Developing a new HRE feature

Revision History

2018-07-13	Michael Erichsen	Original draft

Programming Principles

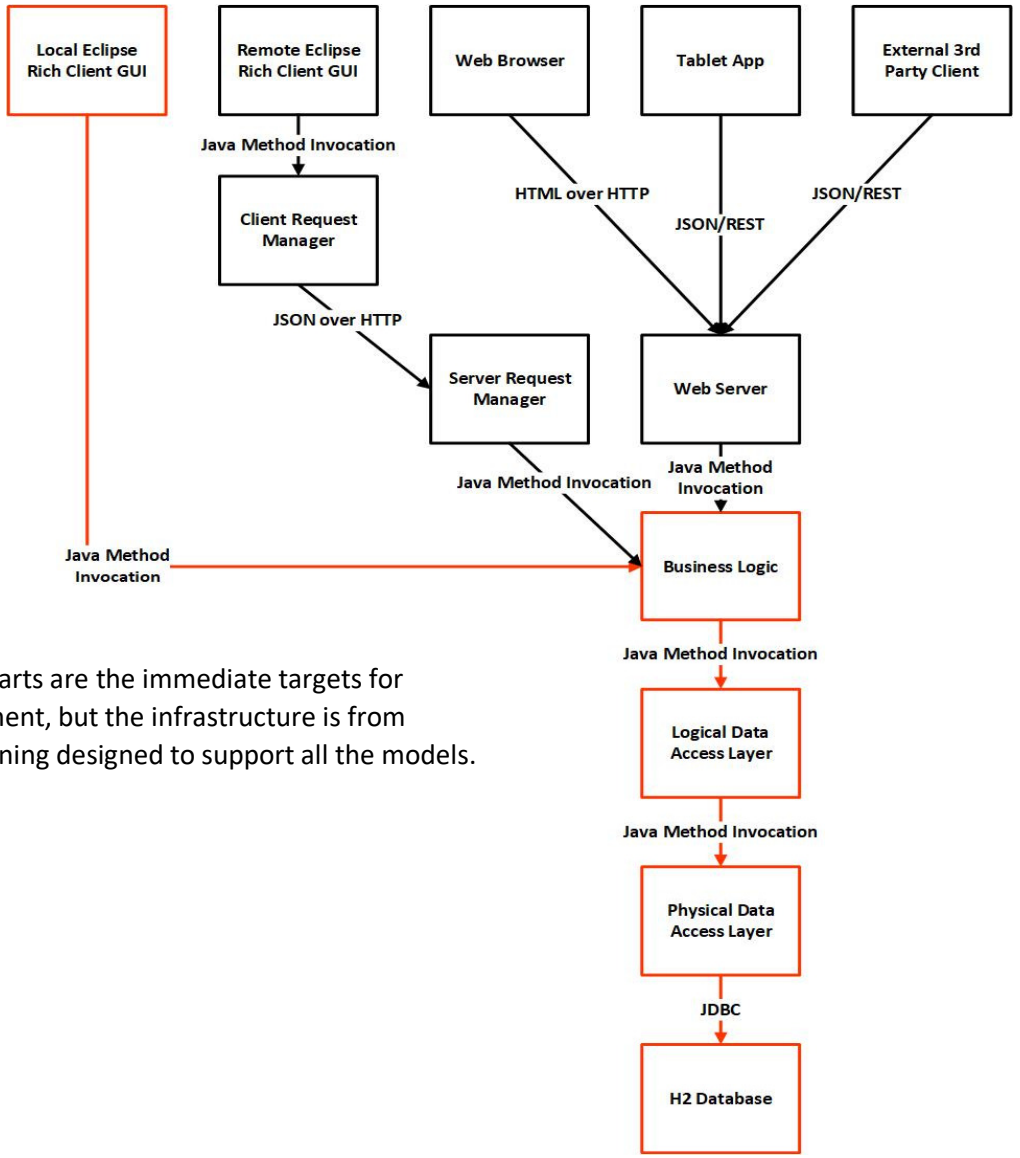
HRE is built on an object-oriented programming model, which especially prescribes encapsulation, inheritance and polymorphism.

Best practices for client-server development prescribes extensible and loosely coupled interfaces.

Best practices for interfaces prescribes using open, established standards, clearly defined protocols and avoiding side-effects.

