Java Enterprise Development Framework 2.01

Build powerful, robust and easy to develop Java desktop application with a high-performance graph-oriented object database in record time!

jedf.org

Contents

1 Introduction

Software development for any business automation begins with the necessity to solve many problems. The developers who decided to get down to this very important work will face the choice of development tools, the choice of data management system, data transfer, integration with the existing software, development and support cost optimization, administration, rights system and users hierarchy, performance and system horizontal scaling and a lot more…

Java Enterprise Development Framework (JEDF) has been created for all-round solving of these and many other problems. JEDF is a rapidly developing platform, which combines graph-oriented object data management system with client application development means.

Wy do we choose NetBeans as the main IDE which we develop JEDF in? There are a few point on this:

- It has a powerful visual disigner for visual development of desktop applications. At least it is much better than we saw in other Java IDE.
- NetBeans is a great plugin platform. It has handy API so that we use it in plugins development for the framework.
- NetBeans is a free IDE that is being active developed by its community.

2 Features

What does it give to software developers?

- Java language to ensure multi-platform. Possibility of migration to Linux with reduction of workplace cost. Opportunity to migrate to Linux with the workplace cost reduction. Using of free software for development and deployment.
- Graph-oriented object database based on any SQL server with business logics inside objects (matters), not inside saved procedures, which enables to use practically any SQL server. Firebird and MySQL tested.
- Objects, which are called Matters, ensure transactions and system integrity, adding fields and properties on the fly. The unlimited number of mutual references between matters enables to create efficient data management systems.
- Rapid visual development of the client software for desktop applications under control of NetBeans using standard Swing with all the JAVA language opportunities. At present JEDF software to make possible a creation of client software for browsers is being developed.
- Ready-made user identification and authentication system, rights to run subsystems, valid forms and reports list.
- Increase in performance due to system scaling, work in a cluster. Data division into several real databases.
- Old data archiving built-in system with the opportunity of quick access to them.
- Request manager to SQL server, which optimizes the number of real connections and makes work with SQL databases much faster.
- Quick transfer of existing programs developed with the help of visual designing systems like Delphi to JAVA technology, using ready-made data-aware components (TableSource) and visual elements.
- Server parts of visual forms which can implement any logics on server side, e.g. about data supplied to the client form.

For more details see the story of creation.

3 Getting Started

3.1 Installing and configuring

Installing and configuring the framework consists of several stages:

- First of all you need to have a blank database instance of any relational database.
- Next download NetBeans IDE 7.4 EE or higher and JDK 1.7 or higher.
- You need to configure the plugins source. To do this you need to open «Tools – Plugins». Next you need to add «Update center»: open «Settings» tab and click «Add» button, as the name of the center write «JEDF Update Center» and as the address «http://jedf.org/nbplugin/updates.xml».
- Install the plugin. You'll see the plugin to installed on the «Updates» tab.
- After plugin installation you have already had JEDFConfig.xml file in your_domain/config directory. This file needs to be filled in since it's the sole configuring xml file that the framework has. This file must have at least one working database alias. To specify it write down the exemplary alias:

```
<?xml version=»1.0» encoding=»UTF-8»?>
<JEDF>
    <Aliases>
     <Alias name=»JEDFSystem»>
            <db_type>firebird</db_type>
            <driver>org.firebirdsql.jdbc.FBDriver</driver>
            <jdbc_url>jdbc:firebirdsql://localhost:3050/C:/JEDFDataBase/
JEDFSYSTEM.FDB</jdbc_url>
            <login>db_login</login>
            <password>db_password</password>
        </Alias>
    </Aliases>
</JEDF>
```

Where:

- db_type – the type of used database; there are the following options: mysql, firebird.
- driver – a full qualified class name of JDBC driver.
- jdbc_url – the database connection string.
- login/password – authentication data to the database instance.

This configuration is enough to make the framework working properly. JDBC driver itself must be put in your_domain/lib/ext directory. After that your

database will be connected to the framework.

- The plugin adds a new project type to NetBeans – JEDF Master Project – the project of controlling building and deployment of JEDF. This project is designed to work with the GlassFish Application Server (version 4+) and contains a set of libraries composing the framework. Instantiated project contains a standard project of java library that has user files (it's easier to work with user files through a standard project, since in this case you can use all the facilities of NetBeans in file management such as refactoring, error highlighting and so on), a folder with JasperReports reports and ant script to build-up server and client framework parts. The plugin also registers 3 libraries in NetBeans:
  - JEDFLibrary – contains system libraries for user classes compilation;
  - JEDFAddClientLibrary, JEDFAddServerLibrary – additional library packages that you can add to client or server part of the framework respectively;
- After the creation of a new project you need to configure a domain that you will be working with (by default it's called domain1 in GlassFish). To do this you need to open project properties and click on «Add» button on «Configuring JEDF servers» tab and after that write in domain name (domain1) and configure the appropriate pathes/names/ports:
  - Path to the application server — path to the installation folder of Glassfish;
  - Domain — domain name (by default is tab name);
  - Administration port (by default is 4848);
  - Debug port – server debug port (by default is 9009);
  - Client port – this port is used for listening commands by the server and also delivery systems of the client application will be deployed;
  - Host;
  - Description – detailed description for the delivery systems;
  - Debugging – a tick of starting the server and the client in debug mode;
  - Web Start — build and deploy the client application as WAR using Java Web Start.
  - JEDF Launcher — build and deploy the client application as WAR using JEDF Launcher.

Deployment specialized settings of the client application are on the appropriate tabs. The context of the client application (WAR-file name) is determined by the «Application» field.

JEDF Launcher was created after the accumulation of certain oddities (bugs) of JWS's behaviour that we used earlier. Some of them:

- A slow start of the application in spite of the fact that JNLP file were set to use methods that indicate the presence/absence of changes in the libraries.
- Regular protocol errors of inability to download a specific library for various reasons.

The deathnail of JWS (in our case at least) can be considered an inexorable trend of Oracle to stop using of self-signed certificates even in the Intranet (an idiotic warning when you try to start the application and the ability to run only at a reduced level of the security). JEDF Launcher provides the ability to centrally update the client application using the HttpURLConnection class for accessing REST-services that provide files for dowload or check.

The installed application is linked to the server from which it was downloaded. I.e. if you run the installer from http://test.jedf.org:9090/TestApp the client application will be access the server exactly at http://test.jedf.org:9090. Both these delivery systems (JWS and JEDF) can be used in common providing that using of different contexts. And, of course, you are free to distribute the client application that is built in dist\client of the project folder.

- After setting up the domain the domain management unit appears. Here is its commands:
  - Rebuild and run classes - stops the domain, compiles custom classes, copies the compiled classes and reports to a special place of the domain and starts the domain and the client application.
  - Run the client - tries to run the client application.
  - Rebuild all and run - the same as the first item but besides custom classes compiling it builds and deploys the client and the server as EAR file. This operation should be performed either after updating the plugin (when changing the system libraries framework) or after adding or changing libraries in NetBeans (JEDFAddClientLibrary and JEDFAddServerLibrary).
  - Prepare the client - builds the distribution kit of the client application and WAR files if needed.
  - Deploy the client - deploys built WAR files on the application server.
  - Prepare the server - builds server EAR files.
  - Deploys the server - deploys built EAR file on the application server.
  - Start / Stop / Restart the server - controlls the application server.

Another important detail in the configuration is JEDFConfig.xml file. It contains

information about databases connections and cluster mode. Let's see in example each of them.

Database configuration is located in JEDF/ Aliases/ Alias path:

```
<Alias name="JEDFSystem">
    <db_type>firebird</db_type>
    <driver>org.firebirdsql.jdbc.FBDriver</driver>
    <jdbc_url>jdbc:firebirdsql://localhost:3050/C:/JEDFDataBase/JEDFSYSTEM.
FDB</jdbc_url>
    <login>sysdba</login>
    <password>your_pasword</password>
</Alias>
```

Name attribute in Alias tag is the name of the alias. It is used in many different parts of the framework. You may define as much as you want database aliases in your project.

To activate cluster mode of your system simply add MainServer tag inside JEDF tag. An exemplary set of tags can be:

```
<MainServer>
    <IP>192.168.0.50</IP>
    <PORT>3128</PORT>
</MainServer>
<AddServer>
    <IP>192.168.0.51</IP>
    <PORT>3128</PORT>
</AddServer>
<AddServer>
    <IP>192.168.0.52</IP>
    <PORT>3128</PORT>
</AddServer>
<AddServer>
    <IP>192.168.0.53</IP>
    <PORT>3128</PORT>
</AddServer>
```

There is can be only one `MainServer` tag that describes IP and port of the main server but you may define lots of additional servers of the cluster by adding AddServer tag, this is your additional servers of the cluster. To see more information about clustering, open 6.

3.2 Hello World Example

After creating JEDF Master Project (see 3.1) open a project inside the created JEDF Master Project folder which is called Classes. This is the project that contains source files itself. The Classes project has already various classes that need for

correct working of the framework. Let's create a simple window that interacts with the server module.

Click right mouse button on the Classes project and select «New->Other...» and in the opened dialog select «JEDF Forms» in the categories list and select «Base Frame» in the file types. Click next and set a class name of the form and a package name. Click finish and your first frame will be created. Next let's add a simple button to any place of the form.

To interact with the server side you need to create the corresponding server part of the form. To do that click again right mouse button on the Classes project and select «New->Other...» and select «JEDF form» and «Base Server Module». Fill out all the necessary fields. There is a simple convention for naming server modules: each server module must have the same name as the form but with «Server» ending (should be with a capital letter «S»). After that click finish button.

Next, you need to add the execCommand method in the server module:

```
@Override
public synchronized InterchangeObject execCommand() {

}
```

It's inherited method from the ShapeServer class that is base for all server module classes so we add @Override annotation. Inside this method add something like that:

```
RequestContextI context = ActiveContexts.getContext();
String s = context.getStringProperty("helloWorldExample");
InterchangeObject res = new InterchangeObject();
switch (s) {
 case "initData": {
       java.util.Random r = new java.util.Random();
       res.setProperties("r", r.nextInt());
       break;
 }
 default: {
       return null;
 }

}
return res;
```

Now we go back to the client side of our module. Double click to added earlier button and you'll see created an event handler like that:

```
private void theButtonActionPerformed(java.awt.event.ActionEvent evt) {
```

```
}
```

Just add the following code snippet to this handler:

```
Map<String, Object> res = execServerCommandRetProps(null,
"helloWorldExample", "initData");
if (res != null) {
 Integer r = res.get("r");
 if (r != null) {
       JOptionPane.showMessageDialog(BaseFormExample.this, "Hello from the
server: " + r);
 }
}
```

Where BaseFormExample is the class name of the client frame. After that you need to add a simple annotation to the client frame class declaration:

```
@ShapeFrameDcl(shapeName = "Base Form Example",
               iconName = "su/jedf/common/images/master_joda.png",
               isStartFrame = true)
```

This allows you to use this frame without dealing with rights system immediately, because this is an example. iconName is a path to the icon that represents the frame on the starting window. Of course you are allowed to add any other icon for your frames.

After saving select your domain in JEDF Master Project and click right mouse button on it and select the first item «rebuild classes and run». After a while you'll see a standard login dialog. After passing it you'll see an icon of your created form. Just double click this icon to open your window and click on the button to get the message from the server side. Thats all.

Thus we got to know how to create a frame that allows us to use all the infrastructure of the JEDF. To see how to dealing with the MatterBase open 4.1 of this reference.

3.3 Example of using BaseListFrame

Let's create a little more complicated example. Here we're going to use a template with a preset JEDFTable and TableSource. These two simplify dealing with table data. Click right mouse button on the Classes project and select «New->Other...» and in the opened dialog select «JEDF Forms» in the categories list and select «Base List Frame» in the file types. lick next and set a class name of the form and a package name. Click finish and your first frame will be created.

Also we need to create the corresponding server part of the form click right mouse button on the Classes project and select «New->Other...» and select «JEDF form» and «Base Server Module». Fill out all the necessary fields. Server module name must consists of «Client frame name» + «Server» (See the previous example). After that click finish button.

Move to the design view in NetBeans, in the Navigator click right mouse button on listSrc[TableSource] in Other Components and in SQL property set:

```
«select cdmatter from report»
```

And set to Alias property «JDFSystem» value. This is necessary to work with the service table of the Report matter.

Just change the body of the onInit method:

```
@Override
protected void onInit() {
    initComponents();
    setTitle("Base list frame example");
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.Add),
toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.Save),
toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Remove), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Filter), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
FilterApply), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Print), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Reopen), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.Exit),
toolBar, controlM);

    listSrc.addFilterBtn(getJDFButton(JDFCommand.Filter));
    listSrc.addFilterApplyBtn(getJDFButton(JDFCommand.FilterApply));
    listSrc.addRemoveBtn(getJDFButton(JDFCommand.Remove));
    listSrc.setPrintBtn(getJDFButton(JDFCommand.Print));


    listSrc.setSourcesHost(getModel());
    listSrc.setResources(JDFResources.getInstance());

    String pM = "org.jedf.metashaper.server.matter.systemclasses.
ReportMatter";
    listSrc.newField("cdMatter", "Report code", Long.class, 45, true, pM).
setKey(true);
    listSrc.newField("", "name", "cdMatter", "Report name", String.class,
```

```
200, true, pM);
    listSrc.newField("", "className", "cdMatter", "Report class name",
String.class, 200, true, pM);
    listSrc.newField("", "comment", "cdMatter", "Comment", String.class,
200, true, pM);

    setVisible(true);
}
```

Here we use a standard matter ReportMatter which is responsible for the reporting system. Firstly we define a set of buttons via the JButtonFactory class, then initialize TableSource (listSrc instance) and finally set a full qualified class name of our matter and specify 4 fields of the table. Also we need to add a special annotation to class declaration:

```
@ShapeFrameDcl(shapeName = "Base List Frame",
           iconName = "org/jedf/common/images/master_joda.png",
           isStartFrame = true)
```

That enables us to run this form from the start form directly.

And here is a final small detail. Go to the server module and add the RootShapeServerI interface to class declaration. This interface has two methods that needs to be implemented:

```
@Override
public boolean checkRootRight(String string) {
    return true;
}

@Override
public void checkRootRightX(String string) {

}
```

These methods allow us to run our exemplary form without dealing with the rights system. By these few steps we built the frame that can add, change, remove, filter entries.

