D
Age	42
DateInitial	1995-10-01 00:00:00.0
TypePractice	3

IDpatient	Sex	Age	DateInitial	TypePractice
65	H	48	1985-01-01 00:00:00.0	1
1955	D	35	1996-03-01 00:00:00.0	3
1724	D	32	1995-04-01 00:00:00.0	1
2778	D	42	1995-10-01 00:00:00.0	3
629	H	43	1996-05-30 00:00:00.0	
23	D	33	1985-01-01 00:00:00.0	2

Using DAS at http://angelina.hlrs.de:8080/wsrf/services/DataAccessService

Figure 3.3: Pop-up window for updating a selected data set

4. TROUBLESHOOTING Q&A

Q: Upon submitting a DAS query, the system responds with an error saying that it is unable to connect to the DAS (*ConnectException – connection refused*).

A: The DAS may be down for maintenance or you may be experiencing network problems. If the situation persists, please contact *Matthias Assel (assel@hlrs.de)*.

Q: The Axis engine reports an error that it could not find the target service to invoke.

A: Check the service URL you are using. This error typically indicates a malformed or incorrect URL.

Q: The DAS complains with an exception that the *'Passed argument cannot be null'* (*IllegalArgumentException*).

A: While calling any of the interfaces provided, please make sure that none of the arguments passed is null.

Q: The DAS complains with an exception that the *'Passed argument cannot be an empty string'* (*IllegalArgumentException*).

A: While calling any of the interfaces provided, please make sure that none of the string arguments passed is empty.

Q: The DAS answers with the exception *'You do not have the permission to perform this action on the resource'* (*DASException*).

A: You are actually not authorized to perform the chosen action on the corresponding resource. This may have two reasons. Firstly, the data provider has denied access to his resources for particular users due to some personal decisions (-> contact the responsible data provider), or secondly your institution is not well prepared for the ViroLab security infrastructure (-> contact your local responsibility).

Q: While executing a query on a particular resource, the DAS throws an exception saying that it is *'Unable to perfom the query on the resource'* (*DASException*).

A: This error might occur basically if a query is incorrect or malformed and cannot be interpreted by the corresponding data resource technology. Please check your statement carefully and try to resubmit the query. In case your queries are correct but you are still getting the error, please contact *Matthias Assel (assel@hlrs.de)* who will be able to check the logs for further information.

5. IMPLEMENTATION STRUCTURE

5.1. PRODUCT USE CASES

The DAS are designed as a set of services that virtualizes different underlying resources so that users and applications can access them in a transparent way. Figure 5.1 depicts multiple requesters using different services provided by the DAS in order to deal with several databases as if they were one large single data resource.

