Client/Server scenarios – local, remote and multiuser

Revision history

2019-01-04	Nils Tolleshaug	CC_Client/Server scenarios – local and remote
2019-01-05	Don Ferguson	Standardised format, tidied language and typos etc
2019-01-07	Robin Lamacraft	Comments re caching and multi-project operations
2019-01-15	Nils Tolleshaug	Included Robin's comment into document

CONTENT

INTRODUCTION 1

SCOPE 1

1.0 SCENARIOS 2

1.1 Locally stored projects 2

1.2 Remote server project 2

1.3 Multi user access to a project 3

2.0 FUNCTIONALITY 4

2.1 Project administration 4

2.2 CRUD project operations (Create/Read/Update/Delete) 4

2.3 Project split, merge or compare from projects 4

2.4 Caching of project data in client 5

INTRODUCTION

The HRE project was foreseen to be used as a tool in project teams with project members working on the same project. The HRE project plan as covered in *01.06 Implementation Stages 2017-04-19* basically describes an HRE development plan for single user project operation where the user has exclusive access to the project. When introducing clients that can also access and update a combination of projects, stored either locally or remotely, it is necessary to add functionality to the HRE project specifications that cover these more complex scenarios and possible access restrictions in the implementation. One HRE database can contain more than one HRE project and to simplify the terminology the term "project" as used here refers to a project database contained in a larger database stored on a local computer or on a remote server.

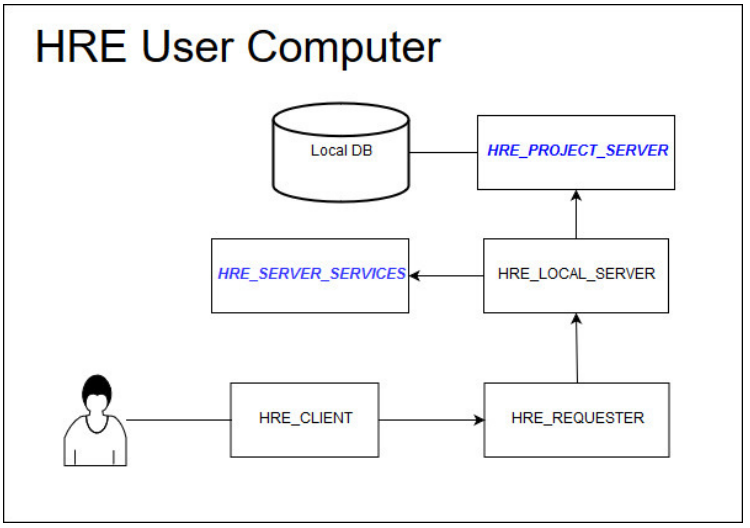
SCOPE

This document describes in the first chapter three HRE Client/Server scenarios - Local project on the same computer as the client, remote projects stored on a remote server and the multi user scenario where more than one client operate on one single database. For each scenario access restrictions to projects are described. The second chapter covers possible functionality that can be introduced to allow multiuser access to projects through semaphores that grant exclusive rights to update a specific record. Also covered is functionality to increase efficiency by caching of data in client in order to reduce the amount of data transferred between client and server.

1.0 SCENARIOS

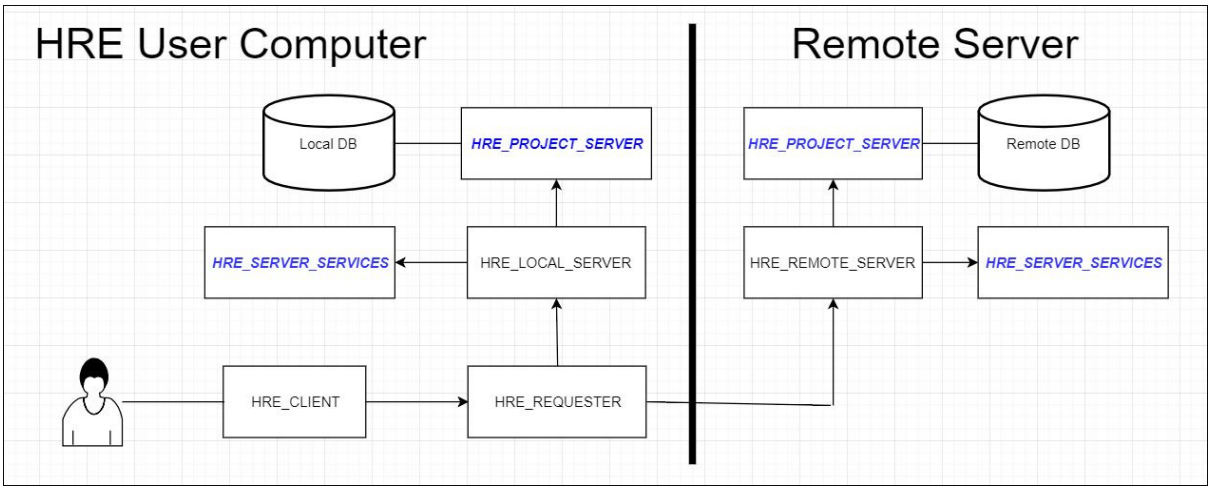
1.1 Locally stored projects

This is a scenario where the client and the projects are located on the same computer without a remote network communication between client and project. This scenario is identical to the setup in the original TMG program. Projects stored are exclusively accessed by the client and no other user can access the project. There can be more than one project in the database. The User can import and export whole projects or parts of projects to be used in other client projects.