After saving select your domain in JEDF Master Project and click right mouse button on it and select the first item «rebuild classes and run». After a while you'll see a standard login dialog. After passing it you'll see an icon of your created form. Just double click this icon to open your window and you'll see created frame. Thats all.

3.4 Example of binding components

Our next example will be about using binding components. Assume the following situation: you need a form that has a table with some rows and a few components that connected with the table depending on the selected row. By

selecting a new row will trigger refreshing the content of these components. You a free to combine whatever data of the table and the data of the components. Let's see how to do this on the simple example.

As in the previous part yoou need to click right mouse button on the Classes project and select «New->Other...» and in the opened dialog select «JEDF Forms» in the categories list and select «Base List Frame» in the file types. Click next and set a class name of the form and a package name. Click finish and your first frame will be created.

Also we need to create the corresponding server part of the form click right mouse button on the Classes project and select «New->Other...» and select «JEDF form» and «Base Server Module». Fill out all the necessary fields. Server module name must consists of «Client frame name» + «Server». After that click finish button.

Move to the design view in NetBeans, in the Navigator click right mouse button on listSrc[TableSource] in Other Components and in SQL property set:

```
«select cdmatter from report»
```

And set to Alias property «JDFSystem» value. This is necessary to work with the service table of the Report matter.

Go to the design view, click right mouse button on GridBagLayout item in [JFrame] section and select «Customize». Click right mouse button on the left vertical gray field on the side of JEDFTable component and select «Insert row after». Add a component named «JEDFEditor» to the added row and fill the «Fill» property to this component to «Both» and close the layout editor. Open properties of the added JEDFEditor component add set «Report class name» to «_label» and «100» to «_labelWidth».

Now go to the source view and insert into the onInit method this code:

```
@Override
protected void onInit() {
    initComponents();

    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.Add),
toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.Save),
toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Remove), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Filter), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
```

```
FilterApply), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Print), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.
Reopen), toolBar, controlM);
    JButtonFactory.createJButtonSpeedAndMenu(getJDFButton(JDFCommand.Exit),
toolBar, controlM);

    listSrc.addFilterBtn(getJDFButton(JDFCommand.Filter));
    listSrc.addFilterApplyBtn(getJDFButton(JDFCommand.FilterApply));
    listSrc.addRemoveBtn(getJDFButton(JDFCommand.Remove));
    listSrc.setPrintBtn(getJDFButton(JDFCommand.Print));

    listSrc.setSourcesHost(getModel());
    listSrc.setResources(JDFResources.getInstance());

    String pM = "org.jedf.metashaper.server.matter.systemclasses.
ReportMatter";
    listSrc.newField("cdMatter", "Report code", Long.class, 45, true, pM).
setKey(true);
    listSrc.newField("", "name", "cdMatter", "Report name", String.class,
200, true, pM);
    listSrc.newField("", "comment", "cdMatter", "Comment", String.class,
200, true, pM);

    listSrc.bindComponent(reportClassNameEd, "className", "value", String.
class);

    setVisible(true);
}
```

This line makes binding the JEDFTable to the JEDFEditor. All the data that will be in the row will be mapped to the according components («className» field in this case). You can change the content of the reportClassNameEd and after saving it changes will in the table.

Don't forget to add the annotation to the class declaration (see the previous example):

```
@ShapeFrameDcl(shapeName = "Base List Frame Bind",
            iconName = "org/jedf/common/images/master_joda.png",
            isStartFrame = true)
```

Where «Base List Frame Bind» is the title of your frame.

Next you need to create the server module of our frame (see the previous example). Go to the server module and add the RootShapeServerI interface to class declaration. This interface has two methods that needs to be implemented:

```
@Override
public boolean checkRootRight(String string) {
```

```
        return true;
    }

    @Override
    public void checkRootRightX(String string) {

    }
```

After saving select your domain in JEDF Master Project and click right mouse button on it and select the first item «rebuild classes and run». After a while you'll see a standard login dialog. After passing it you'll see an icon of your created form. Just double click this icon to open your window and you'll see created frame.

3.5 Deploying the application

A couple of words about deploying your application. Regardless of used technology (JWS or our own JEDF Launcher) finally you or all your users at least will work with only one shortcut that runs the application. The detailed procedure of the deployment is described in 3.1.

4 MattersBase – Graph-oriented object database

The idea of any object oriented database is obvious, such database doesn't operate records and tables but only objects. Object oriented databse JEDF MatterBase isn't the exception, it has to do with only objects. Each object, which would be kept in database, must be an heir of the abstract field of the class Matter. Matter class contains all the necessary mechanisms in order to operate with the database. MatterBase's features:

- Each instance of the Matter that we can work with in the same way – using the same methods of the base Matter class. There is no need to know what exactly subclass is used. All the necessary methods are realized in the Matter class.
- Each matter can have unlimited number of links with another matter.
- Supporting relation's type «many-to-many» without using additional data structures.
- Each matter knows its structure and any changes in the class can't corrupt data that already is stored in the databse.
- Instances of the same matter class can even have their own unique field set.
- Adding or removing fields in a mater in run-time is correct operation.
- Access to references of the matter is implemented by using regular

invocations of java methods, without using SQL. MatterBase will perform all the necessary actions in order to get objects from the databse as required.

- Distribution a large volume of data among different databases.
- Built-in archiving system.
- Built-in versioning system.

To work with matters any extra actions aren't required, but to create a new matter, you need to fill it with data and save; MatterBase will perform all the rest:

- Transaction's starting and committing.
- Putting a matter to the cache.
- Saving a matter in the database.
- Loading one from the database.
- Managing referential integrity.
- Cluster management.

4.1 Quick start

The main goal of MatterBase, like any other, is to provide developers with the facility of dealing with data. It's simple to understand how to use matter's mechanism with the following example. In the beginning let's create one matter and save into the database. Let's assume that the matter describes a measurement unit. In this matter we have sole field - a denomination of the measurement unit. The class of client matter must be inherited from the base matter class «Matter». Next we'll override all the abstract methods of the Matter class.

```
public abstract short getCode();
```

This method returns a unique code of the matter class. The code of the matter class is a number of short type, its value can be from 5001 to 20000 inclusive. Each value relates to only one matter class. MatterBase checks code uniqueness of all matter classes while starting the system. Codes from 0 to 5000 are reserved for the system. All matters must be registered in JEDF (see JEDFConfig.xml and the compiling plugin in 3.1).

```
public abstract Matter getNewClone();
```

This method returns a new instance of the current class.

```
public abstract void fillDefaultFields(MatterFields fields);
```

This method sets values to field object.

```
public abstract String getInfo() throws Exception;
```

This method returns information on the matter. This method enables to watch matter list not kwonwing their real fields.

There is a listing of the exemplary matter class MshMatter.java

```
package su.jedf.metashaper.server.matter.userclasses;

import su.jedf.annotations.MatterDcl;
import su.jedf.metashaper.auxiliary.ShapeMalfunction;
import su.jedf.metashaper.server.matter.Matter;
import su.jedf.metashaper.server.matter.MatterFields;
import su.jedf.metashaper.server.matter.ServiceTable;
import su.jedf.metashaper.server.matter.fields.MatterFieldString;

/**
 * Measurement unit
 * @author EC
 */
@MatterDcl(code = MshMatter.code)
public class MshMatter extends Matter {
    public static final short code = 5004;
    private static final long serialVersionUID = 1L;

    @Override
    public void fillDefaultFields(MatterFields fields) {
        addField(fields, new MatterFieldString (MSH, ""));
    }

    @Override
    public String getInfo() throws ShapeMalfunction {
        return getString (MSH);
    }

    @Override
    public short getCode() {
        return code;
    }

    @Override
    public Matter getNewClone() {
        return new MshMatter();
    }

    public static final String MSH = "msh";
    public static final String CD_MSH = "cdMSH"; //For matters that refer to
measurement units
    public static final String SERVICE_TABLE_NAME = "MSHREF";
```

```
    @Override
    public ServiceTable[] getServiceTableArray() {
        return st;
    }

    private static ServiceTable[] st = {new ServiceTable (SERVICE_TABLE_
NAME, "",
            new String[]{MSH},
            null,
            new String[]{"alter table mshref add unique (msh)"},
            new MshMatter())
    };

}
```

This class declaration has an annotation @MatterDcl whose target is to generate a class for matter registration in the system. Also this class declaration has one optional so-called service table, which is used for searching instances of the created matter class and to prevent matters with the same value in «msh» field from duplication (see service table in detail 4.7).

After creation of the matter class we need to organize the following operations:

- Creating instances
- Modifying instances
- Saving a modified instance

All these operations will be performed on JEDF server as a result of user actions.

JEDF considers a frame (window) as a unit of interaction with a client. The frame has controls by using them it can perform operations including calling JEDF server. These controls are standard Swing components and their heirs. Each frame has created server representation where unique parameters and data from the form itself are stored; the frame only presents data to the user.

If you want to perform some specific operations with matters, you will have to create your own server side frame. The server side form must be an heir of the ShapeServer class and has a compound class name: «FrameClass» + «Server». To set an example «QStart» - is the name of the frame and the server side representation «QStartServer».

A couple of words about dealing with the server side: if a frame uses standard behavior, such as lists browsing and editing, references browsing and editing, you won't need to create a server side representation for performing application tasks in most cases, but this option enables to perform any possible operations for JEDF

server and GlassFish application server (JEDF works under GlassFish control but uses only web-services JAX-RS RI for data exchange with the client part; there is also JEDF without using GlassFish).

Further we need to create a frame with controls performing them the server side code will be executed. Only on the server side the creation of an instance will be performed and then will be saved to the database.

To do this we need to run NetBeans, create a frame class, which is an heir of the ShapeFrame class, create a server class, which is an heir of the ShapeServer class, and override the «execCommand» method (see working with frames in detail in this part below).

```
package su.jedf.quickstrat;

import java.util.Map;
import javax.swing.JOptionPane;
import su.jedf.client.ShapeFrame;

/**
 * @author EC
 */
public class QStart extends ShapeFrame {

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        mainTB = new javax.swing.JToolBar();
        StartWHMB = new javax.swing.JMenuBar();
        debugM = new javax.swing.JMenu();
        do1MI = new javax.swing.JMenuItem();
        jSeparator1 = new javax.swing.JPopupMenu.Separator();
        closeMI = new javax.swing.JMenuItem();

        setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_
CLOSE);
        setTitle("Quick Start");
        getContentPane().setLayout(new java.awt.GridBagLayout());

        mainTB.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.
swing.border.BevelBorder.RAISED));
        mainTB.setFloatable(false);
        mainTB.setMaximumSize(new java.awt.Dimension(2000, 1000));
        mainTB.setMinimumSize(new java.awt.Dimension(500, 250));
        mainTB.setPreferredSize(new java.awt.Dimension(500, 250));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
```

```
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.weighty = 1.0;
        getContentPane().add(mainTB, gridBagConstraints);

        debugM.setText("Do");
        debugM.setActionCommand("debug");

        do1MI.setText("Do1");
        do1MI.setActionCommand("sync");
        do1MI.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                do1(evt);
            }
        });
        debugM.add(do1MI);
        debugM.add(jSeparator1);

        closeMI.setText("Close");
        closeMI.setActionCommand("fixDublFields");
        closeMI.setName(""); // NOI18N
        closeMI.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                close(evt);
            }
        });
        debugM.add(closeMI);

        StartWHMB.add(debugM);

        setJMenuBar(StartWHMB);

        pack();
    }// </editor-fold>

    private void do1(java.awt.event.ActionEvent evt) {
       Map<String, Object> m = execServerCommandRetProps(null, "cmd",
"do1");
       if (m != null) {
           JOptionPane.showMessageDialog(this, m.get("info"));
       }
    }

    private void close(java.awt.event.ActionEvent evt) {
        closeFrame();
    }

    // Variables declaration - do not modify
    private javax.swing.JMenuBar StartWHMB;
    private javax.swing.JMenuItem closeMI;
    private javax.swing.JMenu debugM;
    private javax.swing.JMenuItem do1MI;
```

```
    private javax.swing.JPopupMenu.Separator jSeparator1;
    private javax.swing.JToolBar mainTB;
    // End of variables declaration

    @Override
    protected void onInit() {
        initComponents();
        setVisible(true);
    }
}
```

This exemplary frame contains a menu with two items. The first one performs writing of the new instance MshMatter class to the database and outputting a text message, which has been sent by the server (see void do1(java.awt.event.ActionEvent evt)). The second item closes the frame (see void close(java.awt.event.ActionEvent evt)).

In the method «do1» «execServerCommandRetProps» method is used which allows to execute user command on the server side of the frame.

There is a listing of the server side of the frame:

```
package su.jedf.quickstrat;

import su.jedf.metashaper.auxiliary.InterchangeObject;
import su.jedf.metashaper.interfaces.RootShapeServerI;
import su.jedf.metashaper.server.matter.Matter;
import su.jedf.metashaper.server.matter.MatterRegister;
import su.jedf.metashaper.server.matter.userclasses.MshMatter;
import su.jedf.metashaper.server.matter.userclasses.spec.AnswerMatter;
import su.jedf.metashaper.server.request.ActiveContexts;
import su.jedf.metashaper.server.request.RequestContextI;
import su.jedf.metashaper.server.shape.ShapeModelSrv;
import su.jedf.metashaper.server.shape.ShapeServer;

/**
 * @author EC
 */
public class QStartServer extends ShapeServer implements RootShapeServerI {

    public QStartServer(ShapeModelSrv shapeModelSrv) {
        super(shapeModelSrv);
    }

    @Override
    public boolean checkRootRight(String rightName) throws Exception {
        return true;
    }

    @Override
```

```
    public void checkRootRightX(String rightName) throws Exception {
    }

    @Override
    public synchronized InterchangeObject execCommand() throws Exception {
       //gets the current server context
        RequestContextI context = ActiveContexts.getContext();
       //property obtaining
        String s = context.getStringProperty("cmd»);

        AnswerMatter answerM;
        switch (s) {
            //opens answer's root element
            case "do1":{
                //creating a new matter
                Matter m = MatterRegister.getNewMatter(MshMatter.class.
getName());
            //saving to the matter
              //In this example we write "MSH" and the matter's code.
              //The code is added only because to create different matters
by multiple executing this commands
              //and to avoid duplication of matters with the same
measurement unit name.
                m.setValue (MshMatter.MSH, "MSH" + m.getCdMatter());
                //Saves the matter to MatterBase
                m.putToBase();
                //Formation of the response to the frame
                return new InterchangeObject (null, "info", "ok");
            }
        }
        return null;
    }

}
```

(The description of obligatory methods checkRootRight and checkRootRightX see 7.3).We are going to give you a little more information about execCommand method. The method begins with getting current server context. The context is the set of current parameters handling the server, the name of the current frame, user parameters, a set of passed parameters and so on. Next we get the value of the «cmd» parameter. In «case» section we check the «cmd» parameter value and if it's equal to «do1», then creation of the matter is executed and value of the «msh» field is set. Next the matter is saved to the database and the response to the frame is formed. The searching of «cmd» and checking of «do1» was unnecessary, but this example shows a standard approach of calling and processing diverse user commands coming from the frame to its server side. To execute other commands it's enough to add more case branches with proper actions.