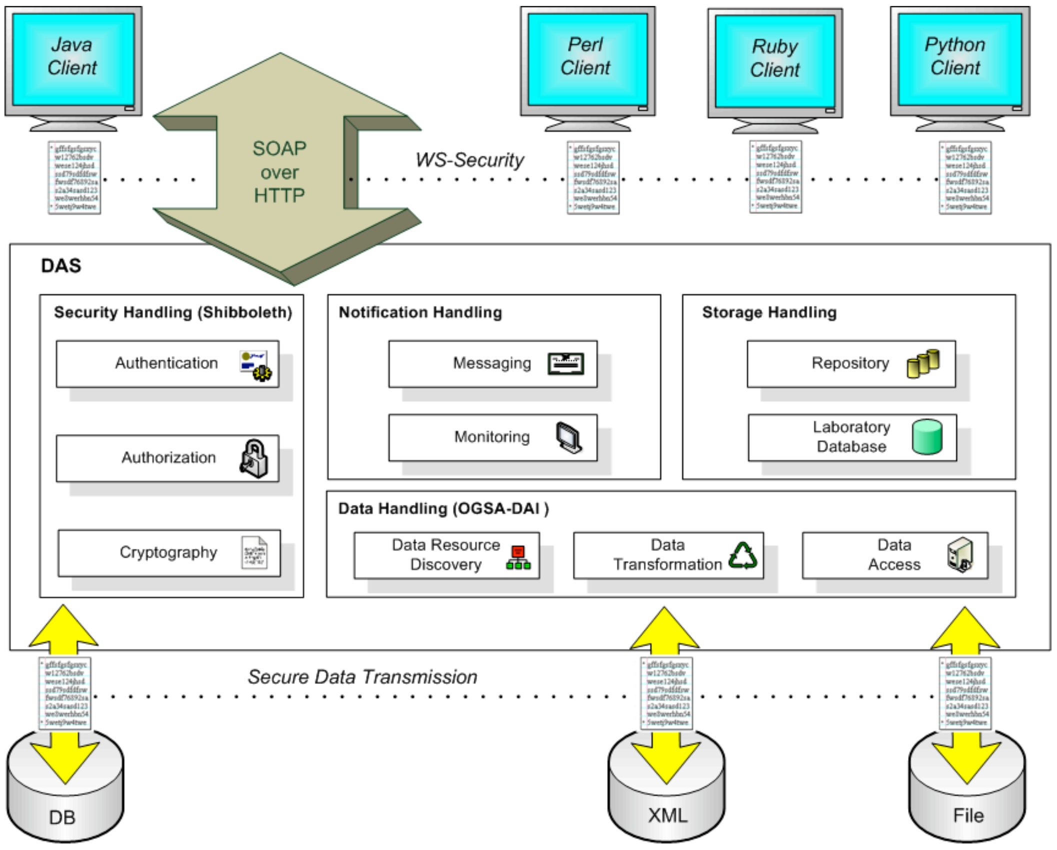


Figure 5.1: General architectural overview of the DAS

A typical use case explaining the core functionalities of the services is shown in figure 5.2. Each single step - starting with the user's request up to the response sent back by the DAS - is highlighted within this chain by one specific block. Basically, all the components play a particular role within this workflow and for each of them different interfaces needs to be provided and implemented. For pure data access, the functionalities provided by the OGSA-DAI toolkit are sufficient, but additional effort on security, mapping of user statements into resource-dependent statements as well as transforming query results into application-readable statements, is needed.

