CC_Client/Server Communication

Revision history

2018-09-05	Nils Tolleshaug	Original draft

INTRODUCTION

HRE APIs for Client/server communication shall be able to provide four basic types of functionality, namely Create, Read, Update, and Delete within the database. These operations are often referred to with the acronym CRUD. HRE implementation must have the ability to handle these four functions but in addition HRE will also have database administration and file transfer functionalities that are not CRUD operations.

To implement a Client/Server CRUD web service, there are two protocol choices, alternatively to implement according to REST or SOAP. REST and SOAP are briefly explained below:

REST – (Representational State Transfer) is an architectural style (CRUD) that defines a set of constraints to be used for creating web services. Web Services that conform to the REST architectural style, or RESTful web services, provide interoperability between computer systems on the Internet. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations.

Typical for REST: Based on HTTP, Implements CRUD on top of HTTP, is simple and flexible to use, supports other transfers of data in formats such as ex JSON or plain text. REST is advantageous for implementing a Client on a browser as IE, Chrome, Firefox.

The following table is from: https://en.wikipedia.org/wiki/Representational_state_transfer

Uniform Resource Locator (URL)	GET	PUT	PATCH	POST	DELETE
Collection, such as https://api.example.com/resources/	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	Not generally used	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
Element, such as https://api.example.com/resources/item 17	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	**Update** the addressed member of the collection.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry within it.	**Delete** the addressed member of the collection.

In REST the type of HTTP operation defines the action and URL string refers to the resource to be accessed in the database. Usually URL in HTTP refers to a piece of code on the server that perform the operation.

SOAP (originally Simple Object Access Protocol) is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks. Its purpose is to induce extensibility, neutrality and independence. It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

SOAP allows processes running on disparate operating systems (such as Windows and Linux) to communicate using Extensible Markup Language (XML). Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.

Typical for SOAP: XML based only, more complex protocol, supports transfer over other protocols than HTTP, older than REST.

For Client/Server communication in HRE the CRUD request must be sent either as an HTTP REST or a SOAP request from the client Request Manager to the Request Listener on the server. Request Listener receives the protocol request, basically transfers the request into an SQL query to the SQL database and receives the output of the SQL query. Received data from the SQL database needs to be converted to a serial data stream coded as JSON (Java Script Object Notation) or XML (Extended Markup Language) and sent over the TCP/IP network to the Client Request Manager. The requested data received in serial form from the network will be converted in the Client Request Manager and the received dataset will be delivered as a pointer to a data object structure to the Business Rules Layer in the client.

DESIGN SUMMARY

Please note the Request Manager and Request listener modules are named according to the building blocks described in the HRE design documents *03.02 Architecture Client* and *03.03 Architecture Server*.

For HRE client/server communication the most modern and flexible alternative is to implementation HTTP REST requests and JSON coding of data. HRE cannot strictly follow REST specification as more parameters are necessary to be transferred as part of the request for data. HRE REST implementation needs to be supplied with requests for data that are more complex that CRUD. HRE Client/server communications cannot be restricted to only CRUD operation in the database as HRE implementation also has database administration and file transfer functionality. Client/Server communications according to REST/JSON have also the advantage that clients can be implemented in a browser based on HTML (HTML5) language.

In the HRE implementation the intention is to use a Java implemented Request Manager in the client and a Java implemented Request Listener on the server. The Request Manager in the client has an API to be used for calls from the Business Rules Layer in the client. Request Manager and Request Listener will have two communication setups, namely: LOCAL, if the client and the server are on a single computer (local communication) or REMOTE if the client and the server are on different computers (remote communication). The intention is to have one implementation of the HRE client that can be used both in the remote and the local case.

The Request Manager will, in the remote communication case, select the Remote Connection Requester that converts the request and sends an HTTP REST request over TCP/IP network to the Remote Connection Listener that passes the request to the Request Listener on the server. The Request Listener will pass the request to the Server Business Layer and receive the SQL questionnaire data response. The Remote

Connection Listener converts the SQL questionnaire response to a JSON serial data stream and sends the stream over the TCP/IP network to the Remote Connection Requester in the client. The Remote Connection Requester will parse the JSON serial data stream and create a JSON-Object data structure and deliver a JSON-Object pointer to the Request Manager. The Client Business Rules Layer has access to the JSON-Object data structure through this pointer.

In the local communication case, the client Business Rules layer sends a request over the Request Manager API interface to the Local Request Manager which is directly connected to the Local Request Listener on the same computer *(Local Request Manager and Local Request Listener are in some HRE design documents called Bridge Communication).* The Local Request Listener passes the call to the Request Listener that passes the request to the Server Business Layer. The Server Request Listener receives the response in the form of a SQL questionnaire data output. The combined Local Request Listener and Local Request Manager converts the data to a JSON-Object data structure and passes the pointer to the JSON-Object data structure over to the client Request Manager. The Client Business Rules Layer receives a pointer to have access to the JSON-Object data structure.

The Client/Server communication shall offer parallel communication of data which implies that the Server Request Manager and partly the Client Request Manger and other class implementations of communication processes must be to running as Threads (implements Runnable). Please note that Java SWING is not Thread safe and therefore “parallel” processing of GUI processes should be avoided. Database requests (database request APIs) in the Request Handler can be implemented as pure Java class method calls. The objective is to implement the Client Request Manager and Server Request Listener all in Java, however an Apache HTTP processor can be an alternative for Remote Request Listener on the server.