JEDF launching, see 3.1

If the launch has been successful, here appears a typical logon window inquiring user login and password see 7.2. If an authorization has passed, here appears a set of starting frame icons, which are allowed to the authorized user; here must be an icon of administration frame at least. If you click on the administration button, an administration tool will open. Using this tool you can add or remove users, manage department structure, manage users rights, manage frames, deal with matters directly, monitor statistics, etc (see 7). To register the created frame, select the third menu item named «Frames», and a window with a list of frames will run. Click on plus icon to add a new frame. Fill in columns with the following names:

- Form name – the name of the form; e.g «do1».
- Form's class name – full qualified class name of the form; e.g. «su.jedf.quickstrat.QStart».
- Main form – if you want this frame to be run after logon, tick off this parameter.

After all alterations click on [floppy_disk.png] to save all changes and after that close the window by clicking on [quit.png]. The control returns to the list of available frames. Refreshing this list you will see a standard icon of the frame titled «Do1». By clicking it, the frame will open. In the opened frame select «Do1» menu item and the frame will show «OK» message. Click it again and you will see «Ok» message a second time. Thereby you have already created and saved two matters into MatterBase. To make sure run administration tool and select «Admin/View Matters». Input «select * from MshRef» to «code or sql query» text box in the appeared window and press «Run». In the left tree you will see all instances of MSH class matter that were registered in the system. All of them are named «MSH» + «number». See more about Matter administrator tool 7.

That was your fist experience with JEDF. In this example you get to know all the main parts of the system, such as client frame, frame server side representation, matter class creation, processing user commands on the server side, creation and saving matter instances. In this reference you can find description of these parts in detail.

4.2 Storage data in matters

Each value of a certain matter is kept in an appropriate field. The number of matter fields isn't restricted. Fields in different instances of some matter class can be diverse. To add or remove a field there are addField and delField methods. To get or set field value use getValue and setValue, respectively. If a field with

specified name doesn't exist, getValue method will return null, but if you try to set value by using «setValue» method, you'll get an exception (see 4.4.2).

4.2.1 Field types

MatterBase has following types:

- MatterFieldByte – one byte.
- MatterFieldString – a string; maximum characters number is equal to Integer. MAX_VALUE (it's a regular java string).
- MatterFieldLong – a long data type.
- MatterFieldRef – a so-called reference field that can refer to another matter and that one is able to get an opposite reference to the first matter. For instance, a matter representing a line of a document has a field «document's code» that will be referring to the head of the document. It's possible to get all document rows from the doucments header just by using a standard method «doFilterRefs» and without direct access to MatterBase and even using SQL.
- MatterFieldRefCount – another reference field. In this case, a referenced matter doesn't know what matters refer to it. It knows only about references existence. For example, it doesn't matter that the reference, which refers to measurement unit matter, uses a bidirectional reference because it doesn't matter what matters use this instance of the measurement unit matter. This type saves system resources in references inserting and changing, especially in cluster mode.
- MatterFieldDouble – contains one double value.
- MatterFieldByteArray – ordinary byte array – byte[].
- MatterFieldTimestamp – for containing date or time.

4.2.2 Handling field values

All fields in MatterBase have two main methods to work with field values: getValue and setValue. The value type must correspond to the field type. All modifications are accompanied by transactions. If a transaction isn't specified as a parameter while setting the value, it will be created implicitly.

4.3 Reference mechanism

Reference in MatterBase is a system class that is created automatically by the system while handling references. All matter references are stored inside the matter. A programmer can handle all matter references using «doFilterRefs»

method. MatterBase supports various kinds of references among the matters. Any matter can refer to any other matter or to itself; there are no restrictions.

MatterBase supports referential integrity automatically. It is impossible to remove some matter that has references from other matters. It is also impossible to set a reference to the matter that doesn't exist.

All references use a unique matter code. Uniqueness is guaranteed to all JEDF servers joined to the cluster. A matter code is a long value which is generated automatically and doesn't change during the matter lifetime in JEDF. This code can be given by calling «getCdMatter» method.

4.3.1 Reference types

Reference types of MatterBase are different from traditional references that are used in relational databases. MatterBase has all the types that are in relational databases and in addition to them there are extra types with special behaviour and usage. References are stored in special inner structures.

There are the following possible matter types between two matters:

- NOTHING - a reference between two matters (let's assume A and B) without involving any matter field and, as it is reflected in the name, this type doesn't set relation between A and B. The reference is set by the static method MatterRefere.CreateLink. In fact, two references will be created. One is kept in A and the other is in B. This reference type corresponds to the relation «many-to-many» or «one-to-one». To clear up briefly, if there is a reference between two matters, it is «one-to-one» relation. But if there are more matters connected with the matter, it is «one-to-many» relation. There are no restrictions to the number of matter references. It's an important advantage of JEDF and MatterBase over classic relational databases.
- DOWN - a reference between two matters (let's assume A and B) without involving any matter field with set relation between A and B. This reference type presupposes that A is situated above B in matter graph structure. The reference is set by the static method MatterRefere.CreateLink. Actually, two references will be created, one reference will have DOWN type and it will be kept in A, and the other one will have UP type and it will be kept in B. This type is useful while describing directional graphs or trees. It corresponds to relational databases' «many-to-many» and «one-to-one» types, but it also has one-directional reference feature.
- UP - bi-directional reference. The same as DOWN, but it has the opposite direction.
- DOWNLINK - one-directional reference from A to B without using matter fields. It is created by MatterReference.CreateLink method. The exact copy of matter B will be created in matter A, and in turn, B matter will have only a mark of having references from class of matter A.

- UP_EXISTS - it is created only by the system as a reverse reference in additional to DOWNLINK references.
- REF - one-directional reference from a matter field to another matter. Let's suppose that matter A has a field with MatterFieldRef type. Then if you set value different from 0, MatterBase will seek a matter that has cdMatter which is equal to the set value. If such matter isn't found, an exception will be thrown; if it is found, a reverse reference with REF type will be created in the found matter. In this case the created reverse reference will have exact data about its usage by the matter B. It corresponds to «one-to-many» relation.
- REF_EXISTS - one-directional reference from a matter field to another matter. If some matter A has a field with MatterFieldRef type, then you set value different from 0, MatterBase will seek a matter that has cdMatter which is equal to the set value. If such matter isn't found, an exception will be thrown; if it is found, the mark of usage by the class of matter A will be set.

4.3.2 Using references

As it was said above, MatterBase has two kinds of references.

The first type stands for references from matter fields and references between matters without using their fields. For instance, there is ShapeMatter matter, which describes one user frame and has CD_OWNER_PERSON field, which is a reference field, refering to PersonMatter matter (description of one user). This is a reference with using matter field.

The second type stands for using references not connected with matter field. Let's say matters PositionMatter and PersonMatter can be connected without using their fields. The point is that in one work place there can be found unlimited numbers of employees and there can also can be unlimited numbers of work places.

4.3.2.1 Creating and deleting references

MatterBase creates MatterFieldRef and MatterFieldRefCount references automatically if the reference field is set a nonzero value. To delete a reference is by setting 0 (zero).

If you need to create a reference between matters witout using their fields you have to use MatterReference.CreateLink static method. To remove reference call MatterReference.RemoteLink.

4.3.2.2 Field references

Field references (MatterFieldRef and MatterFieldRefCount) are nearly ordinary

fields of Long type. getValue method returns cdMatter of the matter that these fields are referenced to. The difference is that the system will not perform additional actions and checks while changing those fields. While changing field values of that class, MatterBase will perform all the necessary changes in matters that these fields are referenced to.

It's normal to work with these fields as with usual fields using getValue and setValue methods. Moreover, there is getValueRef method which returns the matter that these fields refer to. But if the reference field has zero value, it will return null. For convenience, you are allowed to set a matter in these fields instead of its cdMatter code. To remove the reference from reference field is by setting zero value. All changes relating to reference setting or removing are performed inside one transaction.

4.3.2.3 Direct references

These references are created by MatterReference.CreateLink method. Due to this approach it is allowed to organize arbitrary references among matters. To remove this type reference is by calling MatterReference.RemoteLink method. All matter changes are made in one transaction. The feature of these references is that a reference can have additional data. A reference is saved in inner structures of both matters.

4.3.2.3.1 Setting value to the direct references

As a reference is an ordinary class, it can have additional data. This feature is available for the direct references only and not for the field references. For example, you set a reference between two matters A and B and as additional data you can set a value to this reference. You can do this by calling setValueRef method or while constructing a reference object.

4.3.2.4 Handling all matter references.

MatterBase has an efficient mechanism to work with references. There is no need to access the database or even to use SQL. Since all references are kept inside their matter, we can work with them by using instance methods. The main method to do this is doFilterRefs method. This one enables to select only required references.

Here is a working example where it is getting all references of the matter that describes a frame containing right matters (ShapeRightsMatter). In other words, it is forming the list of all frame rights.

```
shapeMatter.doFilterRefs(ShapeMatter.CD_SHAPE, ShapeRightsMatter.class, new
DoRef() {

    @Override
    public boolean ref(MatterReferenceAbstract r) {
        ret.add ((ShapeRightsMatter)r.getMatter());
        return true;
    }
});
```

To handle references an instance of DoRef interface is created which has ref method. In this example the ref method gets the matter which that reference is referenced to and after that got matter is written into the list ret. To clear up briefly, each right matter has a field named CD_SHAPE, which has cdMatter of ShapeMatter matter.

If a reference field of a matter refers to another matter, it is allowed to work with such fields as with ordinary data fields by using getValue or setValue methods. These methods get and return Long value which stands for a code of matter which that reference is referenced to by this value. There are also getValueRef method in MatterFieldRef and MatterFieldRefCount reference types which immediately return an associated matter or null if the value is zero.

Example 1:

```
Matter m;
...
Long cd = m.getValue (NAME);
Matter M2 = MatterCahce.getMatter (cd);
```

In this example Long value is got from the reference field which means cdMatter of the matter and then the matter is loaded from the cache.

Example 2:

```
Matter m;
...
Matter M2 = m.getValueRef (NAME);
```

In this example we get the matter from the reference filed at once.

4.3.3 Saving references

Field's references are saved to the database after committing the transaction or by calling putToBase method. Direct references aren't saved to the database but only change a matter in the cache. The storing of a matter having changed references is performed in accordance with the timer. By default it is one time per hour or while GlassFish is stopping. But if it is off-nominal stopping, unsaved

references will be lost. Lost references can be restored with the help of matter administrator tool (see 7.5).

4.4 Persistence Basics (CRUD).

In the previous topics we have discussed matters, and now we need to come back to general issues of working with data. As stated above, all MatterBase data are kept in matters that are heirs of Matter class, more specifically, in their fields. The field number is only restricted by the available memory size. Data fields represent concrete classes (see 4.2.1). Each instance of a matter class can have a various set of fields. A matter is charged with data validation.

MatterBase tries to keep maximum quantity of matters in the cache. That's why operations with matter are hastened as much as possible, as the probability of having the required matter in the cache is high, especially if this one was used recently. If memory comes to an end, MatterBase will run cleaning of the cache: at first it will remove those matters which weren't used fro a long time period.

4.4.1 Creating

A matter can be created only by using matter factory MatterRegister. It's only possible to create matters which were registered in the MatterBase. If you are using NetBeans, the matter registration code is created automatically during the project compilation (see 4.10).

MatterBase allows to create either a normal matter or a temporary one. Using temporary matters is useful in case of operational storing and data copying. Temporary matters don't have a unique matter code, they aren't stored in the matter cache and can't have references.

4.4.1.1 MatterRegister factory's methods to create new matters

```
/**
 * Creates a new matter.
 * @param nameMatter matter class name.
 * @param mt transaction with which the matter is created
 * @return new matter of a specified class
 * @throws ShapeMalfunction
 */
public static Matter getNewMatter (String nameMatter, MatterTransaction mt)

/**
 * Creates a new matter.
 * @param nameMatter matter class name. The transaction is automatically
created.
```

```
 * It can be obtained by calling getMatterTransaction method in the created
matter.
 * @return new matter of a specified class
 */
public static Matter getNewMatter (String nameMatter)

/**
 * Creates a new matter.
 * @param c matter class whose instance to be created. The transaction is
created automatically.
 * It can be obtained by calling getMatterTransaction method in the created
matter.
 * @return new matter of a specified class
 */
public static <T extends Matter> T getNewMatter (Class<T> c)

/**
 * Creates a new matter.
 * @param c matter class whose instance to be created
 * @param mt transaction which the matter will be created in
 * @return new matter of a specified class
 */
public static <T extends Matter> T getNewMatter (Class<T> c,
MatterTransaction mt)
```

4.4.2 Saving and updating

All changes in matter fields or matter references take place in a transaction. To save changes it's necessary to call MatterTransaction.commitTransaction() method. All matter changes modified by this transaction will be saved. Another way is to call matter putToBase() method. In this case, this method will find matter transaction and execute MatterTransaction.commitTransaction() method. If you call putToBase() of a certain matter, all matter changes from the same transaction will be saved here. This is quite useful: for example, if you need to change a reference field, the changes of the matter which that matter refers to will be included to the same transaction automatically. It is just enough to call putToBase method to save all the changes correctly.