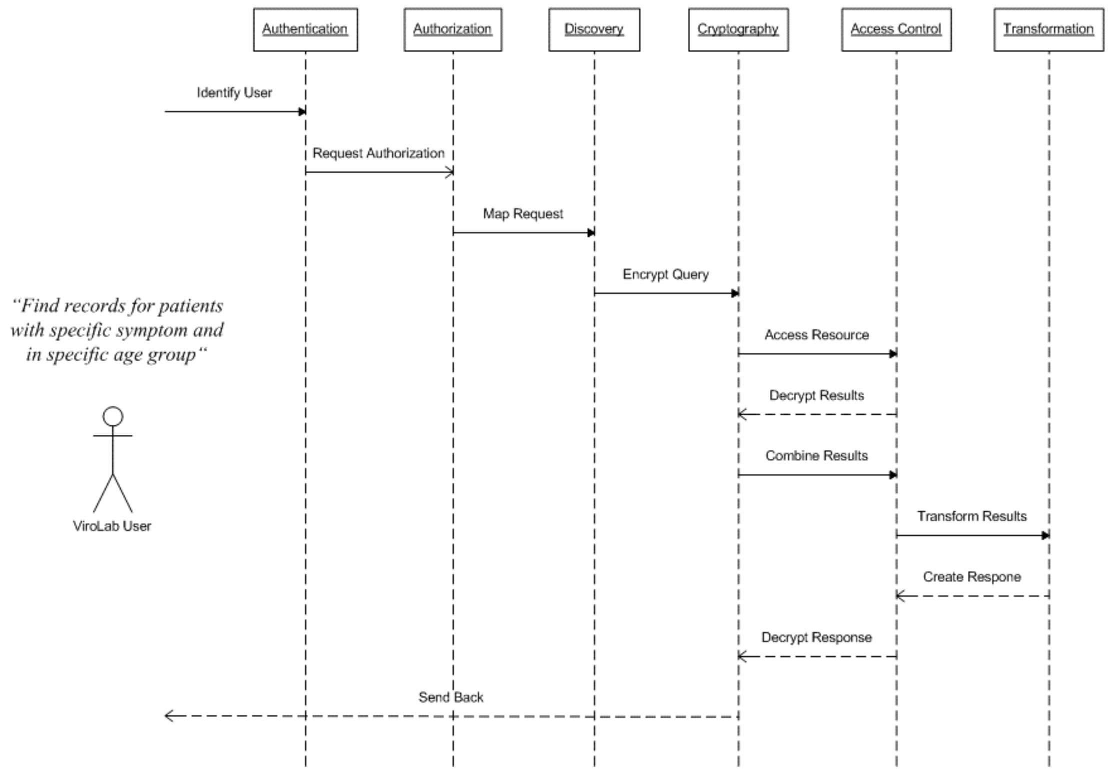


Figure 5.2: A typical use case within the ViroLab scenario

5.2. PRODUCT COMPONENT MODEL

An overview on the main components of the DAS and their dependencies among each other is visualized in figure 5.3.