The Logical Technology Model

The HRE skeleton and sample are developed according to this Logical Technology Model:

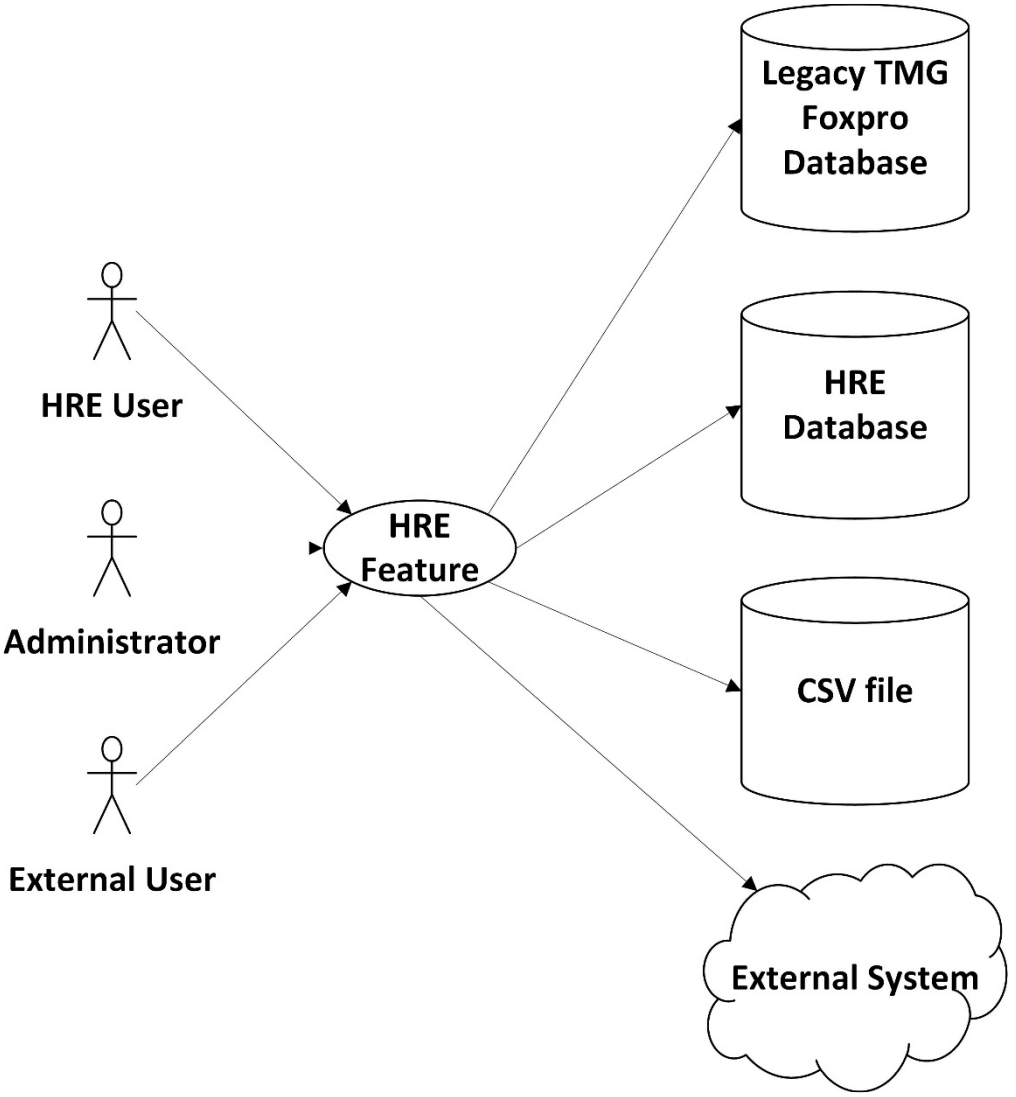


The red parts are the immediate targets for development, but the infrastructure is from the beginning designed to support all the models.

Designing the Use Case for a Feature

The first thing to do is to define the use case that the new feature will implement.

Which kinds of users will need which functions, and which data and external systems will they need to use?



Will the feature be read-only or also update?

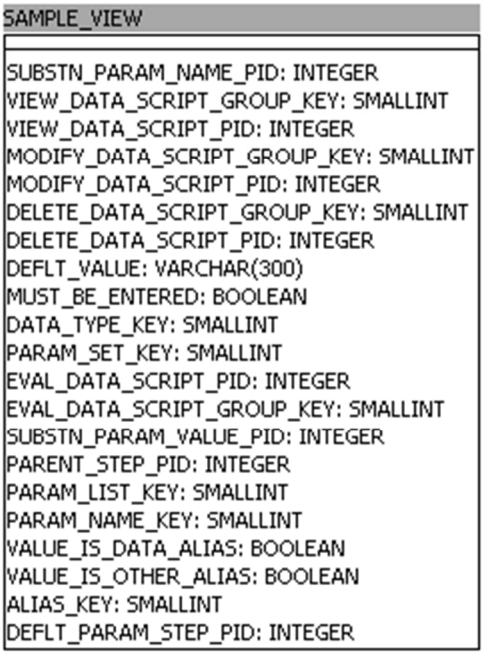
Defining the Data Model

Define the logical data model as the data items that the feature will expose to the user, and which supporting data is needed.