To change matter data it is necessary to call one of the version of matter setValue method. All the possible versions of that method have two mandatory parameters: field name and it's new value. If the field isn't found, createNewMatterField method will be called which can create the missing field. This method is used for those matters for which the total number of fields is uncertain in advance. createNewMatterField needs to be overriden in a specific matter to avoid returning null value. If setValue method doesn't find the required field, it will be an exception.

```
// Calling factory method to create a new matter
```

```
Matter m = MatterRegister.getNewMatter(MshMatter.class.getName());
// Changing one matter field
m.setValue ("NAME", "kg");
// Saving the matter to MatterBase
m.putToBase();
```

4.4.3 Deleting

Removing a matter can be made by calling Matter.remove() method. But if a matter has references that refer to it, there will be an exception. To save such changes you need to commit the transaction where the matter has been removed. To do this just call MatterTransaction.commitTransaction() or putToBase() matter method. The calling of putToBase() looks unexpected in case matter removing, but actually it's a commiting of the transaction. Both of these approaches are interchangeable.

4.4.4 Understanding cascading updates and deletes

It is simple to organize cascading update or deletion in MatterBase. MatterBase has following methods:

```
protected boolean fieldChanged (MatterFieldSystem f, Object v,
MatterTransaction mt)
```

It's called while changing a specific field.

```
public void onCreate(MatterTransaction mt)
```

It's called after matter creation.

```
protected void onCommit(DBConnectionList dbConnectionList,
MatterTransaction mt)
```

It's called while performing saving to the database.

```
protected void onRemove(MatterTransaction mt)
```

It's called while removing.

```
protected void onChange(MatterTransaction mt)
```

It's called while changing matter field.

It is possible to override any of them in heirs and implement necessary actions connected with removing or changing. By default it is calling the following methods in onRemove:

```
removeFieldRefs(mt)
```

Removes references from its fields to other matters. This method must be called always.

```
removeRefs(mt, false, false)
```

Removes direct references. If you change method parameters, the method will be able to remove references that refer to the matter being removed by setting zero value (0). Classic relational databases don't have such feature as a rule. If the removing matter has references that refer to it, it won't be removed. an exception about reference misusing will be thrown.