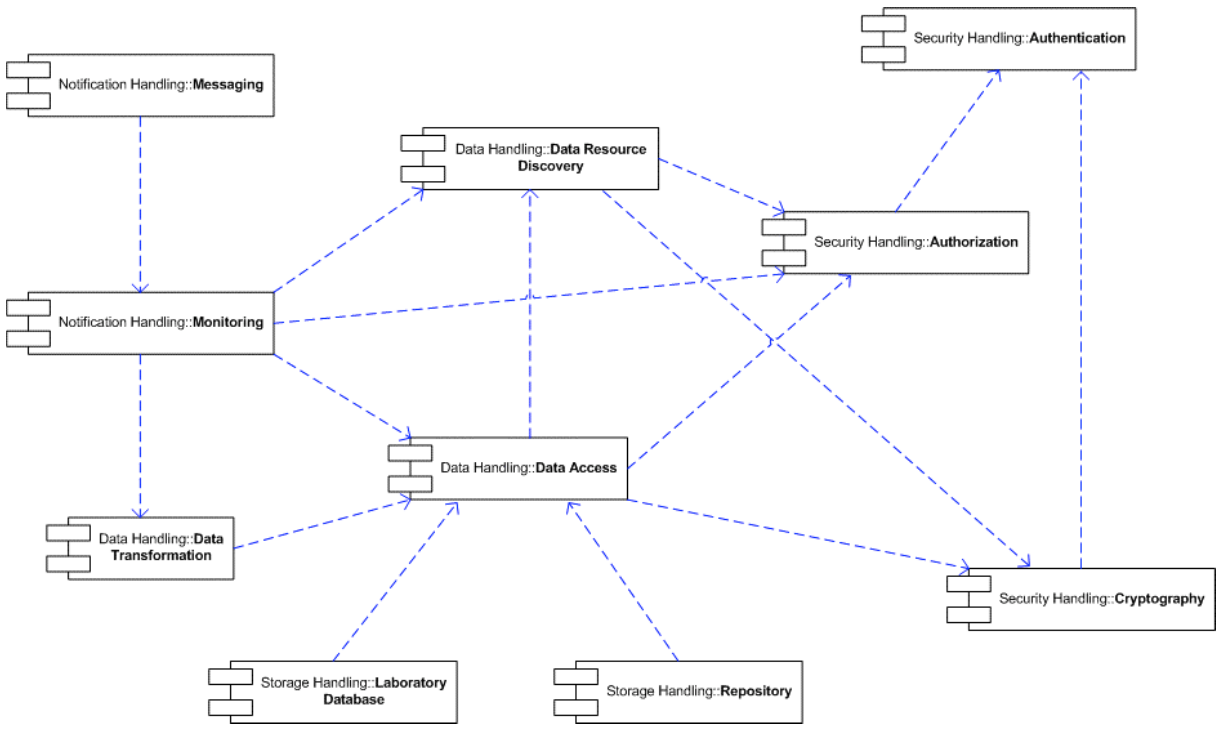

Notification Handling::Messaging
Security Handling::Authentication
Data Handling::Data Resource Discovery
Security Handling::Authorization
Notification Handling::Monitoring
Data Handling::Data Access
Data Handling::Data Transformation
Security Handling::Cryptography
Storage Handling::Laboratory Database
Storage Handling::Repository

Figure 5.3: Main components of DAS

- *Authentication*: The authentication module is responsible for the identification of a user based on his credentials
- *Authorization*: The authorization interface decides whether a user is authorized to perform a certain task by mapping user attributes on data handling activities and resources
- *Cryptography*: The service provides capabilities for decrypting incoming and encrypting outgoing messages to ensure secure transmission between different endpoints
- *Data Resource Discovery*: The discovery service virtualizes the location of data resources by mapping common language terms onto data resource-dependent statements
- *Data Access*: The data access infrastructure is the most important part of the overall system. It provides interfaces to access different types of resources. Its main functionalities are based on the OGSA-DAI toolkit.
- *Data Transformation*: The transformation service provides methods for dynamically adding new transformation schemes in order to change the output format conceptional for a user application
- *Messaging*: The messaging subcomponent of the notification handling infrastructure contains mechanisms for publishing, subscribing to, and managing subscription to notifications about single events or families of interest
- *Monitoring*: The monitoring service is responsible for recording all transactions that occur inside the data access subsystem
- *Repository*: The repository will be used for storing any kind of intermediate data
- *Laboratory Database*: The laboratory database acts as a long time storage with a relatively short access time and can be used also by other components of the ViroLab infrastructure like for example the Provenance system

5.3. DETAILED IMPLEMENTATION MODEL

Since the DAS is based on an service-oriented architecture and its functionalities are provided as standard web service interfaces, the focus for describing the implementation model lies on one specific service implementation class (*'DataAccessServiceImpl'* – see API description for further details), which includes almost two-thirds of the main interfaces. For more details on the design and implementation of single DAS components, please refer to deliverable D3.3.

The interfaces can be summarized into two main parts. The first part contains standard interfaces for querying remote databases including one specific interface for submitting distributed queries, whereas the second part includes particular methods for requesting publicly available rule sets and for storing application-dependent data.

Part 1: Connecting to and querying from remote databases

Most of the interfaces provided are directly connected with corresponding OGSA-DAI interfaces. Figure 5.4 depicts the specific use case where a user wants to invoke a query using the interfaces of the DAS. On the right side of the picture, the involved components of OGSA-DAI and their interactions are listed. Based on the request, different activities are performed after an authorization mechanism has granted access to them.