Define the physical data model as the tables, view and columns in the HRE database that should be used by the feature. SQL Power Architect can be used as a tool for this. The Community Edition is available at http://www.bestofbi.com/page/architect_download_os

In this example we have created a database view to implement the physical data model:

```
"CREATE VIEW PUBLIC.SAMPLE_VIEW AS"
        + " SELECT PUBLIC.SUBSTN_PARAM_NAMES.SUBSTN_PARAM_NAME_PID, PUBLIC.SUBSTN_PARAM_NAMES.VIEW_DATA_SCRIPT_GROUP_KEY,"
        + " PUBLIC.SUBSTN_PARAM_NAMES.VIEW_DATA_SCRIPT_PID, PUBLIC.SUBSTN_PARAM_NAMES.MODIFY_DATA_SCRIPT_GROUP_KEY,"
        + " PUBLIC.SUBSTN_PARAM_NAMES.MODIFY_DATA_SCRIPT_PID, PUBLIC.SUBSTN_PARAM_NAMES.DELETE_DATA_SCRIPT_GROUP_KEY,"
        + " PUBLIC.SUBSTN_PARAM_NAMES.DELETE_DATA_SCRIPT_PID, PUBLIC.SUBSTN_PARAM_NAMES.DEFLT_VALUE,"
        + " PUBLIC.SUBSTN_PARAM_NAMES.MUST_BE_ENTERED, PUBLIC.SUBSTN_PARAM_NAMES.DATA_TYPE_KEY,"
        + " PUBLIC.SUBSTN_PARAM_NAMES.PARAM_SET_KEY, PUBLIC.SUBSTN_PARAM_NAMES.EVAL_DATA_SCRIPT_PID,"
        + " PUBLIC.SUBSTN_PARAM_NAMES.EVAL_DATA_SCRIPT_GROUP_KEY, PUBLIC.SUBSTN_PARAM_VALUES.SUBSTN_PARAM_VALUE_PID,"
        + " PUBLIC.SUBSTN_PARAM_VALUES.PARENT_STEP_PID, PUBLIC.SUBSTN_PARAM_VALUES.PARAM_LIST_KEY,"
        + " PUBLIC.SUBSTN_PARAM_VALUES.PARAM_NAME_KEY,"
        + " PUBLIC.SUBSTN_PARAM_VALUES.VALUE_IS_DATA_ALIAS, PUBLIC.SUBSTN_PARAM_VALUES.VALUE_IS_OTHER_ALIAS,"
        + " PUBLIC.SUBSTN_PARAM_VALUES.ALIAS_KEY, PUBLIC.SUBSTN_PARAM_VALUES.DEFLT_PARAM_STEP_PID"
        + " FROM PUBLIC.SUBSTN_PARAM_NAMES, PUBLIC.SUBSTN_PARAM_VALUES"
        + " WHERE PUBLIC.SUBSTN_PARAM_NAMES.PARAM_SET_KEY = PUBLIC.SUBSTN_PARAM_VALUES.PARAM_SET_KEY;" };
```



Getting HRE from Github

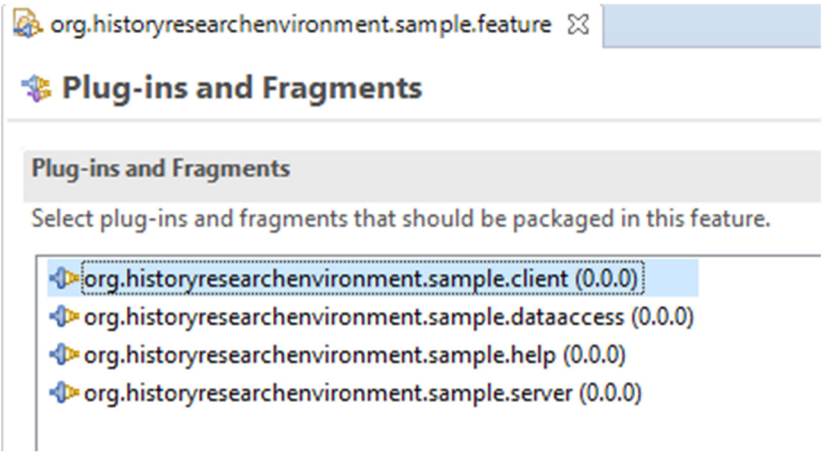
Please clone, branch, fork or whatever is best for you to create your own copy of HRE v0.1 from https://github.com/History-Research-Environment/HRE--History-Research-Environment/tree/develop

Please bear in mind the contents of '*Contributing – Github Branch Management*' which may be found on the HRE GitHub site.