4.5 All methods of the Matter class

```
/** Must return a unique code of the matter class */
public abstract short getCode();

/** Must return a new matter instance */
public abstract Matter getNewClone();

/**
 * Saves the matter to the database. Actually calls commitTransaction
method of the transaction object if it exists.
 * The method is slightly faster than without appSrv parameter.
 * @param appSrv application
 * @throws Exception
 */
public synchronized void putToBase(AppSrv appSrv)

/**
 * Saves the matter to the database. Actually calls commitTransaction
method of the transaction object if it exists.
 * @throws Exception
 */
public void putToBase()

/**
 * Creates fields of the current matter and initializes them in case they
haven't been initialized yet.
 * The matter is initialized as temporary.
 */
public void initFields()

/**
 * Prepares a temporary matter to save to the database. Starts the
transaction, the matter gets unique code and identifier and it saves to the
cache.
 * Field values aren't changed. If the matter is temporary, nothing will
happen.
 * @param mt transaction
 */
public synchronized void prepareTmp(MatterTransaction mt)

/**
 * Returns a new instance of VersionMatter class which is oriented to
describe the version of the current matter.
```

```
 * By default, returns null, i.e. versions are not supported. The method
must be overriden in case the matter must have versions.
 */
protected Matter getNewVersionMatter()

/**
 * Returns field name of the reference to the descriptor of the matter
version.
 * By default, returns null, i.e. versions are not supported. The method
must be overriden in case the matter must have versions.
 * This method must return the name of the field which is in the main
matter.
 */
protected String getVersionNameField()

/**
 * Returns true if the matter supports versions.
 */
public boolean isHaveVersion()

/**
 * Returns the matter describing the version of the current matter.
 */
public Matter getVersionMatter()

/**
 * Returns the active version of the matter of the given date in case
versions are supported by the current matter.
 * @param dt date
 */
public Matter getVerOnDate (Timestamp dt)

/**
 * Returns the last active version of the matter of the given date in case
versions are supported by the current matter.
 */
public Matter getVerLastActive ()

/**
 * Returns the last version of the matter of the given date in case
versions are supported by the current matter.
 */
public Matter getVerLast()

/**
 * Fills all its fields with values from fields of m matter, except for the
version field. If a field doesn't exist in m, sets null.
 * @param m source matter
 */
public synchronized void assign (Matter m)

/**
```

```
 * Fills the default fields in the heirs.
 */
public abstract void fillDefaultFields(MatterFields fields);

/**
 * It's used in the heirs to use in space-saving mode in the database. If
it returns false, default values of fields won't be saved in the database.
 * The matter requires the double amount of memory. The method can be
overriden in the heirs in order to change the value.
 */
public boolean isSaveDefaultValues()

/**
 * Returns the field value, depending on the passed r object. r can be
MatterTransaction or AppSrv. If the matter is being edited and the request
has come from another application, the previous data will be returned.
 * If the request has come from the application that has started the
transaction, the data from the clone will be returned. The method isn't
synchronized.
 * @param nameField field name which value to be returned
 * @param r value of the determining object. MatterTransaction or AppSrv
 * @return field value. If the field is not found, returns null
 */
protected Object getValueInternal (String nameField, Object r)

/**
 * Returns old field value, i. e. before field modification.
 * @param nameField field name which value to be returned
 */
public Object getOldValue (String nameField)

/**
 * Returns the field value subject to the current application.
 * @param nameField field name which value to be returned
 * @param appSrv application for which we get the value
 * @return field value. If the field is not found, returns null
 */
public Object getValue (String nameField, AppSrv appSrv)

/**
 * Returns the field value subject to the transaction.
 * @param nameField field name which value to be returned
 * @param mt transaction that is accounted in getting data
 * @return field value. If the field is not found, returns null
 */
public Object getValue (String nameField, MatterTransaction mt) {
    return getValueInternal (nameField, mt);
}

/**
 * Returns the field value subject to the current application.
 * @param nameField field name which value to be returned
```

```
     * @return field value. If the field is not found, returns null
     */
    public <T> T getValue (String nameField)

    /**
     * Returns the field value subject to the current application.
     * @param nameField field name which value to be returned
     * @param def default value in case the field value is null
     * @return field value. If the field is not found, returns null
     */
    public <T> T getValue(String valueName, T def)

    /**
     * Returns field value and blocks its change by other threads.
     * @param nameField field name which value to be returned
     * @param mt transaction for which field lock will be
     * @return
     */
    public <T> T getValueExclusive (String nameField, MatterTransaction mt)

    /**
     * Returns the field value subject to the current application.
     * If the value is null, the Exception will be thrown.
     * @param nameField field name which value to be returned
     * @param appSrv application for which we get the value
     * @return field value.
     */
    public long getLong (String nameField, AppSrv appSrv)

    /**
     * Returns the field value subject to the transaction.
     * If the value is null, the Exception will be thrown.
     * @param nameField field name which value to be returned
     * @param mt transaction that is accounted in getting data
     * @return field value.
     */
    public long getLong (String nameField, MatterTransaction mt)

    /**
     * Returns the field value subject to the current application.
     * If the value is null, the Exception will be thrown.
     * @param nameField field name which value to be returned
     * @return field value.
     */
    public long getLong (String nameField)

    /**
     * Returns the field value.
     * Def will be returned, if the field value is null.
     * @param nameField field name which value to be returned
     * @param def default value in case the field value is null
     * @return field value.
```

```
 */
public long getLong (String nameField, long def)

/**
 * Returns the field value.
 * Def will be returned, if the field value is null.
 * @param nameField field name which value to be returned
 * @param def default value in case the field value is null
 * @param mt transaction
 * @return field value.
 */
public long getLong (String nameField, long def, MatterTransaction mt)

/**
 * Returns the field value subject to the current application.
 * If the value is null, the Exception will be thrown.
 * @param nameField field name which value to be returned
 * @return field value.
 */
public double getDouble (String nameField)

/**
 * Returns the field value subject to the current application.
 * @param nameField field name which value to be returned
 * @param def default value in case the field value is null
 * @return field value.
 */
public double getDouble (String nameField, double def)

/**
 * Returns the field value subject to rc application.
 * @param nameField field name which value to be returned
 * @param def default value in case the field value is null
 * @param rc application
 * @return field value.
 */
public double getDouble (String nameField, double def, AppSrv rc)

/**
 * Returns the field value subject to the transaction.
 * @param nameField field name which value to be returned
 * @param def default value in case the field value is null
 * @param mt transaction
 * @return field value.
 */
public double getDouble (String nameField, double def, MatterTransaction
mt)

/**
 * Returns the field value.
 * If the value is null, the Exception will be thrown.
 * @param nameField field name which value to be returned
```

```
     * @return field value.
     */
    public boolean getBoolean (String nameField)

    /**
     * Returns the field value subject to the current application.
     * @param nameField field name which value to be returned
     * @param def default value in case the field value is null
     * @return field value.
     */
    public boolean getBoolean (String nameField, boolean def)

    /**
     * Returns the field value subject to the transaction.
     * @param nameField field name which value to be returned
     * @param def default value in case the field value is null
     * @param mt transaction
     * @return field value.
     */
    public boolean getBoolean (String nameField, boolean def, MatterTransaction
    mt)

    /**
     * Returns the field value subject to appSrv application.
     * @param nameField field name which value to be returned
     * @param appSrv application
     * @return field value.
     */
    public String getString (String nameField, AppSrv appSrv)

    /**
     * Returns the field value subject to the transaction.
     * @param nameField field name which value to be returned
     * @param mt transaction that is accounted in getting data
     * @return field value.
     */
    public String getString (String nameField, MatterTransaction mt)

    /**
     * Returns the field value subject to the current application.
     * @param nameField field name which value to be returned
     * @return field value.
     */
    public String getString (String nameField)

    /**
     * Returns the field value subject to the transaction.
     * @param nameField field name which value to be returned
     * @param mt transaction that is accounted in getting data
     * @param def default value in case the field value is null
     * @return field value.
     */
```

```
public String getString (String nameField, MatterTransaction mt, String
def)

/**
 * Returns the field value subject to the current application.
 * @param nameField field name which value to be returned
 * @param def default value in case the field value is null
 * @return field value.
 */
public String getString (String nameField, String def)

/**
 * Returns the matter from the reference filed for appSrv application.
 * @param nameField reference field name
 * @param appSrv application
 */
public <T extends Matter> T getMatterRef (String nameField, AppSrv appSrv)

/**
 * Returns the matter from the reference filed for the transaction.
 * @param nameField reference field name
 * @param mt transaction
 */
public <T extends Matter> T getMatterRef (String nameField,
MatterTransaction mt)

/**
 * Returns the matter from the reference filed.
 * @param nameField reference field name
 */
public <T extends Matter> T getMatterRef (String nameField) {
    return getMatterRefInternal (nameField, ActiveContexts.getCurAppSrv());
}

/**
 * Returns true in case bit(s) is set to the field with specified name.
 * Field type must be long otherwise Exception will be thrown.
 * @param name field name
 * @param v bits values.
 */
public boolean isBit (String name, long v)

/**
 * Sets v bit to the field with specified name.
 * Field type must be long otherwise Exception will be thrown.
 * @param name field name
 * @param v bits values.
 */
public void setBit (String name, long v)

/**
 * Clears v bit value in the specified field.
```

```
 * Field type must be long otherwise Exception will be thrown.
 * @param name field name
 * @param v bits values.
 */
public void clearBit (String name, long v)

/**
 * Sets or clears v bit value to the field with specified name depending on f
parameter.
 * Field type must be long otherwise Exception will be thrown.
 * @param name field name
 * @param v bits values
 * @param f if it's true - set, otherwise clear
 */
public void setBit (String name, long v, boolean f)

/**
 * Sets field value for the current application. After setting value calls
onChange method.
 * @param nameField field name
 * @param v value
 */
public void setValue (String nameField, Object v) {
    MatterTransaction mt = initTransaction(ActiveContexts.getCurAppSrv());
    setValue (nameField, v, mt);
}

/**
 * Sets field value for mt transaction. After setting value calls onChange
method.
 * @param nameField field name
 * @param v value
 * @param mt transaction
 */
public void setValue (String nameField, Object v, MatterTransaction mt)

/**
 * Setting the values to the fields for the current application. After
setting all the field values calls onChange only once.
 * The number of field names must be the same as the number of its values.
 * @param nameFields field names array
 * @param v field values array
 */
public void setValues (String[] nameFields, Object[] v)

/**
 * Setting the values to the fields for mt transaction.
 * After setting all the field values calls onChange only once.
 * The number of field names must be the same as the number of its values.
 * @param nameFields field names array
 * @param v field values array
 * @param mt transaction
```

```
 */
public void setValues (String[] nameFields, Object[] v,  MatterTransaction
mt)

/**
 * Sets the value to f field and it will return true, if the setting is
successful. The method can be overriden to control set values to the fields.
 * The method can modify other fields by using setValue method but it is
necessary to provide a mechanism for eliminating recursion because setValue
method calls fieldChanged method.
 * @param f field you want to set
 * @param v field value
 */
protected boolean fieldChanged (MatterFieldSystem f, Object v,
MatterTransaction mt)

/**
 * Creates a new field with nameField name. By default, the method does
nothing, i.e. returns null. If you need a matter with indefinite number of
fields, you have to override this method.
 * @param nameField field name to be created
 * @param v value for this field
 * @return
 */
public MatterFieldSystem createNewMatterField(String nameField, Object v)

/**
 * Removes the field with defined name subject to mt transaction.
 * @param name name of removable field
 * @param mt transaction
 */
public void removeField (String name, MatterTransaction mt)

/**
 * Returns general information about the matter. As a rule corresponds to
one or more matter fields.
 * For example, this method would return the name of measurement unit in
measurement unit matter.
 */
public String getInfo()

/**
 * Returns matter code + the value of getInfo method.
 */
public String toString()


/**
 * Returns the total number of matter fields.
 * @return
 */
public int getFieldsCount()
```

```
/**
 * Returns MatterField with the number i.
 * @param i field number
 */
public MatterField getField (int i)

/**
 * Updates the date of the last matter use.
 */
public void lastUseDateUpdate()

/**
 * Returns the transaction if it exists for the current application or null
if it doesn't exist but you can pass true to initNew parameter and the
transaction will be created.
 * @param initNew sign of creation transaction if it doesn't exist.
 * @return
 */
public synchronized MatterTransaction getMatterTransaction(boolean initNew)

/**
 * Returns true if the transaction for the matter has been started for the
current application.
 */
public boolean isTransaction()

/**
 * Returns the unique code of the matter. Each matter has its own unique
code.
 */
public long getCdMatter()

/**
 * Returns the unique identifier of the matter. Such identifier is the same
for all versions of the matter.
 */
public long getID_Matter()

/**
 * Returns the date and time of last matter usage.
 */
public long getLastUseDate()

/**
 * Returns true if the matter has been loaded from the database or has been
created and subsequently saved.
 */
public boolean isFromBase()


/**
```

```
 * Returns true if the matter has been removed from the cache. This can be
done by memory controlling manager.
 */
public boolean isCleared()

/**
 * Returns true if the matter is temporary.
 */
public boolean isTmp()


/**
 * Returns true if the matter has been removed.
 */
public boolean isDel()

/**
 * Returns the sign that the matter is outdated. This means that another
process has made changes to this matter and saved it to the database.
 */
public boolean isOld()

/**
 * Returns the sign that the matter is archival. This means that the data
of the service tables must be in the archive.
 */
public boolean isArc()


/**
 * Checks matter state and whether it must be reloaded to the cache. It is
possible in two cases: if the matter has been removed from the memory due
to long disuse or it has been changed and the new version has been created.
 * @return matter with the valid state.
 */
public final synchronized Matter reNew ()

/**
 * It's called after creating the matter. This method can be overriden for
matter initialization.
 */
public void onCreate(MatterTransaction mt) {};


/**
 * It's called before commit method. It can be used in user objects in
order to implement matter chained dependences.
 */
protected void onCommit(DBConnectionList dbConnectionList,
MatterTransaction mt) {}

/**
```

```
 * It's called when deleting the matter. Can be used in order to implement
cascade deleting. By default, the method implements removing references
from matter fields and direct references.
 * @param mt transaction
 */
protected void onRemove(MatterTransaction mt)

/**
 * It's called when a field starts changing.
 * @param mt transaction
 */
protected void onChange(MatterTransaction mt) {};

/**
 * Removes the matter. The removal will happen only after calling
putToBase. This method calls onRemote method to remove cross-references and
assign removable status to the matter.
 * If an exception occurs, it's necessary to call rollBack method in the
transaction object. If the procedure has been successful, you need to call
commit or putToBase method. If the removable matter has references from
other matters, there will be the
 * exception.
 */
public void remove(MatterTransaction mt)

/**
 * Starts the transaction and remove the matter. If it's without
exceptions, the changes will be saved to the database.
 * If na exception occurs, rollbackTransqaction calls. If the transaction
has been already started, the exception will be thrown.
 * If the removable matter has matters from other matters, the exception
will be thrown as well.
 */
public synchronized void remove()

/**
 * Removes field references from all the matters that refer to this matter.
 * By default, calls from onRemove method, i.e. the matter must be blocked
for changes and a transaction must be started.
 */
protected void removeFieldRefs(MatterTransaction mt)

/**
 * Removes direct references, i.e. references without using matter fields,
and field references that refer to the current matter from other matters.
 * If it's a field reference then while deleting them the reference field of
the original matter will be set zero value (0). This feature works
 * only if delFromFieldRef parameter has true value.
 * By default, onRemove method call removing only for direct references.
 * @param mt transaction to performing deletion
 * @param delFromFieldRef if it's true, all the references will be removed
in all the matters that refers to the removable matter.
```

```
 * The according reference fields, which refers to the removable matter,
will be set zero value, except for count type references.
 * @param delFromFieldRefCount if it's true, all the references in all the
matters that refers to the removable matter will be set zero value.
 */
protected void removeRefs(final MatterTransaction mt, boolean
delFromFieldRef, boolean delFromFieldRefCount)

/**
 * Removes all the matters of the specified class that have reference field
referring to the matter being removed.
 * For example, this method makes it easy to remove the lines of a document
while removing its head from the document.
 * In fact, this method implements the most common case of cascade
deletion.
 * @param mt transaction to performing deletion
 */
protected void removeAllFieldRefMatters(MatterTransaction mt)

/**
 * Removes all the matters of the specified class that have reference field
referring to the matter being removed.
 * For example, this method makes it easy to remove the lines of a document
while removing its head from the document.
 * In fact, this method implements the most common case of cascade
deletion.
 * @param mt transaction to performing deletion
 * @param fieldRefName field name
 * @param m matter class that referes to this matter
 */
protected void removeFieldRefMatters (MatterTransaction mt, String
fieldRefName, Class<?extends Matter> m)

/**
 * Finds specified reference and removes the matters that are refered to
this reference.
 * @param nameRef reference name
 * @param typeRef reference type
 */
protected void removeLinkedByRefsMatters (MatterTransaction mt, String
nameRef, MatterReferenceType typeRef)

/**
 * Returns the number of references that refer to the matter.
 * @return
 */
public int getRefsCount()

/**
 * Returns a reference with i index. The usage is not recommended.
 * The method doesn't consider the current changes. Use doFilterRefs.
 * @param i reference index
```

```
 */
public MatterReferenceAbstract getRef (int i)

/**
 * Reference filter by defined criteria excluding transaction changes.
 * @param fieldRefName name of reference field which we are looking for the
references
 * @param m matter class that contains reference field
 * @param doRef handler
 */
public <T extends Matter> MatterReferenceAbstract doFilterRefs  (
String fieldRefName, Class<T> m, DoRef<T> doRef)

/**
 * Reference filter by defined criteria as amended mt transaction.
 * @param fieldRefName name of reference field which we are looking for the
references
 * @param m matter class that contains reference field
 * @param doRef handler
 * @param mt incompleted transaction subject to references are processing
 */
public <T extends Matter> MatterReferenceAbstract doFilterRefs (String
fieldRefName, Class<T> m,  DoRef<T> doRef, MatterTransaction mt)

/**
 * Reference filter of the specified type excluding transaction changes.
 * @param typeRef reference type
 * @param doRef handler
 */
public <T extends Matter> MatterReferenceAbstract doFilterRefs
(MatterReferenceType typeRef, DoRef<T> doRef)

/**
 * Reference filter between the two matter classes excluding transaction
changes.
 * The first class is the current instance the method of which is called.
 * @param c2 the second class
 * @param typeRef reference type
 * @param doRef handler
 */
public <T extends Matter> MatterReferenceAbstract doFilterRefs  (Class<T>
c2,
        MatterReferenceType typeRef, DoRef<T> doRef)

/**
 * Reference filter between the two matter classes as amended mt
transaction.
 * The first class is the current instance the method of which is called.
 * @param c2 the second class
 * @param typeRef reference type
 * @param doRef handler
 * @param mt incompleted transaction subject to references are processing
```

```
 * Returns the reference for which the hadler returned false value.
 * Use to implement references seeking.
 */
public <T extends Matter> MatterReferenceAbstract doFilterRefs  (Class<T>
c2,
        MatterReferenceType typeRef, DoRef<T> doRef, MatterTransaction mt)

/**
 * Traversals all matter fields excluding transaction changes.
 * @param chdFieldOnly if it's true, the only modified fields will be
processing only
 * @param doFields handler
 */
public void doFields  (boolean chdFieldOnly, DoField doFields)

/**
 * Traversals all matter fields as amended mt transaction.
 * @param chdFieldOnly if it's true, the only modified fields will be
processing only
 * @param doFields handler
 * @param mt transaction
 */
public void doFields  (boolean chdFieldOnly, DoField doFields,
MatterTransaction mt)

/**
 * Returns true in case the field has been changed by the current
application
 * or it is a new matter.
 * @param f field
 */
public boolean isFieldChanged (MatterField f)

/**
 * Returns true in case f field has been changed by mt transaction or it is
a new matter.
 * @param mt transaction
 */
public boolean isFieldChanged (MatterField f, MatterTransaction mt)

/**
 * Returns true in case the field has been changed or has been created by
the current application.
 */
public boolean isFieldChanged (String nameField)

/**
 * Returns the field with the specified name. The value can not be changed.
 * @param nameField field name
 * @return found field or null
 */
protected MatterField getField(String nameField) throws MatterFieldUnknow
```

```
/**
 * Starts mt transaction for the matter.
 * @param mt transaction
 */
public synchronized void startTransaction(MatterTransaction mt) throws
MatterDeadlock


/**
 * Inserting data into the service table.
 * By default, it returns false and does nothing else or returns true in
case the standard processing is canceled but
 * processing has to do this method itself.
 * The standard processing inserts one record into the service tables for
each matter.
 * @param st a service table for which the operation is performed
 * @param dbConnLst connection list
 * @param aliasTmp database alias
 */
public boolean insServiceTable (ServiceTable st, DBConnectionList
dbConnLst, String aliasTmp)

/**
 * Changing the data in the service table.
 * By default, it returns false and does nothing else or returns true in
case the standard processing is canceled but
 * processing has to do this method itself.
 * The standard processing changes all the modified fields of the record in
service tables in case matter appropriate fields have been modified.
 * @param st a service table for which the operation is performed
 * @param prevMatter matter before change
 * @param dbConnLst connection list
 * @param aliasTmp database alias
 */
public boolean updServiceTable (ServiceTable st, Matter prevMatter,
DBConnectionList dbConnLst, String aliasTmp)

/**
 * Deletes data from the service table.
 * By default, it returns false and does nothing else or returns true in
case the standard processing is canceled but
 * processing has to do this method itself. The standard processing removes
the record from matter service table.
 * @param st a service table for which the operation is performed
 * @param dbConnLst connection list
 * @param aliasTmp database alias
 */
public boolean delServiceTable (ServiceTable st, DBConnectionList
dbConnLst, String aliasTmp)

/**
```

```
 * Returns the value of the service table.
 * This method cat be used by the heirs in order to return composite values
to to the service table.
 * By default, seeks the field with nameField name and returns its value.
 * @param nameTable table name for which you want to return the value of
the field.
 * @param nameField field name of the service table the value of which you
want to return.
 */
public Object getValueForService(String nameTable, String nameField)

/**
 * Returns the value of the primary key of service table.
 * By default, returns a matter unique code.
 * @param nameTable
 */
public Object getPrimaryKeyForService(String nameTable)

/**
 * Returns the field name as JDBC type.
 */
public String getFieldJDBType (String nameField) {

/**
 * Retains an array of service tables descriptors.
 * By default, returns null.
 * If a matter has service tables, this method must return them.
 * The declaration of service tables must be static.
 * @return
 */
public ServiceTable[] getServiceTableArray()

/**
 * Sets the value of an additional parameter to the reference of typeRef
type that refers to m2 matter.
 * If the reference is found, it returns true. The reference value is set
to the response matter of m2 matter as well.
 * Values of the reference are set only for the following reference types:
 * MatterReferenceType.DOWN, MatterReferenceType.UP,
 * MatterReferenceType.NOTHING,  MatterReferenceType.DOWNLINK
 * @param m2 matter of which are we looking for the reference.
 * @param typeRef desired reference type
 * @param mt transaction; if it is not specified, it will be created and
committed inside the method
 * @param valueRef set value
 * @return
 */
public boolean setValueRef (Matter m2,
                        MatterReferenceType typeRef,
                        MatterTransaction mt,
                        SimpleValue valueRef)
```

```
/**
 * Seeks a reference with specified parameters.
 * @param m2 desired matter of the reference.
 * @param typeRef reference type; if it's null, then it's any type
 * @param mt transaction
 * @return found reference or null otherwise
 */
public MatterReferenceAbstract getRef (
                                final Matter m2,
                                final MatterReferenceType typeRef,
                                final MatterTransaction mt) {


/**
 * Compares two matters only by their unique codes.
 * @param obj
 * @return
 */
public int compareTo(Object obj)

/**
 * Matter hash code. To compute the hash there are used only matter codes.
 */
public int hashCode()

/**
 * Returns true if matters are equal.
 * To check the equality there are used only matter codes.
 * @param obj
 */
public boolean equals(Object obj)

/**
 * Sets the new value for the reference that is between two specified
matters.
 * @param cdMatter1 matter code of the first matter
 * @param cdMatter2 matter code of the second matter
 * @param str new value
 */
static public boolean updateParamValue(Long cdMatter1, Long cdMatter2,
String str)


/**
 * Adds a new field to the field list.
 * It's used only in fillDefaultFields method to create a matter field list.
 * @param fields
 * @param f
 */
public static void addField (MatterFields fields, MatterFieldSystem f)
```

4.6 Locking, Deadlock

MatterBase doesn't have explicit locks. There are implicit locks and they occur in the following cases: In cluster mode a matter transaction can be started only in one cluster server. Trying to start a transaction in another server, there will be an error message. The same matter can have several started transactions. The exception will occur only if two transactions try to change the same field. If the second transaction tries to change the field in parallel with the first one, the second transaction will get an error message on condition that the waiting time has run out. The waiting time is set in JEDFConfig.xml (see 3.1).

4.7 Meaning of service tables

Service tables extend basic capabilities for operating with matters. The main purpose of the service tables is to search matter codes (cdMatters) for further loading to the cache. MatterBase does all the work automatically: insertion, removing and changing records of a certain service table. Matter has capability to organize its own algorithm of saving to a service table. By default each matter has one record in its own service table. Here is an example of the measurement unit matter with a standard service table (see 4.1):

```
@Override
public ServiceTable[] getServiceTableArray() {
    return st;
}

private static ServiceTable[] st = {new ServiceTable (SERVICE_TABLE_NAME,
"",
        new String[]{MSH},
        null,
        new String[]{"alter table mshref add unique (msh)"},
        new MshMatter())
};
```

In this code's listing getServiceTableArray method is overriden. This method returns an array of matter service tables. The array itself is declared as a static class member since all instances of this matter have the same service tables. It's standard usage of service tables. A service table is useful in case of searching and forbidding to create matters with identical title. In order to do this here is a unique index for the title of measurement units.

For instance, you have created a matter with «kg» title and let's say cdMatter is 210. While MatterBase was creating this matter there were the following steps: the matter itself was saved to Matter table in the inner system format and a record was inserted by «insert into MshRef (cdMatter, msh) values (210, «kg»)» command

to MshRef table («MshRef» derived from MshMatter matter definition).

There is also one important feature related to service tables. Service tables contain only those records that have been committed (by commitTransaction or putToBase). Thus if you change or create matters witout committing them, you won't be able to see any actual data about these matters. It's because uncommitted changes don't influence a service table.

4.7.1 Searching with service table

To search in service tables there is a special ServiceMatterFind class. Here is a listing where the search for a measurement unit matter is performed. After the successful searh the matter is loaded. If the matter isn't found findMatter method will return null.

```
Matter m = ServiceMatterFind.findMatter(MshMatter.class.getName(),
MshMatter.SERVICE_TABLE_NAME, MshMatter.MSH, "kg");
```

4.7.1.1 ServiceMatterFind methods.

```
/**
 * Seeks the matter. If there are several ones, it returns the first matter
found.
 * @param className matter class name to be searched
 * @param serviceTableName service table name in which we seek the matter
 * @param findFieldName name of the field in which we are seeking
 * @param findValue searching value
 * @return Matter found matter or null
 */

public static Matter findMatter(
        String className, String serviceTableName, String findFieldName,
Object findValue)


/**
 * Seeks the matter. If there are several ones, it returns the first matter
found.
 * @param className matter class name to be searched
 * @param srcInArc sign of the need to search in the archive table
 * @param serviceTableName service table name in which we seek the matter
 * @param findFieldName name of the field in which we are seeking
 * @param findValue searching value
 * @return Matter found matter or null
 */

public static Matter findMatter(
        String className, boolean srcInArc, String serviceTableName, String
findFieldName, Object findValue)
```

```
/**
 * Seeks the matter in different fields.
 * If there are several ones, it returns the first matter found.
 * @param className matter class name to be searched
 * @param serviceTableName service table name in which we seek the matter
 * @param fv array of pairs with field name / value
 * @return Matter found matter or null
 */

public static Matter findMatterM(
        String className,
        String serviceTableName, Object... fv)

/**
 * Seeks the matter in different fields.
 * If there are several ones, it returns the first matter found.
 * @param className matter class name to be searched
 * @param srcInArc sign of the need to search in the archive table
 * @param serviceTableName service table name in which we seek the matter
 * @param fv array of pairs with field name / value
 * @return Matter found matter or null
 */

public static Matter findMatterM(
        String className,
        boolean srcInArc,
        String serviceTableName, Object... fv)


/**
 * Seeks the matter. If there are several ones, it returns the first matter
found.
 * @param className matter class name to be searched
 * @param serviceTableName service table name in which we seek the matter
 * @param findFieldName name of the field in which we are seeking
 * @param findValue searching value
 * @param dbCon database connection
 * @return Matter found matter or null
 */

public static Matter findMatter(
        String className,
        String serviceTableName, String cdMatterFieldName,
        String findFieldName, Object findValue, DBConnection dbCon)

/**
 * Seeks the matter. If there are several ones, it returns the first matter
found.
 * @param className matter class name to be searched
 * @param srcInArc sign of the need to search in the archive table
 * @param serviceTableName service table name in which we seek the matter
```

```
     * @param findFieldName name of the field in which we are seeking
     * @param findValue searching value
     * @param dbCon database connection
     * @return Matter found matter or null
     */

    public static Matter findMatter(
            String className,
            boolean srcInArc,
            String serviceTableName, String cdMatterFieldName,
            String findFieldName, Object findValue, DBConnection dbCon)

    /**
     * Seeks matters.
     * @param serviceTableName service table name in which we seek the matter
     * @param findFieldName name of the field in which we are seeking
     * @param findValue searching value
     * @return list of found matters or null
     * @throws Exception
     */

    public static List<Matter> findMatterList(
            String className,
            String serviceTableName,
            String findFieldName, Object findValue)
            throws Exception {
        return (List<Matter>)findMatterInternal(
                className, true, serviceTableName, Matter.CD_MATTER,
                false, null, findFieldName, findValue);
    }

    /**
     * Seeks matters.
     * @param className matter class name to be searched
     * @param srcInArc sign of the need to search in the archive table
     * @param serviceTableName service table name in which we seek the matter
     * @param findFieldName name of the field in which we are seeking
     * @param findValue searching value
     * @return List<Matter> list of found matters or null
     */

    public static List<Matter> findMatterList(
            String className,
            boolean srcInArc,
            String serviceTableName,
            String findFieldName, Object findValue)

    /**
     * Seeks matters.
     * @param className matter class name to be searched
     * @param serviceTableName service table name in which we seek the matter
     * @param findFieldName name of the field in which we are seeking
```

```
     * @param findValue searching value
     * @param dbCon database connection
     * @return List<Matter> list of found matters or null
     */
    public static List<Matter> findMatterList(
            String className,
            String serviceTableName, String cdMatterFieldName,
            String findFieldName, Object findValue, DBConnection dbcon)

    /**
     * Seeks matters.
     * @param className matter class name to be searched
     * @param srcInArc sign of the need to search in the archive table
     * @param serviceTableName service table name in which we seek the matter
     * @param findFieldName name of the field in which we are seeking
     * @param findValue searching value
     * @param dbCon database connection
     * @return List<Matter> list of found matters or null
     */
    public static List<Matter> findMatterList(
            String className,
            boolean srcInArc,
            String serviceTableName, String cdMatterFieldName,
            String findFieldName, Object findValue, DBConnection dbcon)
```

4.7.2 Service table methods

Here are constructors and methods of ServiceTable class.

```
    /**
     * The first constructor of service table descriptor.
     * @param tableName sevice table name in the database
     * @param primaryKeyName servie table primary key; if not specified it's
    cdMatter
     * @param matterFields list of matter field names to be inserted into the
    service table
     * @param tableFields list of service table field names that must correspond
    to matter fields exactly.
     * If it isn't specified, the field names will be the same as in the matter.
     * @param addSQL an additional list of queries to be executed while
    creating the table.
     * @param m an instance of the matter for which this service table will be
    used.
     */
    public ServiceTable(String tableName, String primaryKeyName,
            String[] matterFields, String[] tableFields,
            String[] addSql, Matter m)

    /**
     * The second constructor of service table descriptor.
     * @param alias - database alias, which will be located in the service
```

```
table.
 * If an alias isn't specified, the service table will be created in
JEDFSystems. This option allows you to spread workload on different
physical databases and make them the optimal size.
 * @param aliasArc - database alias for archival matters. Archival matter
is a matter in which isArc method returns true. By default, the date of
compare change with
 * the date of matter transfer to the archive from the service table. To
transfer to the archive, use the administration tool.
 * @param tableName - service table name
 * @param primaryKeyName servie table primary key; if not specified it's
cdMatter
 * @param matterFields list of matter field names to be inserted into the
service table
 * @param tableFields list of service table field names that must correspond
to matter fields exactly.
 * If it isn't specified, the field names will be the same as in the matter.
 * @param addSQL an additional list of queries to be executed while
creating the table.
 * @param m an instance of the matter for which this service table will be
used.
 */

public ServiceTable(String alias, String aliasArc,
        long dateTpArc,
        String tableName, String primaryKeyName,
        String[] matterFields, String[] tableFields,
        String[] addSql, Matter m)

/**
 * Checks the current location of the archival record in accordance with
Matter.isArc method.
 * If it doesn't conform, it will change to the right.
 */
public void updateArc(Matter m, DBConnectionList dbConLst)
```

4.7.3 Updating service tables

A standard algorithm of data updating in service tables presupposes that each matter corresponds to one record in a sevice table. Each column from a table corresponds to one field from a matter. If the matter is created, one record will be created with relevant columns in each service table of the matter. If values of matter fields are changed, values in relevant columns will be changed as well. If the matter is removed, the matter record will be removed from all matter service tables. These rules can be changed. For example, there is an accumulation service table where accumulation data is collected for several matters. All we need to do is to override the following methods:

```
/**
 * Retains the value of the service table field.
```

```
 * Can be used in heirs in order to return composite field values in the
service table.
 * By default, seeks a field with nameField and returns its value.
 * @param nameTable - table name for which you want to return the value of
the field.
 * @param nameField - field name of the service table the value of which you
want to return.
 * @return
 */
public Object getValueForService(String nameTable, String nameField)

/**
 * Inserting data into a service table.
 * By default, it returns false and does nothing else or returns true in
case standard processing is cancelled but
 * processing has to do this method itself.
 * Standard processing inserts for each matter one record into the service
table.
 * @param st a service table for which the operation is performed
 * @param dbConnLst connection list
 * @param aliasTmp database alias
 * @return
 */
public boolean insServiceTable (ServiceTable st, DBConnectionList
dbConnLst, String aliasTmp) {
    return false;
}
/**
 * Changing data in the service table.
 * By default, it returns false and does nothing else or returns true in
case the standard processing is canceled but
 * processing has to do this method itself.
 * Standard processing changes all the modified fields of the matter in the
service table.
 * @param st - a service table for which the operation is performed
 * @param prevMatter - a matter before the change
 * @param dbConnLst - connection list
 * @param aliasTmp - database alias
 * @return
 */
 public boolean updServiceTable (ServiceTable st, Matter prevMatter,
DBConnectionList dbConnLst, String aliasTmp)

/**
 * Deletes data from the service table.
 * By default, it returns false and does nothing else or returns true in
case the standard processing is canceled but
 * processing has to do this method itself.
 * standard processing removes a record in the service table that
corresponds to the matter.
 * @param st - a service table for which the operation is performed
 * @param dbConnLst - connection list
```

```
     * @param aliasTmp - database alias
     */
    public boolean delServiceTable (ServiceTable st, DBConnectionList
    dbConnLst, String aliasTmp)
```

4.8 Transactions in details

MatterBase has transactions but they are different from relational databases transactions. MatterBase transaction is a system class instance which has changes of matters for which this transaction has been started. All these changes can be saved or canceled by commitTransaction or rollbackTrancasction methods respectively. MatterBase doesn't have isolation levels like a relational database does. Dealing with matters in MatterBase resembles dealing with relational databases that support versioning of records but with some assumptions. Transactions can be two types: explicit and implicit. Each transaction has information about what application has created it. For one matter and for one application can be started only one transaction. The application is a system structure on JEDF server which corresponds to the first (main) frame running after login. For example, the administration tool will be the main frame.

4.8.1 Accessing matter data from the transaction

If process 1 changes field F in matter A, all the rest processes will see field F value prior to the changes of process 1. The changes will have been available to other processes only after commiting the transaction with the changes of matter A. If process 1 refers to field F in matter A after its changes but before performing commitTransaction method, it will see new set changes of field F as these changes are made by itself (see the example in 4.8.2).

4.8.2 Accessing matter data after committing transaction

MatterBase has the next rule: if you get a reference to a matter, the data of the matter will be permanent, except for outdated matter sign with isOld method. This happens if another process has changed this matter and saved it by performing commitTransaction method. Actually the following happens: a new copy of the matter was created, where all the changes were relocated and after that the new matter was put into the cache. The outdated sign is set to the original matter. This sign doesn't influence work with the matter, it's just an information sign. But the reference (pointer) is still pointing at the outdated matter. To get the actual matter you need to call reNew instance method or get the matter straight from the cache by calling MatterCache.getMatter.

Here is a clarifying example. Let's say that MatterBase has a matter with cdMatter

which is equal to 23 and having MSH field with value «kg». In this example we will change field value to «KG».

```
// Getting the matter with the code 23
Matter m = MatterCache.getMatter (23);
// Getting the value of MSH field
s = m.getValue ("MSH");
// The value of s variable is "kg"
// Sets the new value of MSH field. An automatic transaction is started.
m.setValue ("MSH", "KG");
// Getting the value of MSH field
s = m.getValue ("MSH");
// The value of s variable is KG because obtaining the value of the field is
made in the same transaction.
// We can see a new value
// The transaction is completed.
m.putToBase();
// Getting the value of MSH field
s = m.getValue ("MSH");
// variable s value = kg.
// As m refers to the same original matter and that value has become the
original as well.
// updating matter reference
m = m.reNew();
// or  m = MatterCache.getMatter (23);
// access MSH field
s = m.getValue ("MSH");
// the value of s variable = kg
```

From the stated above we can conclude that you shouldn't keep a reference (pointer) to a matter instance long as it need to be updated otherwise you'll have outdated data. Transactions should be as short as possible time and this is usually what exactly happens in real applications. For example, a user request comes to the application server, a required matter is got from the cache, changes are set to the matter (the transaction starts) and these changes are saved (the transaction commits). If you need this matter again you need to start with getting the matter from the cache.

4.8.3 Implicit transactions

An implicit transaction starts every time you change matter field values or references to it without explicitly declaring the transaction. In the example above there was an implicit transaction. The implicit transaction starts when you call m.setValue(«MSH», «KG»). Transaction itself can be obtained through the method m.getMatterTransaction(false). Implicit transactions are convenient for small changes in the matters, as you don't need to use the optional transaction

parameter in changing matter values. If you want to change several matters in a single transaction, you have to use this optional transaction parameter. Implicit transactions require a bit more CPU.

4.8.4 Explicit transactions

When you use an explicit transaction, you have to create it prior to matter changes. There are two main ways how to do it:

```
MatterTransaction mt = MatterTransactionFactory.createNewTransaction()
```

It is possible to create a transaction using MatterTransactionFactory class. All other possibilities also use this class anyway.

```
MatterTransaction mt = m.getMatterTransaction(true)
```

You can get the transaction through calling matter method. If the thransaction exists, it will be returned otherwise it will be created and started for this matter.

Further transaction mt must be passed as a parameter when changing the matter. There is an example of changing field's value similar to the example in 4.7.2 but using an implicit transaction.

```
// Gets the matter with the code 23
Matter m = MatterCache.getMatter (23);
// Creates a transaction.
MatterTransaction mt = MatterTransactionFactory.createNewTransaction()
// Sets the new value of MSH field using an explicit transaction.
m.setValue ("MSH", "KG", mt);
// Saving the matter - completion of the transaction occurs.
m.putToBase();
```

There is an equivalent alternative:

```
// Gets the matter with the code 23
Matter m = MatterCache.getMatter (23);
// Creates a transaction.
MatterTransaction mt = m.getMatterTransaction(true);
// Sets the new value of MSH field, using an explicit transaction.
m.setValue ("MSH", "KG", mt);
// Complition of the transaction with saving the matter
mt.commitTransaction();
```

4.9 Matter cache and database

Matter cache is one of the key MatterBase components. Matter Cache provides high performance with matters. When talking about loading or getting a matter is meant that the matter will be loaded into the cache, and it will be returned by reference (pointer). Loading is performed by getMatter method. This method quite

often met in the examples. Let's see:

```
Matter m = MatterCache.getMatter (23);
```

This means loading a matter from the cache whose cdMatter is 23. The cache works with matters only with their cdMatter code. This code can be obtained by using a service table, a reference field or reference lists.

4.9.1 Memory usage by Matter cache

To improve performance, the cache strives to keep matters in memory as long as possible. If there has been access to a matter, it will be kept in the cache until memory is full or the application server is stopped. Checking on the need to remove matters from the cache is performed every 5000 calls to the cache but more often than once every 20 seconds. First and foremost, matters which weren't used the longest will be removed.

4.9.2 All methods of MatterCache

```
/**
 * Returns the number of matters that are in the cache.
 */
public static int getMatterCashSize() {
    return map.size();
}

/**
 * Provides access to matters. Calls from reference classes and classes
that store direct references to the matters.
 * The method validates the matter and update it if necessary. Matter state
can be changed by another user or manager, who is removing matters from the
memory unused for a long time.
 * @param ma - initial matter.
 * @return matter
 */
public static Matter getMatter (Matter ma)

/**
 * Seeking the matter in the cache. If it is not found, loads it from the
database.
 * @param cdMatter - matter code.
 * @return the found matter.
 */
public static <T extends Matter> T getMatter (long cdMatter)

/**
* Retains all of the matters, wich are with modified references, stored in
the cache.
* Saving occurs only if more than saveIntervalRefs passed from the time of
```

```
the last call.
*/
public static void flushMatterRefs()

/**
* Retains all of the matters, wich are with modified references, stored in
the cache.
*/
public static void flushMatterRefsNow()

/**
* Retains all of the matters, wich are with modified references, stored in
the cache.
* Messages about an output process are stored in ps.
*/
public static void flushMatterRefsNow(PrintAndSave ps)


/**
 * Returns the amount of free memory in JVM.
 * @return long
 */
public static long getFreeMemory()

/**
 * Returns the amount of memory that can be used by JVM.
 * @return long
 */
public static long getTotalMemory()

/**
 * Returns the maximum amount of memory that can be used by JVM.
 * @return long
 */
public static long getMaxMemory()


/**
 * Retains all the matters made by the application to the database.
 * @param appsrv - application
 */
public static void putToBaseAppSrv (AppSrv appsrv)

/**
 * Undoes changes made by the application among all the matters.
 * @param appsrv - application
 */
public static void rollBackAppSrv (AppSrv appsrv)

/**
 * Returns saving interval of modified matters.
 */
```

```
public static long getSaveIntervalRefs()

/**
 * Changes saving interval of modified matters.
 */
public static void setSaveIntervalRefs(long saveIntervalRefs)


/**
 * Returns the database alias of the matter.
 * @param m - matter
 */
public static String getMatterBaseAlias(Matter m)

/**
 * Returns the database alias for the specified code of the matter.
 * @param cd - matter code.
 */
public static String getMatterBaseAlias(long cd)
```

4.10 New matter class registration

To work with a matter it needs to be registered. During the registration MatterBase reads information about matter class. After registering the new matter class it is allowed to perform all available actions such as instance creating, modifying and deleting. To register a matter, a specal line of code must be created and executed:

```
ret.add (new MatterReg ("su.jedf.metashaper.server.matter.userclasses.
MshMatter", new Integer(5004).shortValue()));
```

This line registers measurement unit matter. When you work in NetBeans matter registration code is automatically generated. Here is created a class in «generated-sources» directory which all the matter classes register in. For automatic generation of matter registration code you must use the annotation:

```
@MatterDcl(code = MshMatter.code)
public class MshMatter extends Matter {
  ...
}
```

Do not create your own matter registration code. Use this plugin. This simplifies programming and guarantees the correctness of matter registration in the system.

4.11 Matter versioning

MatterBase has built-in feature to create versions of a matter. For example, if a matter describes a document, such as an account, the document can have a few

versions in the system. All these versions of the document have the same matter identifier. To get this identifier just call getID_Matter matter method. To realize this feature you need to create a special matter class that extends VersionMatter matter class and to override getNewVersionMatter method in the document matter. This method will return new matter version. Further you need to save the matter code of the necessary version for further use.

Also you have to override getVersionNameField method which returns the name of the field that stores the code of the version matter.

There is an example describing this feature:

```
//Creates a cpecialized class of the version
protected Matter getNewVersionMatter() {
    if (getMatterTransaction(false) == null) {
        throw new MatterNoTransaction (this, MessageTexts.get ("incorrect
call"));
    }
    Matter m = MatterRegister.getNewMatter(SpecVersionMatter.class.
getName(),
            getMatterTransaction(false));
    setValue(CD_VERSION, m.getCdMatter());
    return m;
}

protected String getVersionNameField() {
    return Matter.CD_VERSION;
}
```

The fields of VersionMatter matter contain information about the source matter, the version number, the user that has created this version, the creation date of the version and the status of the version. Each version can have several states. States are described by another system matter – VersionSatetMatter. By default there are four states of matters versions: WORK, ACTIVE, ARCHIVE, DISABLED. The list of the states can be extended through the creation of its own state matter class which is their of the VersionStateMatter class.

4.12 Archiving service tables

MatterBase already has archiving feature. Archiving only relates to data of the service tables. Archiving means to divide the data that is often used (live data) and that is used rarely (archive data). The smaller the size of the live database, the faster are all its operations. If service table data can be archived, it is necessary to override isArch method that by default, returns false, which means the data of the service tables isn't archived. If the method returns true, then the data of

the service tables is in the archive. Usually as an archived sign you can use the difference between the current date and the date of the matter but you can write any other algorithm. For example, the transfer of annual accumulative service tables to the archive needs to be done once a year.

4.13 The database size optimization

All matters are stored in the sole table and consequently the operation speed of working with big files of the database and performing service maintenance can be unacceptable. To solve this problem MatterBase can distribute matters on different bases. By default there are recorded 50 million matters to each MatterBase database. This parameter is set in JEDFSystem.xml file. The system assumes aliases with a specified name. The first alias is named «JEDFSystem», each following is «JEDFSystem_n», where «n» - is an alias number that begins with 1. By default, all matters with codes less than 50 million are saved under «JEDFSystem» alias, matters with codes in range from 50 million up to 99 999 999 are saved under «JEDFSystem _1» alias and so on. Thus using the option «MattersInOneBase» you can change the distribution of matters on databases.

4.14 SQL performance optimizations

MatterBase provides one more feature it's a SQL optimizer. Even if you don't use matters at all, just using this optimizer you save resources of the database and accelerate executing queries. The optimizer consists of the following classes:

- DBSQLCache – the main class of the optimizer
- DBStatement – request to the server
- DBConnection – server connection
- DBConnectionList – list of connections to the server

4.14.1 DBSQLCache

This is the main class for handling SQL. There are two important methods:

- getPreparedStatement – returns prepared query. To this prepared query you can set parameters and execute it.
- getDBConnection – returns a connection. You can execute several queries. Connections enable to control transactions.

```
/**
 * Returns DBStatement with the required sql.
 * It is obligatory to call close method in the end.
```

```
 * @param alias database alias
 * @param sql query
 * The method is executed in a separate transaction and then immediately
commits transaction.
 * For direct transaction management use getDBConnection method.
 */
public static DBStatement getPreparedStatement (String alias, String sql)

/**
 * Returns DBConnection. It is obligatory to call close method in the end.
 * preSql, which from the database descriptor, will be executed before
returning DBConnection,
 * if its parameter has been changed or the method is called from another
application.
 * @param alias database alias
 * @return DBConnection
 */
public static DBConnection getDBConnection (String alias)
```

4.14.2 DBStatement

This class is designed for executing SQL. Queries were being prepared previously. After working with this class you have to call close method, otherwise resources won't be available for repeated usage. There are recommended form of calling with using AutoCloseable interface:

```
try (DBStatement ds1 = DBSQLCache.getPreparedStatement("JEDFSystem",
"Select * from MshRef")) {
 //other operations with ds1...
}
```

Some useful methods of the class:

```
/**
 * Returns prepared statement.
 */
@Override
public PreparedStatement getPreparedStatement() {
    whenUsed = System.currentTimeMillis();
    return preparedStatement;
}

/**
 * Changes the query.
 * @param sql - new SQL
 */
public PreparedStatement prepareStatement(String sql)

/**
 * Returns ResultSet of the executed query.
 * @return
```

```
 */
public ResultSet getResultSet(boolean reset)

/**
 * Closes the query and makes it available for reuse. If the connection is
used only, it will become available for reuse.
 */
public void close()


/** Returns query as text */
public String getSql()

/**
 * The connection of the query.
 * @return DBConnection
 */
public DBConnection getDBConnection()
```

4.14.3 DBConnection

It's a class that stands for connecting to the database. Each connection can have several queries. By using connections you can handle committing transactions.

```
/**
 * Returns the current connection alias.
 */
public String getDbAlias()

/**
 * Sets autocommit sign.
 * @param autoCommit
 */
public void setAutoCommit(boolean autoCommit)

/**
 * Returns the current state of the autocommit.
 */
public boolean getAutoCommit()

/**
 * Performs commit. Completes the transaction and saves all the changes
that were made in the database.
 */
public void commit()

/**
 * Performs rollback. Completes the transaction and undoes all the changes
that were made in the database.
 */
public void rollback()
```

```
/** Seeking the query among the statements already prepared in the
connection. If the connection isn't found then
 * a new prepared statement is created. If the total amount exceeds the
total allowed prepared statements to the database, the oldest prepared
statement will be closed.
 * If the mode of operation through the connection is set, the connection
number of prepared statement is not limited.
 * @param sql - query
 */
public DBStatement getDBStatement(String sql)

/**
 * Returns immediately the prepared query
 * @param sql - query
 * @return Prepared statement
 */
public PreparedStatement getPreparedStatement(String sql)

/**
 * Closes all the prepared requests and this connection become ready for
reuse. The connection to the database is stored.
 */
public void close()

/**
 * Closes the connection with the database
 */
public void clear()
```

4.14.4 DBConnectionList

The connection list. Stands for concurrent controlling of transactions in several databases.

```
/**
 * Returns prepared statement
 * @param alias - database alias for which the request is needed
 * @param sql - query
 * @return PreparedStatement
 */
public PreparedStatement getPreparedStatement (String alias, String sql)

/**
 * Performs commit for all connections from the list
 */
public void commit()

/**
 * Performs commit for all connections from the list
```

```
     */
    public void rollback()

    /**
     * Closes all connections from the list and makes them available for reuse
     * @throws Exception
     */
    @Override
    public void close()5 Desktop Application Framework
```

This framework is aimed at rapid and intuitive creation of client-server applications using the platform elements of J2EE.

5 Desktop Application Framework

5.1 Used technologies

5.1.1 The runtime

Server and client parts of the framework require Java runtime environment 7 or later. It is desirable that JRE version on the client machine matches to the server JRE.

5.1.2 Application server

System server is deployed under the application server that supports JavaEE 7 Full Profile. We use an application server Glassfish 4.0 for development and debugging.

5.1.3 Third-party libraries

5.1.3.1 GUI libraries

As the main library for building user interface we use Swing that is contained in JRE. Additionally, we use the SyntaxPane library for syntax highlighting in SQL editor and JasperReports to implement the system built-in reports.

5.1.3.2 Server connection libraries

Jersey (REST API Reference Implementation) is used for communicating with the server.

5.1.3.3 JDBC

We use JDBC driver of the corresponding database to work with the external database. The driver must be placed to the extensions directory of the application server. This directory is domains\[domain_name]\lib\ext. Database configuring will be shown in detail in 5.2.4

5.1.3.4 Others

To bind the data on the forms is recommended to use the Better Beans Binding library. It is used in TableSource for binding arbitrary elements of the user interface.

5.1.4 The structure of own libraries

5.1.4.1 Client module (MetaShaper-Client)

The module is responsible for client interaction with the server and provides a template for creating custom forms (ShapeFrame). ShapeFrame is the source parameters and values by name. The parameter list is one for each form. Values must be obtained from ModelSourceI located on the form. By default, the first ModelSourceI is used as the source values. If you want another behavior, you need to specify explicitly ModelSourceI by overriding getValue (String valueName) method.

5.1.4.1.1 Module of the components

The module contains components for building client user interface and it's responsible for the registration of these components as JavaBeans. The main components are:

- DoubleCalendar – calendar-form to select period.
- SingleCalendar – calendar-form to select date.
- JEDFDateChooser – service class for period and date selection.
- JEDFEditor component – field with a label and an arbitrary number of buttons. You can add the input mask and output format to this field. The value of the field is available for binding by value name.
- JEDFDateTimePicker component – date button to select a date from the calendar. The date is available for binding by date name.
- JEDFRefEdit component – value from the reference. It has a label, a code field,

a field value and two buttons – default values reset and selection from the reference. It is also possible to create a mark-button instead of static label.

- JEDFTable – table header filtering with the bottom of the table to show aggregate values, with the possibility of fixing the left and right columns. Works in conjunction with TableSource.
- JButtonFactory – keys factory with the possibility to generate simultaneously pressing JToolBar button and menu items in JMenu.

5.1.4.2 Server Module (MetaShaper-serverobjects)

Module is responsible for the supply the information to the client in response to its request. Also, this module is responsible for matter registration and system command handlers registration from the client to the server. The main components are:

- AppSrv – container of server reflection forms for a single connection (user).
- ShapeModelSrv – server reflection of a client form (ShapeFrame). Contains a list of data providers (DataSrvI) – server reflections of the model source (ModelSourceI). As ShapeFrame client, this object is a parameter source and its values by name.
- ShapeTableSrv – server reflection of TableSource. Contains data for a one table and methods for getting the data.
- ShapeTableProcessor – handler / supplier data for ShapeTableSrv. Can be overriden in ShapeServer.
- ShapeTableListLine – class of a one row in ShapeTableSrv. It can be inherited to determine the setters and getters fields in the row. Computable field are implemented through the mechanism of getter and setter methods.
- ShapeServer – class of user data processing. By using it, you can override the behavior ShapeTableSrv through overriding ShapeTableProcessor. Here are given the rights of the form and custom commands.
- RequestHandler – handler system commands from the client to the server.
- JEDFRequestI – handler interface of one command from client to the server.

5.1.4.3 Support module (MetaShaper-interfaces-lib)

Module contains classes that provide data transfer between the client and the server and vice versa. Also it contains the helper classes such as:

- Classes that work with locks;
- Code processors for matters registration;

- Classloader;

5.1.4.4 Communication and system maintenance module (MetaShaper-REST)

The module contains a handler classes of incoming REST-requests that are transferred to the processing RequestHandler. Also here wound timer of closing unused connections to the database save the matter changes to the persistent store and closing unused sockets in the cluster mode.

5.1.4.5 Client objects module (Classes project)

The module is a collection of client objects (forms, forms server, table line handlers, matters). Module in the assembled state is not used. Used directly compiled class-files and resources, which are deployed to the server.

5.1.4.6 System management module (JEDFDeployModule)

The module is a plug-in for NetBeans, which after installation to the IDE registers a new project type named MetaShaper. Through this project, you can manage system building and deployment. All system management tasks are implemented with ant script located in nbproject\MetaShaper-build.xml. Data for the script are set in the project properties where you can register various domains of the application server, with different settings for the deployment.

5.2 Principles of working the system

5.2.1 Mirroring data between the client and the server

Data that the server sends to the client are left on the server until client window isn't closed in proper way.

5.2.1.1 Interfaces ModelSourceI and DataSrvI

ModelSourceI is a data provider for client components. It's created on the frame and inseparably connected with DataSrvI - data provider/holder on the server. ModelSourceI defines data sets for sending to the server and events wich trigger the sending. For example, it's defined for the pair of TableSource and ShapeTableSrv that when a user selects a new row in thetable, the information about new selection is sent. ShapeTableSrv detects independently came

information and refreshes it own structures at the same time. A form can have several data providers.

5.2.1.2 Implementation in the example TableSource and ShapeTableSrv

TableSource is a data provider of table data that needs to be used in conjunction with JEDFTable directly or with JTable via JTableLinker class. TableSource defines SQL query and table fields set. ShapeTableSrv is a data provider/holder of the server for TableSource. ShapeTableSrv holds a full data set got from the database or formed differently; but the client gets only part of that data which are visible on the screen at the moment. Therefore, when the visibility zone on the client is changed, the server sends a new portion of the information. It's possible to form a data set not by using SQL query and by overriding the resetData method of the ShapeTableProcessor object in the server module. ShapeTableSrv has a feature to sort and filter data sets as well.

5.2.2 Benefits of data processing (business logic) on the server side.

In order to minimize client work and performing business logic in one place you should put application logic in the application server.

5.2.2.1 Server custom forms (ShapeServer)

User server form allows to set specific right to the client form and override data in the ShapeTableSrv class by overriding ShapeTableProcessor.

5.2.2.2 Table rows handlers (ShapeTableListLine)

Data in ShapeTableSrv are stored as a list of ShapeTableListLine instances. By default, the data are set and extracted as a regular objects but it's possible to define field getters and setters, thus determining the logic of setting and extraction. A specific string handler class is set in the handlerClass property of the TableSource component.

5.2.3 Lack of rigidly structured application

5.2.3.1 Starting forms

Start frame is a regular application frame while the registration it was set that it's start (via isStartFrame property of @ShapeFrameDcl annotation). Available to the user starting forms are showm in the window that is next after user authentication. Start frame must have the server module that implements RootShapeServerI interface. This interface has two methods checkRootRight and checkRootRightX which can be defined in a standard way by overriding isRight and isRightX methods or if you need specific handling standard rights such as INSERT, UPDATE, DELETE with using specific parameters.

5.2.3.2 Application as a set of forms opened by the user (AppSrv)

Application exists in the system as a user has authorized in the system and started to work with allowed to him frames starting with the start frame. Thus it's formed a list of frames available to the user.

5.2.4 Working with relational databases

5.2.4.1 Configuration

Configuring the data store is carried in JEDFConfig.xml file. In the case of using glassfish this file is placed in the config folder of your domain.

5.2.4.2 Aliases

Aliases of data store are defined in JEDFConfig.xml through them access is carried from the application. It's required one alias at least - JEDFSystem; through it access is carried to the main data store.

5.2.5 Possibility of using third party technologies such as on the server-side and on the client-side.

Using third-party technology is possible due to the mechanism of embedding additional libraries to the distribution kit of the client and the server. See 3.1.

5.3 Built-in report system

JEDF has an integrated report system that allows to manage all reports in the system. It consists of two parts:

- Reports store - keeps all the reports and provides a versioning for each report. It means that each report can have versions but you need to select the

main version that in use at the moment. Also any version of any report can be edited by iReport report editor (needs to be installed) by clicking on the edit button. After saving the opened report it will save the report automatically in the database.

- Rights to reports – there is a possibility of granting right to any report. Each report can have any number of custom rights. Each right can be enabled of disabled in the departments management window on the reports tab. It is the same approach that used for forms in that window (the forms tab).

A report itself must have a class that have business logic. The full qualified class name you need to set in the reports store window.

6 Cluster

The delegation of the functioning within a complex system is a vital point. When the whole load rests with the only one server, even it's powerful, it can have grave implications when the number of online users increases by leaps and bounds. The typical solution can be enhancing of computing power by adding additional memory or replacing the processor. But this rash action can be avoided by using the distribution of system's logical parts.

Many ERP/CRM systems are based on the following scheme: Application server, where allocated the application itself and RDBMS, where the data are stored. In this case the delegation of the functioning is allowed but it reduces to additional efforts: system administrator has to use a particular software to reduce the current system into cluster model, moreover the increment isn't guaranteed.

JDEF team suggests another idea of increasing and delegating different of the functioning: The whole ERP/CRM system must be designed as the ensemble of independent logical parts (applications) so that each allocates into it own application server. Thus we have a couple of main advantages:

- Hardware can be selected properly depending on load onto certain application.
- Whole matter's cache is divided into several parts for each cluster node, thus matter's codes don't overlap and this fact positively influences perfomance on the whole.

Distinctive features:

- There is only one instance of permanent storage;
- Each cluster node has a matter cache but it constantly maintains in a synchronized state between every nodes so it gives us actual data within the whole system.

- There is only one server can be marked as a main and others as additional.

To make clusters nodes there are few steps:

You need to write In JEDFConfig.xml into JEDF tag the following tags:

```
<MainServer>
    <IP>IP_of_the_main_server</IP>
    <PORT>Port_number_of_the_main_server</PORT>
</MainServer>
<AddServer>
    <IP>IP_of_the_additional_server_number_1</IP>
    <PORT>Port_number_of_the_additional_server_number_1</PORT>
</AddServer>
<AddServer>
    <IP>IP_of_the_additional_server_number_2</IP>
    <PORT>Port_number_of_the_additional_server_number_2</PORT>
</AddServer>
...
<AddServer>
    <IP>IP_of_the_additional_server_number_N</IP>
    <PORT>Port_number_of_the_additional_server_number_N</PORT>
</AddServer>
```

Database alias must be written with using IP-address. E.g.:

```
<jdbc_url>jdbc:firebirdsql://192.168.1.100:3050/C:\\JEDFDatabase\\
JEDFSystem.fdb</jdbc_url>
```

If there are two transactions that try to commit the same matters at the same time, there will be the following actions: the first transaction will commit it own changes when the second one will be waiting for a while (waiting time is set in JEDFConfig.xml). Further the second transaction will try to commit again and if the first transaction has been committed, it will commit it own changes as well. There is a few attempts of each transaction that also is set in JEDFConfig.xml.

7 Administration

7.1 Intro

JEDF provides essential administration tools to make controlling more efficient. There are such things like:

- Controlling users, their right and their positions.
- Creating new departments and editing existent ones.
- Registering and removing any windows or reports.

- Several powerful tools to controlling all the matter aspects.
- A tool that makes possible creating and loading absolutely new classes in run-time.

7.2 Users

To open User Management open Administration window and select User Management button and a new window will open. There are 3 main areas:

- A top area contains a list of users that registered in the system. You can add a new user or remove existed one. Each user has name, login and password. All that can be changed at any time.
- A middle area shows what positions a certain user belongs to. User positions can be changed by adding or removing ones.
- And the last, bottom, area displays what forms are allowed to the selected user. But this list depends on a specific position: each position has predefined windows.

7.3 Departments

To open Department Management open Administration window and select Department Management button. There will be 3 main areas:

- Left area represents a department tree. Each node of the tree means a department, each leaf is a position. You can add a department or a position by means of clicking proper button. If you want to add a new department, you have to select required department (some node) and click on Add department button. You'll have to input department name. Every department can have many inner departments. If you want to add a new position in a certain department, you have to select required department and click on Add position button. You'll have to input necessary information for the new position (position name, profession (select from the profession reference), position description and other data). Each department can have many positions. All information of some department or some position can be edited by clicking on Edit button.
- Right top area contains two tabs. The first tab contains a table with list of windows that are allowed for the selected position and the second tab contains a list of allowed reports for this position. You can add or remove a certain window or report in the selected position.
- The last right bottom area is a table that contains rights to selected window or report. Each right is set in a server form or a report individually. Here you can only set it on or off.

7.4 Windows

To open Windows form open Windows form. The opened frame is a list with registerd frames that have JEDF. There is such information like form class, path to the icon, whether it's on a main frame.

7.5 Reference checking

To open the Reference checking tool click on Check matter refs button in the main administration window. This is a tool for searching and correcting corrupted matter references. A corrupted reference is a reference that doesn't have reverse matter. After finishing checking a short report will appear. It has information about:

- The total number of checked matters.
- How many reference fields has been deleted.
- How many reference fields of the count type has been deleted.
- How many references has been deleted.
- How many reference fields has been restored.
- How many references has been restored.
- The total number of checked reference fields.
- The total number of checked reference fields of the count type.
- The total number of checked references.

7.6 Matters dumping

To open the Reference checking tool click on Check matter refs button in the main administration window. This tool allows to dump all in-memory data into the storage. This means that all matters, which located in the memory cache, will save into the storage. After this procedure there will be a message informing us about the total number of matters in the memory cache and how many of them has been saved into the storage.

7.7 Matter administrator tool

To open matter administrator tool you have to click on Matter administrator tool button. There will be a window contains three areas:

- Top area is an input box where can be input code of a certain matter or even sql-query like «select * from [Service_Table_Name]», where Service_Table_Name is a table in the database of a specific registered matter. After that a

matter or matters will be loaded into the second area.

- The second, left, area represents a tree which contains loaded matter or matters. Each node in the tree is a matter; a node can be expanded so that appeared inner nodes are others matters. All appeared matter (inner nodes) is linked with each other through matter reference mechanism. A reference type stands for various icons of each node.
- Right area is a table that contains field names and their values. Each value can be changed.

Also there is a panel that contains a few buttons:

- Save button needs to save changes in a certain matter after editing its value.
- Remove matter button needs to remove selected matter. Removing is possible if the matter has no any outer links to itself.
- Remove bidirectional reference button needs to remove bidirectional link between selected matter and the matter which code you type into opened dialog.
- Editing additional reference data button needs to edit data which located «on» the reference.

7.8 In-memory classes tool

To open In-memory classes tool you have to click on In-memory classes tool button. This tool stands for creating, editing and loading java source classes in run-time. In order to set any outer data usually is used XML; XML is only static data without any logic. But what if we want to use something more functional? Simple but effective solution at the same time is applying the same source classes what we are using day after day. This approach allows us to keep outer data with outer logic. So when you need to create new in-memory class just click on Add button, type in its name and in the right area, which is a source code editor, type whole class source. If your class requires other class or classes, you may specify this - just fill this into according column. As soon as you save your class, it will be allowed in your application through standart classloader.

7.9 Report tool

To open report tool just click on Report tool button. Report tool helps you manage and organize all your reports. A typical report consists of the next parts:

- Instance of Report Matter;
- Report class that has all logic and rigts definitions;

- One or many versions of the report. It means that each report can have more than one actual edition. Each version can be set as main and after calling this report in the application you'll get preset actual version. Moreover, each edition can be modify straight from the tool (if you have iReport though) by clicking Modify report button.

Detailed information about reports and this tool in particular is in FAQ and Guides.

7.10 Deadlock window

To open Deadlock window select Windows menu and click on Deadlocks. This is a viewer that shows the list of locked.

7.11 Active transactions

To open Active transactions window select Windows menu and click on Active transactions. This is a viewer that demonstrate a list of current transactions in the system. There are three areas:

- Top area that contains a table with transactions. There are such details as transaction date, user started transaction.
- Left bottom area is a table showing what matters engaged in the transaction.
- Right bottom area is a table with fields from the matter from the previous table. There are only fields engaged in transaction.

Each transaction can be removed (rollback) by clicking on Remove button.

7.12 Active forms

To open Active forms window select Windows menu and click on Active forms. This is a viewer that just shows what forms are active at the moment.

8 Migration from Delphi

There is no such a thing as «choose Delphi project, then click a button and you'll get full-functional JEDF project» (at least not now), but rapid migration from Delphi is possible due to very similar way to access databases in Delphi and JEDF.

So, in Delphi we use TForm, to which we put TQuery with a query to db, link TQuery with TDataSource and then link TDataSource to Data-aware components

such as TDBGrid, TDBEdit, etc.

In JEDF we use ShapeFrame, to which we put TableSource with a query and link TableSource to Data-aware components, such as JEDFTable or link it to any component on the frame through BeansBinding mechanism.

Visual editing of ShapeFrame in JEDF is done via NetBeans's JFrame visual designer, which is quite different from Delphi visual designer, but is really useful and provides quite the same functionality as Delphi Designer.

Also you'll get out-of-the-box JEDF features such as transparent rights management, SQL cache, etc. Furthermore you can lately adapt your business logic for use of MatterBase, which wil give you more production speed with reduce of complexity of source code.