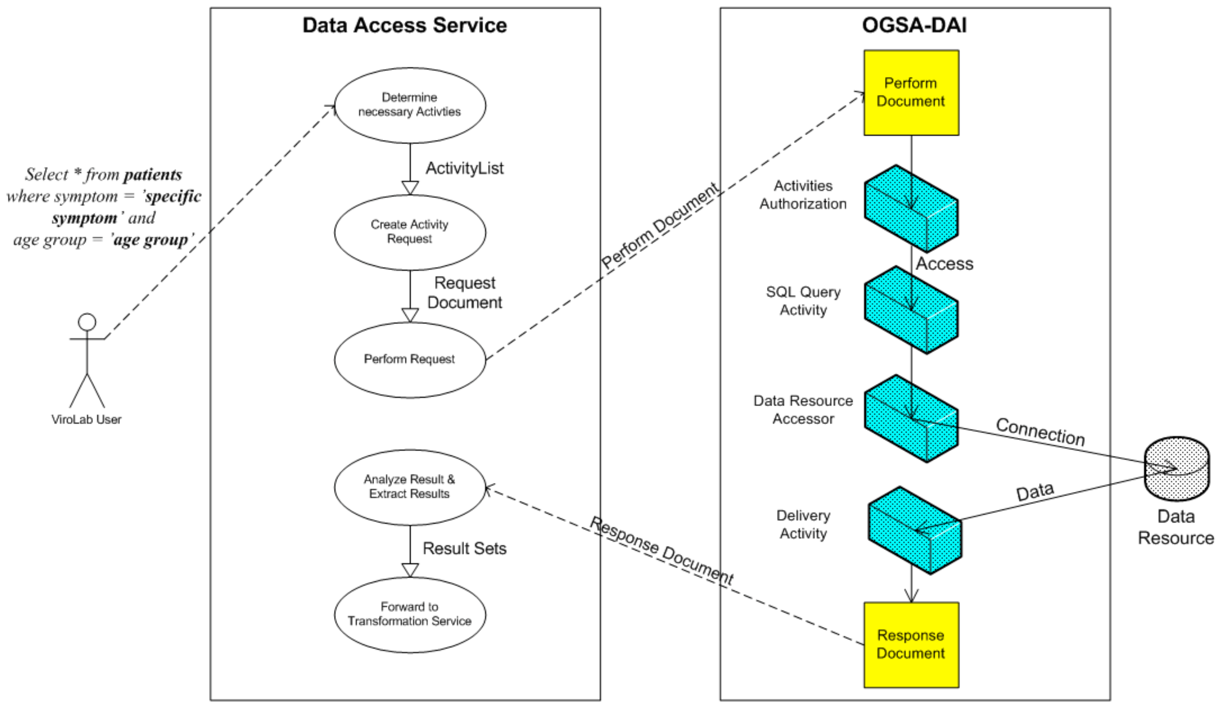


Figure 5.4: Use case showing a typical data access request and corresponding interactions with OGSA-DAI

They are typically connected with one data resource accessor, which uses a database-dependent driver to establish a connection with the underlying resource.

When dealing with multiple data providers, each of them usually has its own installation of a data access system including the data access service linked with an OGSA-DAI data service. The coordination of all these single systems requires one central entry point, which acts as the only "visible" and accessible data access system, and which hides all other data access systems from the users. In theory users should be unaware that they are using a federation rather than a single data resource. Currently, the DAS offers one specific functionality that handles a federated query. The main operations are similar to the one shown in figure 5.4 with the difference that this has to be done many times.

Part 2:

- Requesting publicly available rule sets

The main usage of the *RequestRuleSets* method has already been decribed in section 2.2 so that only the different steps, which are transparent to the users, shall be explained in more detail. The following diagram schematically shows the control flow of the *RequestRuleSets* function in a sequential order.

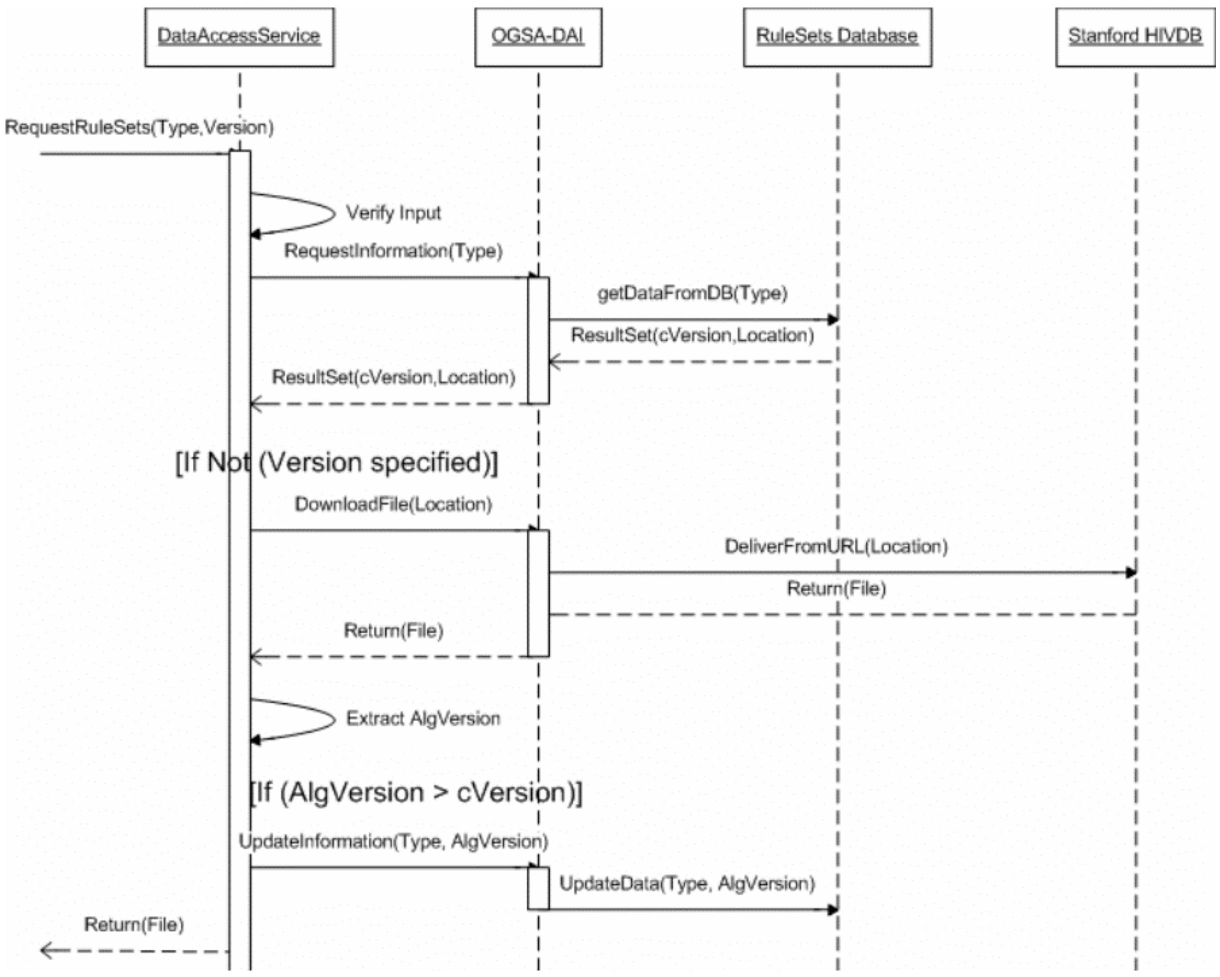


Figure 5.5: Control flow of the specific *RequestRuleSets* method

One can see that the function reverts to OGSA-DAI's data access capabilities, which are used to download files from an online data repository and also for storing relevant information in a local database. This local information is used to manage the current available and used versions of the rule sets. The diagram also illustrates the interactions between the DAS specific components/services and the corresponding interfaces offered by OGSA-DAI.

- Storing application-dependent data

Based on the general description in section 2.2, the following diagram shall explain the internal steps performed during the processing. Depending on the current application type, a particular internal application-related function is called, which performs relevant data transformation on the values provided. Once these data manipulations are finished, the OGSA-DAI functionalities for accessing distributed databases are used by the method in order to store the values in the corresponding database.

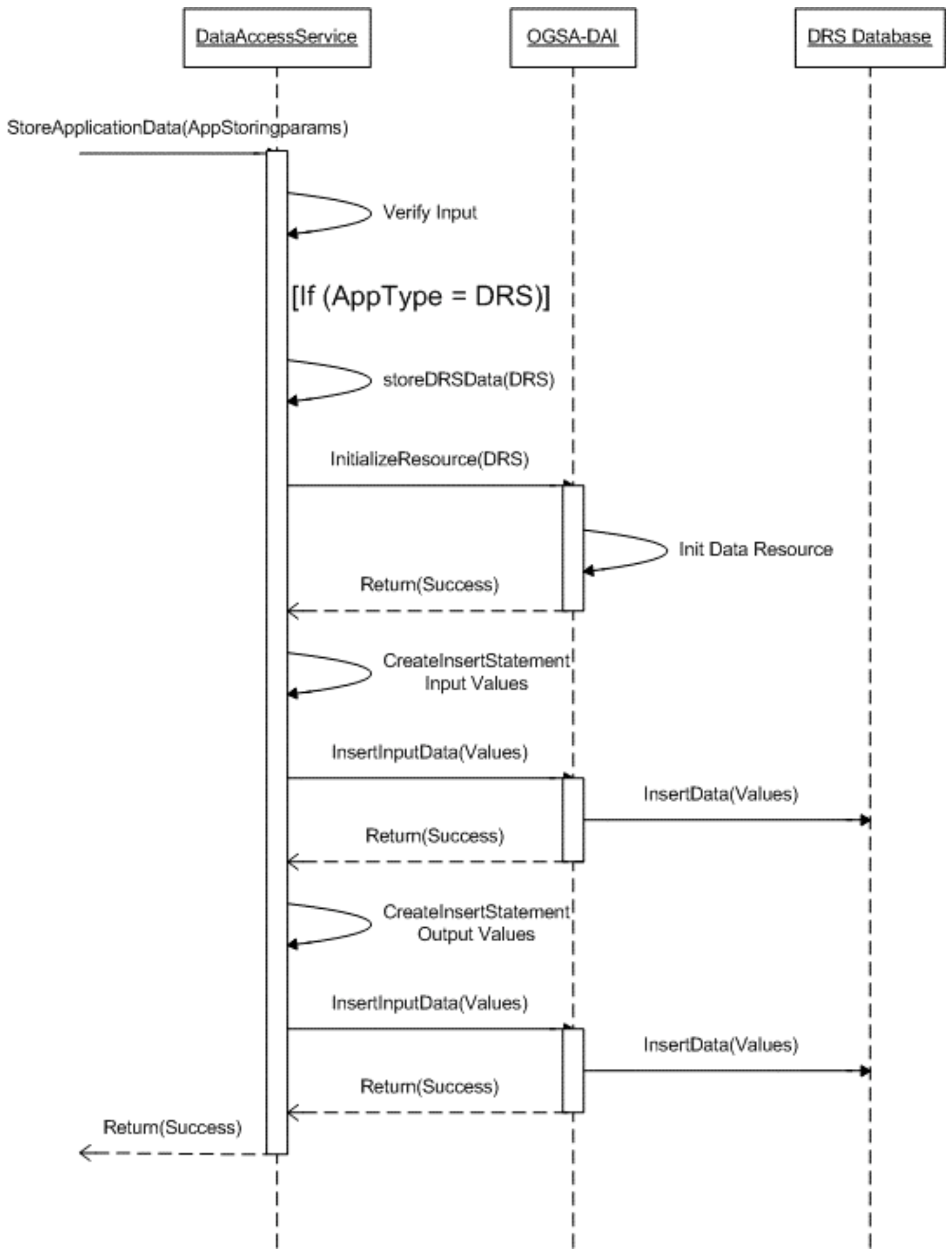


Figure 5.6: Internal flow of *StoreApplicationData* method

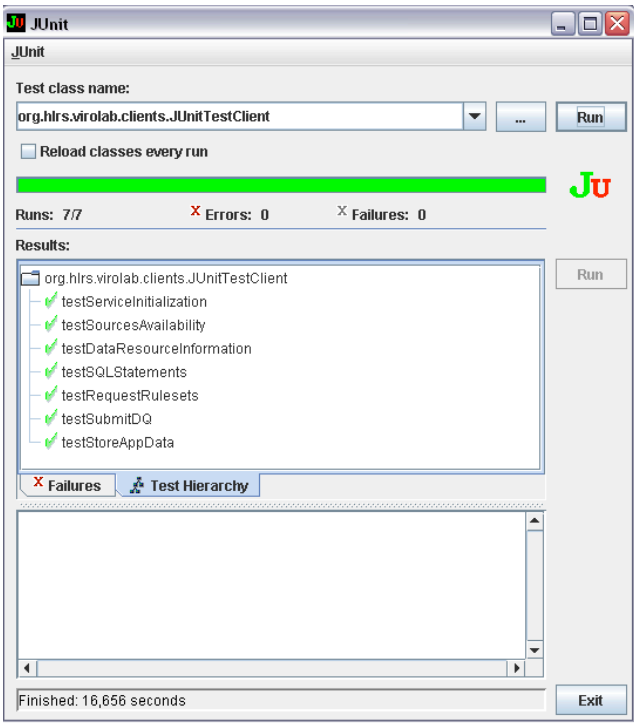
6. PRODUCT TESTING

According to Work Package 4 integration guidelines (specified in D4.2), each component itself should perform a unit testing procedure for its functions and methods. For verifying reliability, efficiency, compatibility, integrity, and usability of the DAS functionalities, a set of unit test cases were written and used to check the source code including different functions of the API:

- *TestServiceInitialization*: Initializes a particular service and its corresponding resource
- *TestSourcesAvailability*: Checks the availability of particular resources
- *TestDataResourceInformation*: Collects the meta-data of the resource
- *TestSQLStatements*: Performs different queries and verifies the data retrieved
- *TestRequestRulesets*: Checks whether different types of rule sets can be requested
- *TestSubmitDQ*: Tests the submission of queries to multiple data resources
- *TestStoreAppData*: Tests the storage of application data (Currently only for the DRS application)

The GUI of the JUnit toolkit, which is shown in figure 6.1, can be used to visualize the testing procedure and to facilitate the testing process. One can simply load the test class and start the procedure by clicking the 'run' button.

The screenshot below depicts the successful test of the above explained testing routines.



Test class name: org.hlrs.virolab.clients.JUnitTestClient

Runs: 7/7   Errors: 0   Failures: 0

```
org.hlrs.virolab.clients.JUnitTestClient
  testServiceInitialization
  testSourcesAvailability
  testDataResourceInformation
  testSQLStatements
  testRequestRulesets
  testSubmitDQ
  testStoreAppData
```

Finished: 16,656 seconds

Figure 6.1: Visualization of DAS unit test cases

7. CONTACT INFORMATION AND CREDITS

For additional information, questions, errors, bugs etc. please contact the following author:

- Matthias Assel (assel@hlrs.de)

The author also wants to thank all who contributed to this work, in particular:

- Aenne Löhden (USTUTT, Stuttgart, Germany)
- Bettina Krammer (USTUTT, Stuttgart, Germany)
- Stefan Wesner (USTUTT, Stuttgart, Germany)
- Piotr Nowakowski (ACK Cyfronet AGH, Kraków, Poland)

8. ABBREVIATIONS

Abbreviation/Term	**Explanation**
API	Application Programming Interface
CA	Certificate Authority
DAS	Data Access Services
DRS	Drug Ranking System
GPL	GNU General Public License
GT4	Globus Toolkit 4.0
GUI	Graphical User Interface
jar	Java Archive
OGSA-DAI	Open Grid Services Architecture Data Access and Integration
PDP	Policy Decision Point
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVN	Subversion, a revision control system
UML	Unified Modelling Language
URL	Uniform (or Universal) Resource Locator
ViroLab	A virtual laboratory for decision support in HIV treatment
WSDL	Web Services Description Language
XML	Extensible Markup Language