Creating Eclipse Plug-in and Feature Projects

We would normally create a feature project with three or four plug-in projects, as shown.

Data access could be combined into server, if that makes better sense.



For each plug-in project you should create a subfolder with the same name in the bundles folder of HRE. Then you should create the project and take care not to create it in the default location, but in the new folder.

The client project should be created as a Fragment Project with org.historyresearchenvironment.client as the host plug-in.

The Help project should use the available template called “Sample Help Content”.

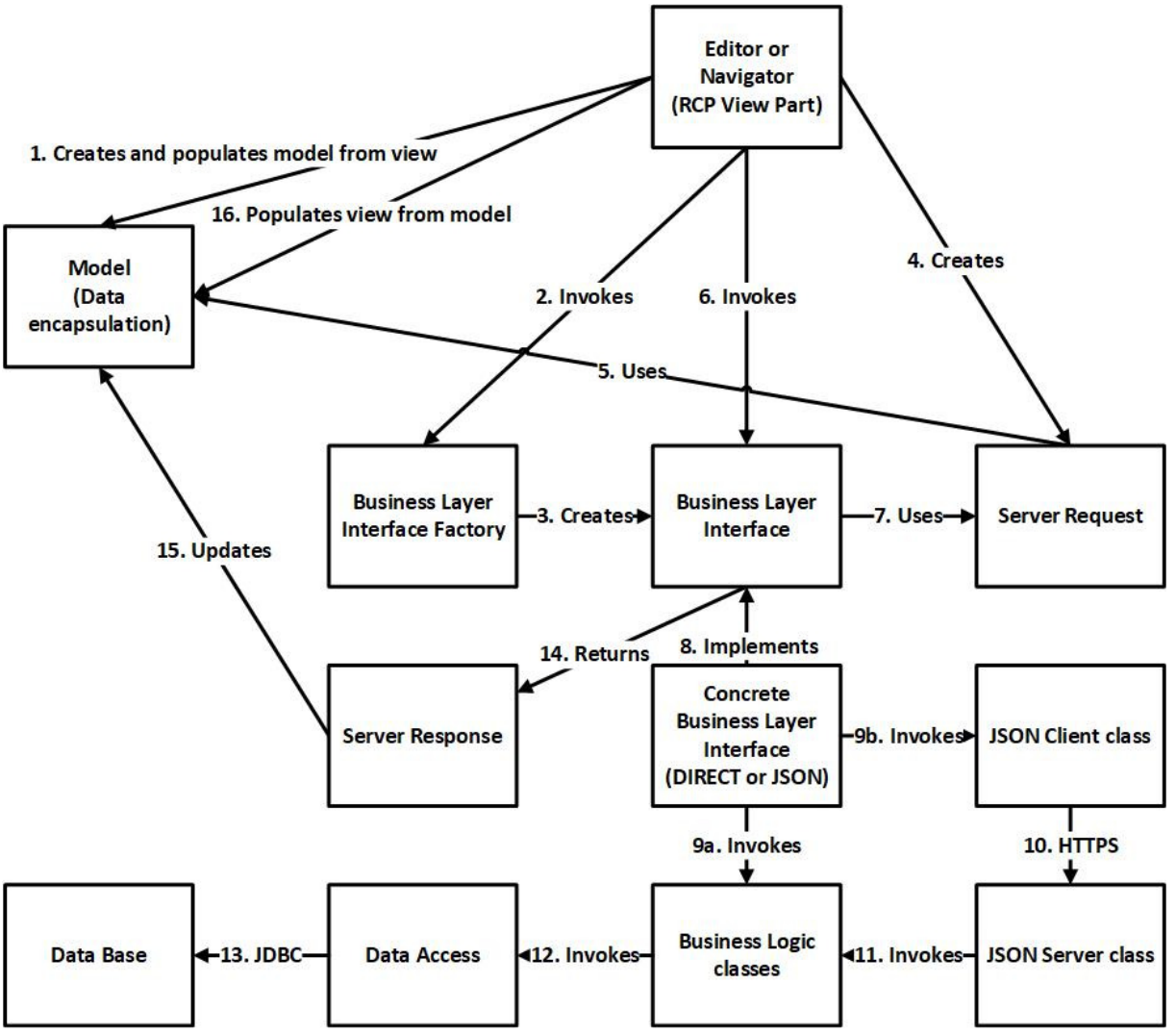
The server project should be created as a simple plug-in project.