The client Request Manager shall have cache storage to store pointers to data retrieved from the database to avoid retransmitting of data requested from the SQL database. Client Request Manager needs to store pointers for all the received JSON-Object data structures. To reduce the load on the server and the Internet communication, each SQL request will ask for more data than needed for the current view in the GUI panel. This will allow the user to select a subset of data to be presented in the GUI panel without retransmitting data from the server. A typical situation is that the client Request Manager receives information about many hundred persons however only a few persons are presented in the visual GUI table of persons.

SCOPE

This document describe the design of a Java implemented Client Request Manager that handles communication requests from the Client Business Layer to the Server Request Listener and the transfer of data returned to the Business Layer in the client.

The communication shall be able to handle STATUS, DATABASE and BATCH requests as specified in *03.02 Architecture - Client Component* and *03.03 Architecture – Server Component*.

STATUS communication process is a command and status report channel between Request Manager in the client and the Request Listener on the server. The channel will exist as long as the user communicates with the HRE implementation. STATUS will have a regular “handshake” to monitor the status of the server communication. This channel is established when the client initiates the server communication and sends userID and password for user authentication.

DATABASE – to be used by GUI SWING components that are not Thread safe. GUI process must wait for response before continuing to process and present data retrieved from database. Only the listener process in the Server needs to be running as a Thread.

BATCH – Transfer of a data file in both directions between Client and Server. Both Client and Server processes need to be running as Threads.

ACTIONS

Client Request Manager shall handle following communication to/from the Server Request Listener:

1. Connect with the Server and establish the STATUS communication channel
2. User authentication
3. Query the Server Status and send commands to/from Client and Server over STATUS channel
4. Large database intensive processes can be initiated over STATUS channel
5. Disconnect from Server and terminate STATUS communication
6. Transfer DATABASE requests to update the database – transfer and receive strings
7. Transfer DATABASE requests to retrieve data from the database – transfer and receive strings
8. BATCH File transfer process that transmits files in both directions between the Client and Server. The file transfers should use background asynchronous FTP transfers
9. BATCH processes can only be initiated by the GUI and read and store files in the file system.

USED BY

Business Rules in Client will in the DATABASE case prepare query data String to the database and interpret the received response data Stream or character String. See APPLICATION PROGRAMMING INTERFACES (API).

STATUS, DATABASE and BATCH processes in the Server will receive data from Server Request Listener and respond with data codes as a Java String or a reference to Java ReadStream.

AUXILIARY DATA

HRE project should prepare the definition of a basis communication handshake (protocol) over the STATUS channel.

SSL or TLS certificate for encrypted HTTPS communications over IP network.

REQUIRED SERVICES

Charset handler – Java uses charset according to national language while Linux servers use UTF-8.

APPLICATION PROGRAMMING INTERFACES (API)

Example of DATABASE API

Client API for BUSINESS RULES database request to Request Manager: -

```
JSONObject responseData = ClientRequestManager.requestDatabaseData(String queryData)  {
        // Receive Send String with query data from BUSINESS layer to server Request Listener
        // Wait for pointer to responseData to be returned to BUSINESS layer
}
```

API for Server Request Listener to request data to SQL database: -

```
String[] responseData = DatabaseHandler.requestDatabaseData(String queryData) {
        // Send String queryData to DATABASE.
        // Wait for responseData to be returned from DATABASE
}
```

Alternatively implementation can include API's that deliver a ReadStream to the BusinessLayer process. In BusinessLayer the ReadStream is used for reading and converting the character stream to an HRE specific object oriented tree data structure if the datastream cannot be coded in JSON format.

Server Request Listener has a HTTPS (secure) server interface that responds to HTTP(S). Apache can be configured to operate as a Remote Server Listener. The intention is that the implementation shall be in accordance with the **Representational state transfer (REST)** specification. REST defines GET , PUT, PATCH , POST and DELETE operations over HTTP(S). The intention is also that the returned data stream will be JSON coded. This REST/JSON interface is used for Client/Server handling of data and will be convenient for communication between HRE database and phones/tablets.

Further API specifications of STATUS, DATABASE and BATCH communication are needed.

WARNING CONDITIONS

The Client Request Manager and server Request Listener must communicate to inform the user of certain problems in the communication. The client Request Manager can throw an HREexception that is picked up by the Business Layer processes. The Server Request Listener uses the STATUS channel to communicate failures over to the STATUS process in client Request Manager which throw an HREexception. Typical examples of reported error conditions are:

- STATUS communication failed between Client and Server
- Server does not respond to datarequest
- Server failure.

ERROR CONDITIONS

User Error messages should be as understandable as possible for ordinary PC users. Error messages should refer to the superior process that failed rather than referring to the name or abbreviation of the software component with the problem.

SPECIAL ISSUES

SEPARATE SERVER COMMUNICATION

HTTPS secure HTTP shall be used for remote server communication.

User authentication is needed for remote Client/Server communication to separate server. The setup can use an API to a LDAP (Lightweight Directory Access Protocol) service on the server or to a LDAP on another server. The authentication process will provide a session ID for each user session.

The Client/Server communication for each client need a Session ID identifying each Client/Server session. Using HTTP REST the Session ID must always be part of the data request to identify the user for Client/Server access.