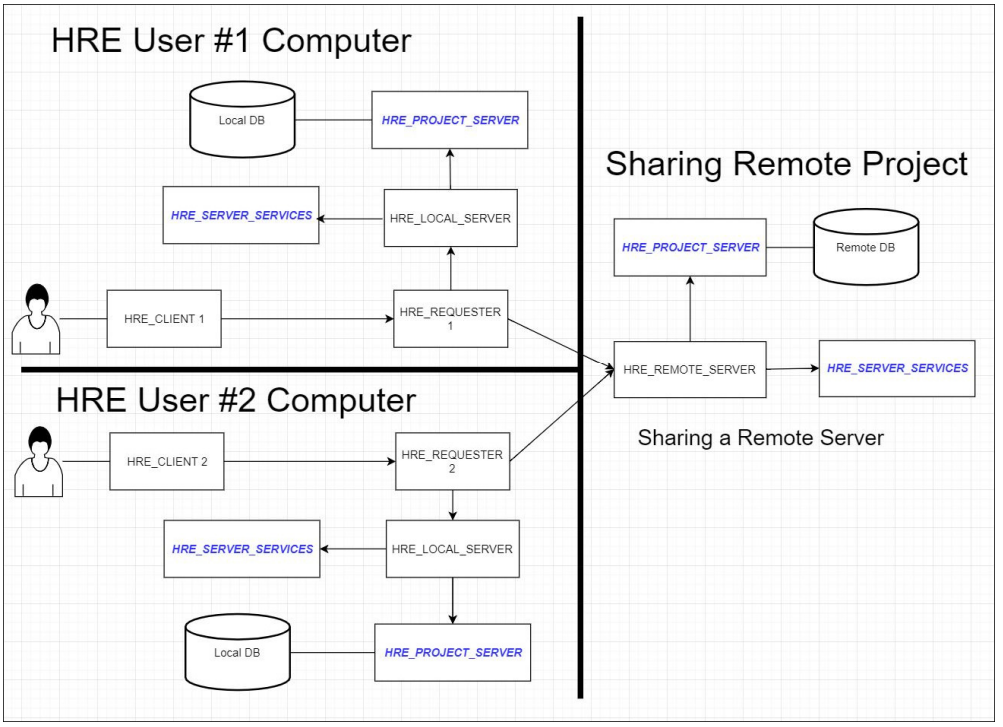
1.2 Remote server project

In this scenario the client can access both local and remotely stored projects. The client has exclusive rights to the projects but can import or export whole project or parts to other users. The remote server can have a database with more than one project however each project is owned by the user client and is exclusively accessed by this user. The local stored projects are according to scenario 1.1 and not accessible over the public network. Because of the interchange of data over a public network, the scenario requires implementation of user access data and user authentication.



1.3 Multi user access to a project

This is the scenario for user shared projects in project teams. The locally stored projects are exclusively owned by the user client according to scenario 1.1 and cannot be accessed from another user. The remote project is the shared project. Because of the interchange of data over a public network, the scenario requires implementation of user access data and user authentication. In this scenario the users can in parallel do CRUD (Create/Read/Update/Delete) operations. To avoid conflicts under project update the scenario needs to have implemented methods that “lock” specific parts of the project when updated by one user.



2.0 FUNCTIONALITY

Parts of the functionality in the block HRE_Server_Services are described here.

2.1 Project administration

Some project operations require exclusive access to one or more projects.

Examples:

- Deleting a project
- Renumbering the visible IDs of persons
- Performing a merge of projects
- Block delete of part of a project.

Some of these will only require a short pause for other users shared access to a project, while others will require a longer time (implying possible termination of connection). In the case of deletion of a project a permanent termination of access by others is required.

2.2 CRUD project operations (Create/Read/Update/Delete)

Operating on single projects with exclusive access needs no restrictions when doing CRUD operations. However in scenario 1.3 with multiuser access, update of records must be controlled by "semaphores" or flags that prohibit possible update by another user if one user is in the process of updating project content. The "semaphores" can be associated with data base records or at a higher level e.g. on an event level.
It is also possible in a project team to give project team members responsibility for parts of the project. This project access data needs to be stored as project access user data. This splitting of project member responsibility can be also agreed as project specific rules that are not part of the HRE implementation.

2.3 Project split, merge or compare from projects

There are some extra cases that need to be described. Typically, these operations would have copy of 2 input projects and possibly a 3rd new result project.

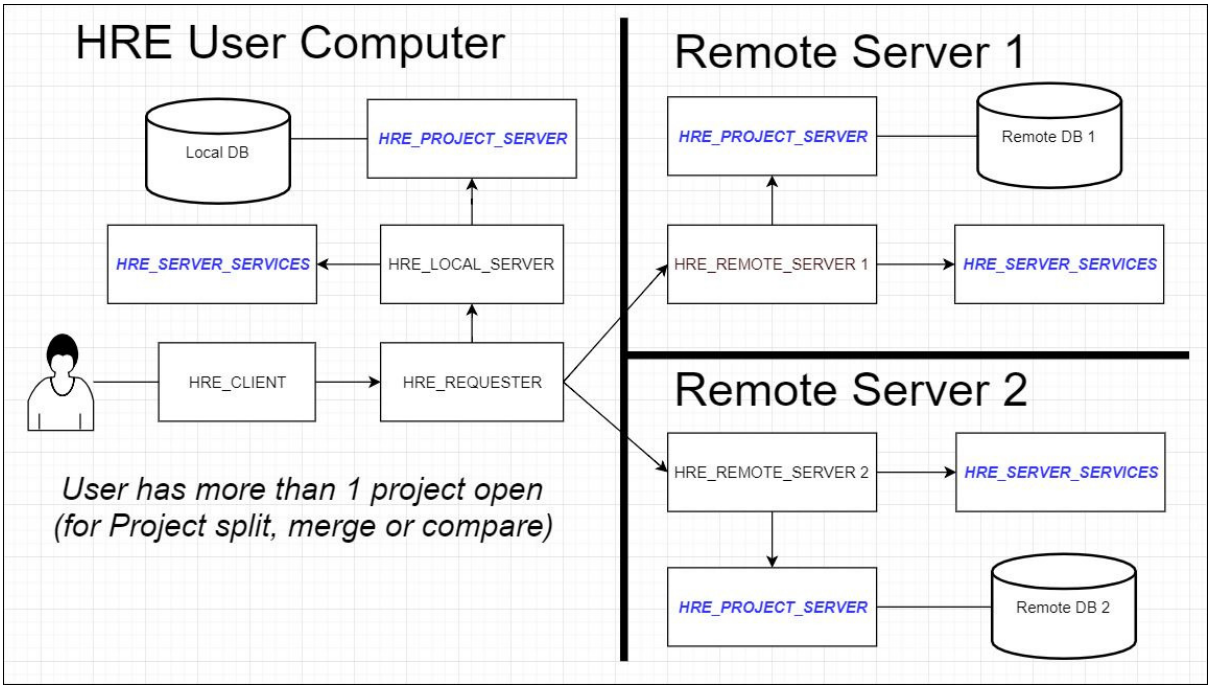
a) Operations conducted entirely within the User Computer on local projects
b) Operations conducted between local and remote databases where the remote databases are on the same remote server
c) Operations conducted between local and remote databases where the remote databases are on different remote servers.

The output database could be located locally or on a remote server of user choice.

For a), b) and c) the source projects will be copies of existing projects, however destination projects need to be updated with exclusive access if other users have rights to update the project.

Some servers may have more capacity to perform multi-project operation. Temporarily moving the required projects to a single server may be advantageous (if feasible) to combine all projects within the same database for the conduct of operations. This concept has some overheads, transmitting large databases to a common server prior to the operations and then again back to their location after the post the operations are completed. It is likely that there are 3 styles of multi-project operations:

1. to progressively grow a main HRE project by merging a small one into the larger project
2. to compare 2 projects
3. to select a subset to create a new project.



2.4 Caching of project data in client

Cashing of project data in client will be uses for scenario 1.1, 1.2 and 1.3. Cashing is primarily introduced to reduce server load as cashing will reduce the need for combining data from several tables with SQL "Join" operation in the project database. Cashing may not be needed for the local scenario 1.1, however to meet the requirement that both local and remote operations from client should be performed over identical API's cashing is used in all three scenarios. The cached project data can be classified as:

a) Permanent project data not to be updated
b) Temporary data that can be updated if changed on the server
c) Variable data that cannot be cached in the client.

There are several types of string storage in the HRE Schema. The issue that needs to be considered here is whether at opening of an HRE project:

1. Is there benefit in reading and saving the content of some Project Tables in Java memory structures?
2. Language - HRE is a multi-parallel language application, where languages are:
   - GUI Language
   - Data Entry Language
   - Second Data Language (use when composing translations)
   - Reporting Language.

In usual use all these are the same language, but they can be different. Thus the GUI Language string data has to be handled by two different methods:

3. If the GUI string value in any language is fixed by a translator, then it is stored as a) in the NLS system
4. If the GUI string value can be edited or new ones created by a user, then it has to be managed by HRE and classified as b).

This mainly applies to labels of selection values. When the state represented by a label is stored in the database it is replaced by an integer key. Subsequent SQL commands use the integer values for comparison.

Because the string translation lists in Item 2 usually are very short and defined in in database tables with _DFNS suffix, these tables should be read into Java structures in memory to reduce the number of JOINS and excessive predictable value mappings.
Because these lookup table key values and translations are likely to be used both by the server and the client, then it is suggested that they be cached in memory in the server on opening of the project. Then each client that has that project open maintains a copy. It is up to the server to keep these string/value caches up to date.