The data access project should be created from the Eclipse JPA perspective as a JPA project, if you need to create classes not already present in the base data access project.

The feature project should be created as a Feature Project and then add the plug-in projects.

Application Structure

The java classes of the feature will normally follow this structure:



Implementing Data Access

The mapping of the database tables can happen using database views or by using composite data model classes, where the SQL statements access one or more tables.

Eclipse can generate java classes from H2 tables and views using JPA, Java Persistency Architecture, if they are not already available. These classes have fields, getters and setters for each database column.

These classes should extend AbstractHreDataAccess and be enriched with methods for SELECT, SELECTALL, DELETE, DELETEALL, INSERT and UPDATE.

They normally reside in the data access plug-in. Please note that a database view needs a trigger to permit updates.

Implementing the User Interface

The most normal user interface parts are navigators and editors.

An editor is a view of a single entity, while a navigator is a list or tree view of a set of entities.

The entities are implemented as view model classes, one to encapsulate each field in the editor part, and one to encapsulate a list of single models.

Editor View Part

View Parts reside in the client plug-in and are created using File, New, Other, Window Builder, SWT Designer, Eclipse 4, ViewPart.

They extend AbstractHreGuiPart, which provides font support and a business layer interface.

Design the view part using Window Builder and code the updateGui method to update the GUI from the underlying provider class:

```
SampleEditorProvider provider = new SampleEditorProvider();
SampleBusinessLogic businessLogic = new SampleBusinessLogic();
…
callBusinessLayer("GET", provider, businessLogic, textparamSetKey.getText());
updateGui();
```

AbstractHreGuiPart

Contains the callBusinessLayer method:

```
protected BusinessLayerInterface bli;
protected ServerRequest request;
protected ServerResponse response;

bli = BusinessLayerInterfaceFactory.getBusinessLayerInterface();
request = new ServerRequest("GET", provider, businessLogic);
response = bli.callBusinessLayer(request);
```

View Provider

The view provider extends AbstractHreProvider and encapsulates the data used by the view part.

The provider must implement the readFromH2 method to invoke the data access classes that do the actual database access:

```
SubstnParamName spn = new SubstnParamName(this.paramSetKey);
SubstnParamValue spv = new SubstnParamValue(this.paramSetKey);
```

AbstractHreProvider

Provides generic methods to writeJson and readJson, when running in client/server mode.

BusinessLayerInterfaceFactory

Implements the getBusinessLayerInterface method:

```
if (servertype.equals("DIRECT")) {
            return new DirectBusinessLayerInterface();
} else if (servertype.equals("SERVER")) {
            return new ServerBusinessLayerInterface();
```

DirectBusinessLayerInterface

Implements BusinessLayerInterface by implementing the callBusinessLayer method to invoke the business logic class as requested by the ServerRequest and return a ServerResponse:

```
AbstractHreBusinessLogic businessLogic = request.getBusinessLogic();
ServerResponse response = businessLogic.execute(request);
return response;
```

ServerBusinessLayerInterface

Used for client/server mode.

BusinessLayerInterface

An interface defining:

```
public abstract ServerResponse callBusinessLayer(ServerRequest request);
```

Business Logic

Extends AbstractHreBusinessLogic.

The business logic class executes the provider readFromH2 method to access the database and any other relevant business logic and rules:

```
SampleEditorProvider provider = (SampleEditorProvider) request.getProvider();
provider.readFromH2(paramSetKey);
response = new ServerResponse(provider, 0, "OK");
```

AbstractHreBusinessLogic

Defines:

```
public abstract ServerResponse execute(request);
```

ServerRequest

Encapsulates:

```
private String operation;
private AbstractHreProvider provider;
private AbstractHreBusinessLogic businessLogic;
```

ServerResponse

Encapsulates:

```
private AbstractHreProvider provider;
private int returnCode;
private String returnMessage;
```

Adding Help

Use the HRE_Helpsystem_Generator project to generate a set of HTML pages and images, or do it all by hand.

Then replace the contents of the html folder in the Help project with your pages and add toc.xml pages to the root of the project folder.

Adding the Feature to HRE

This is done by adding the feature to the org.historyresearchenvironment.client.product file in the org.historyresearchenvironment.client.product folder of the releng folder: