The database management system **JEntigrator**.

White paper. Alexander Imas, 2016, Darmstadt.

1.Definition.

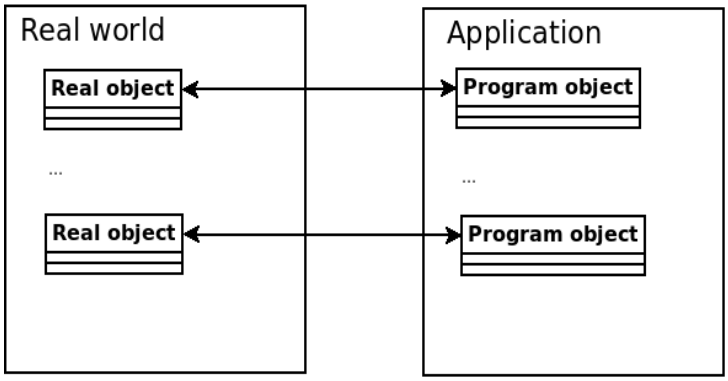
The **JEntigrator** is the single-user local file-based database management system implemented as a **java** program. It is intended to manage semi-structured evolving data objects and to serve as a basis to build applications using such data objects.

2. Evolution data objects within database applications.

Database applications reflect objects from the real world in program objects in the application. The application provides the interface to control data objects and synchronize changes of corresponding objects in both environments.

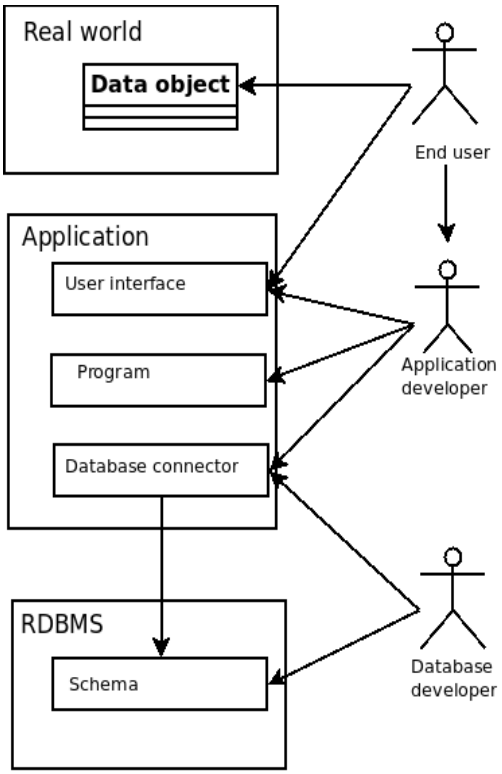
Under the data object we understand an object having state and behaviour. The state is represented as an array of named fields , the behaviour is defined as a set of reactions on impacts. We distinguish two kinds of object changes: content and structure. The content change means changing the state variable values. The number and names of state variables stay intact as well as objects behaviour. The structure change (evolution) means either changing the set of state variables or object behaviour. When the content change is the standard operation under the responsibility of the end user (the application usually provides the convenient interface for it), then the structure changes is much more complicated challenge that demands changes in the application code and the database schema.

The figure below shows the evolution of the data object across the application based on the classical relational database management system (RDBMS) .

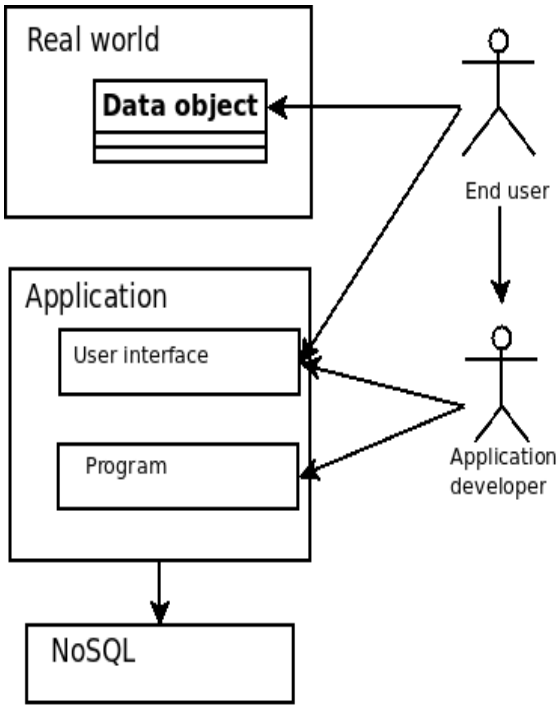


Assume the application user have recognized the necessity to change the structure of the real data object. He requests changes by the application developer. The application developer modifies the application code and requests changes of the schema in the RDBMS by the database developer. The

evolution process involves both program and database components of the application under the responsibility corresponding developers.

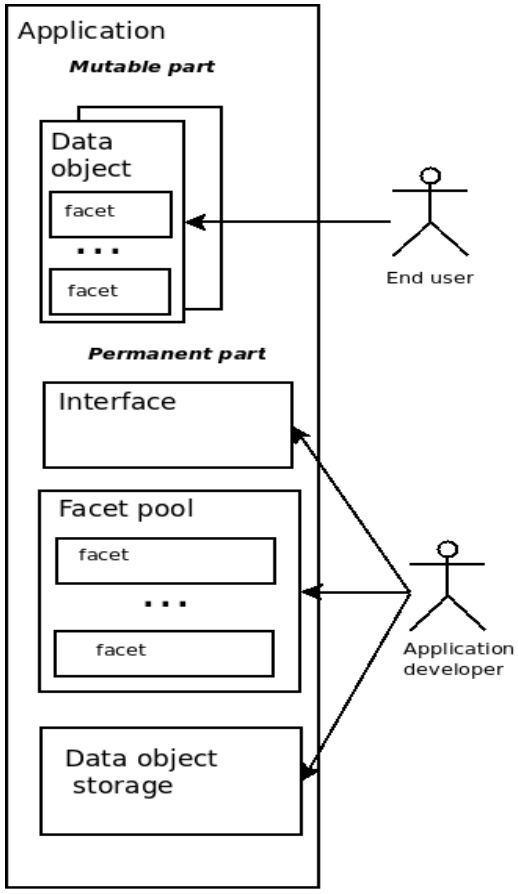


If the application data is not relational then a NoSQL database can be used. The NoSQL database management system has no schema and its interface is independent on the structure of data objects . The application developer does not need assistance of the database developer to complete the evolution. All changes are encapsulated in the code of the application. In that case the evolution process shown on the figure below demands lesser efforts and can be maked quicker and cheaper. The participation of the application developer in the evolution process seems to be inevitable.



3. The compositional data model.

The **compositional data model** is based on the assumption that the real data object can be represented as a composition of more elementary components called **facet**s. A **facet** is defined as a portion of data and assotiated with it behaviour. For example the **phone** facet includes the phone number (data) and the call (behaviour). Another example is the **image** facet uncluding the image file (data) and viewing of the image(behaviour). A **facet** can be implemented as a program object where the data component is represented as a set of named fields and the behaviour is implemented as a facet's handler – the code processing these fields. The facet's data contains a descriptor of facet's handler. The most fundamental fact is that the set of facets is essentially smaller and stabler than the set of data objects. The second important fact is that the data object structure can be built or modified in the same fashion as the content – by assigning data to the object without changing of the application code. In other words the data object design can be delegated to the end user. The data object can be considered as a facet container and the data object design means to add/remove facets to it . The application developer must provide the permanent part of the application: a pool of facets and the interface to manage universal data objects – facet data containers. The end user takes over the design and maintenance of data objects, again, without any coding. This work belongs to the mutable part of the application as shown on the figure below.

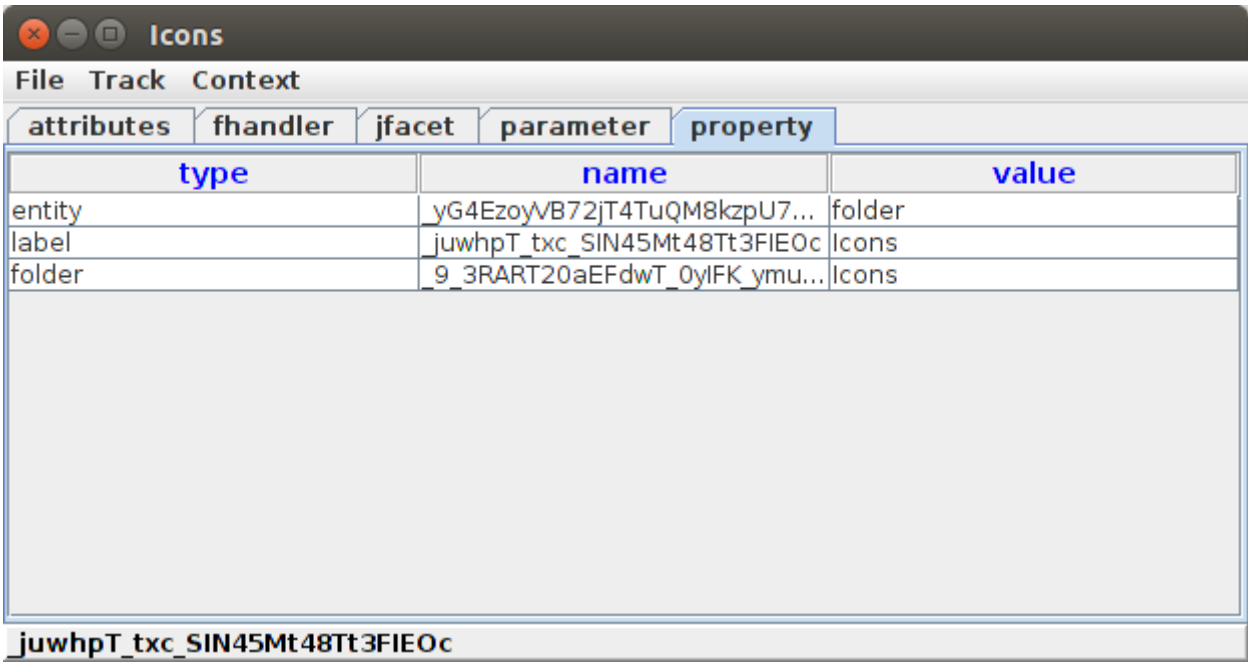


But what if there is no facets in the pool to fulfil new requirements ? The composition data model has the ability to absorb data objects of the special **extension** type. The extension contains the library file and descriptors of the facets belonging to this extension. It is enough to import the

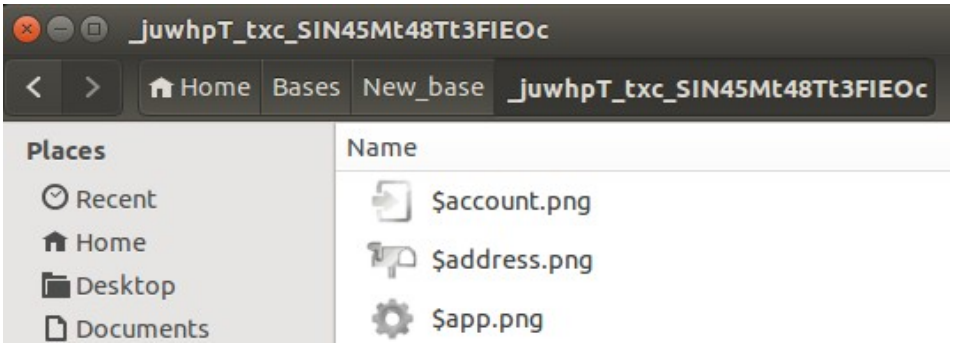
extension data object in the data object storage in order to add new facets in the facet pool. The application loads the extension library and makes new facets available to the end user. The application code stay intact. The compositional model supports container-component relations between entities enabling users to build complex hierarchical structures.

4.The basic data object **entity**.

The **entity** is the basic data object structure in the compositional data model. An **entity** is defined as a facet container. It collects facet's data and handlers that can be changed during object lifecycle. The picture below shows the entity representing the **Icons** folder displayed in the **JEntityEditor**.



From the logical point of view each entity contains any number of elements having unique names within entity. Each element can contain any number of cores **.** Each core contains three string fields: **type, name** and **value.** The **name** of the core must be unique within element. Each entity has unique fields key and label within the database. The key is immutable during the entity lifecycle. If an entity has a property **folder** assigned then the directory named as entity key will be created to keep files associated with the entity.



Physically an entity is an **xml** file saved in the database entity directory.

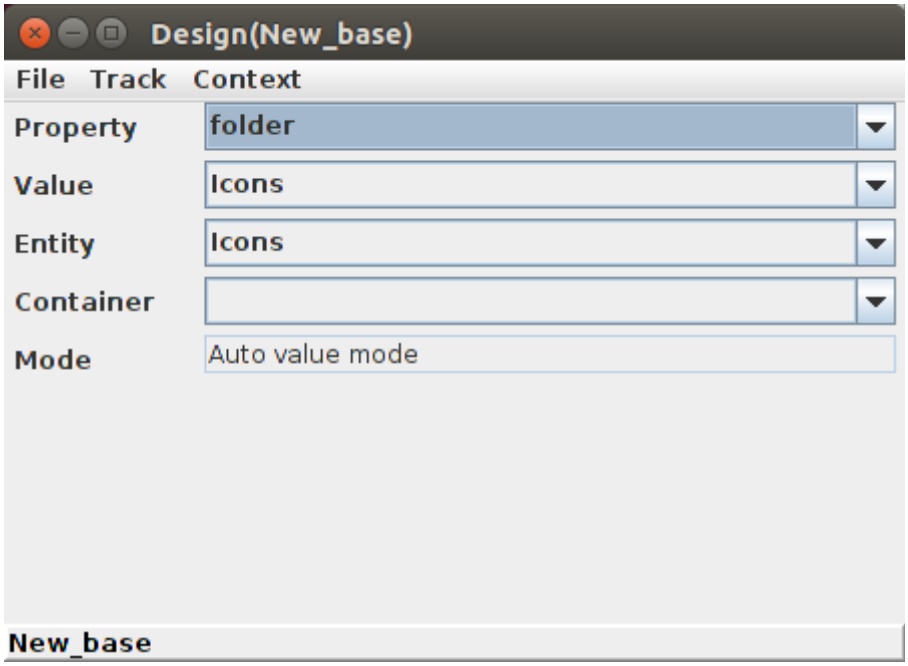
_juwhpT_txc_SIN45Mt48Tt3FIEOc

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<sack path="null" key="_juwhpT_txc_SIN45Mt48Tt3FIEOc">
<attributes>
<attribute type="String" name="alias" value="Icons" />
<attribute type="null" name="icon" value="gallery.png" />
<attribute type="null"
name="key"
value="_juwhpT_txc_SIN45Mt48Tt3FIEOc" />
<attribute type="key"
name="residence.base"
value="_xXsnV5R_SGxkuH_SpLJDeXCglybws" />
<attribute type="null"
name="timestamp"
value="1457293244279" />
</attributes>
<elements>
<element title="fhandler" >
<item type="null"
name="gdt.data.entity.facet.FolderHandler"
value="null" />
</element>
<element title="handler" >
<item type="null"
name="gdt.android.entity.service.digest.ServiceDigestHandler"
value="null" />
<item type="null"
name="gdt.android.entity.service.folder.ServiceFolderHandler"
value="null" />
<item type="null"
name="gdt.android.entity.service.primary.ServiceEntityPrimaryHandler"
value="null" />
</element>
<element title="jfacet" >
<item type="gdt.jgui.entity.folder.JFolderFacetAddItem"
name="gdt.data.entity.facet.FolderHandler"
value="gdt.jgui.entity.folder.JFolderFacetOpenItem" />
</element>
<element title="parameter" >
<item type="String" name="camera.resolution" value="phone" />
<item type="String" name="reuse" value="true" />
</element>
<element title="property" >
<item type="folder" name="_9_3RART20aEFdwT_0yIFK_ymuLg" value="Icons" />
<item type="label" name="_juwhpT_txc_SIN45Mt48Tt3FIEOc" value="Icons" />
<item type="resource" name="_xhehXxQ22Hn4KutYvB61MrUMWgk" value="Icons" />
<item type="entity" name="_yG4EzoyVB72jT4TuQM8kzpU7qYE" value="folder" />
</element>
</elements>
</sack>
```

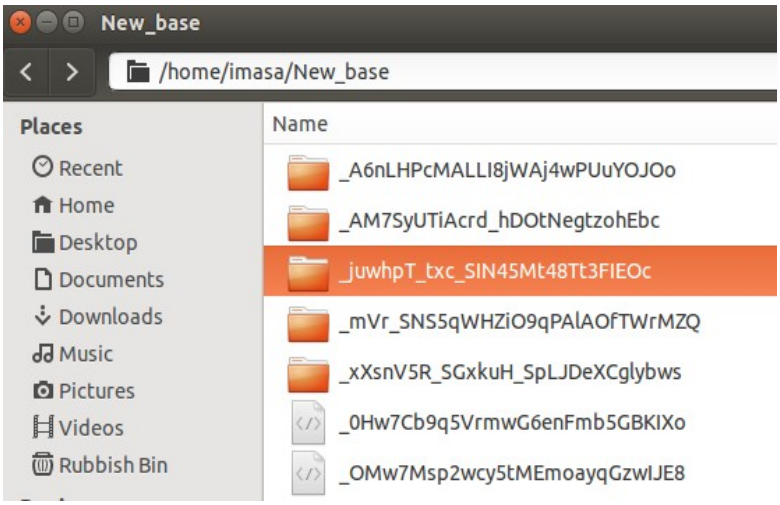
5. The property index.

An index is an essential part of the database management system. It is defined as an additional data structure that improves the speed of data retrieval operations on a database. The **Entigrator** has the built-in global index for the fields in the mandatory entity element **property.** The **type** field of the property core contains the name of the property, the **value** field contains the property value, the **name** field contains the unique index entry key. The picture shows the **JDesignPanel –** the multipurpose database management console in the **JEntigrator.** The **Property** selector contains all property names in the index. The **Value** selector contains all values for the selected property name. The **Entity** selector contains a list of entities having the selected value assigned.



6. The database file system.

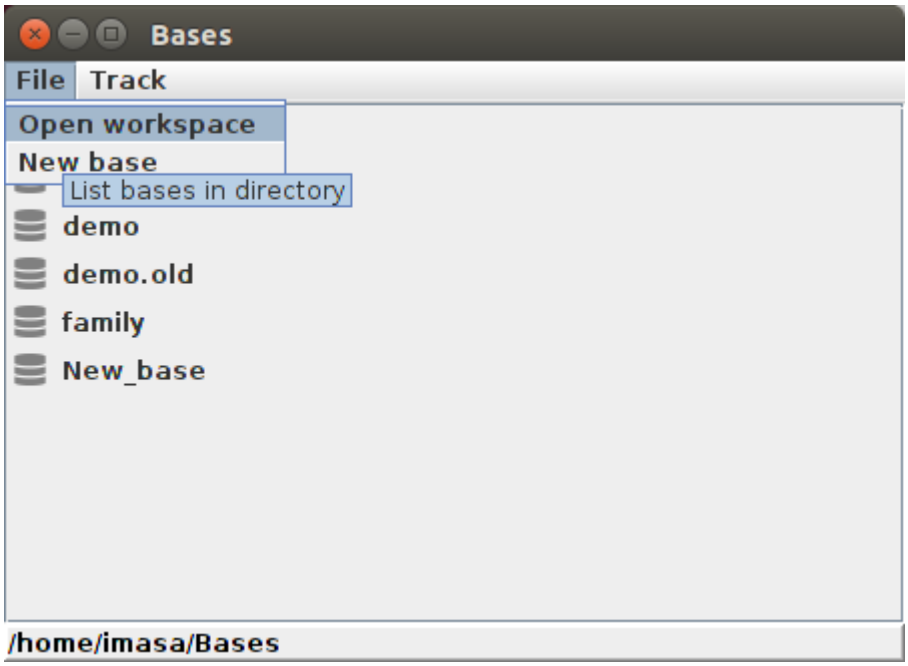
Physically the database is a file structure located somewhere on the disk. The number of directories in the root directory can change, because each entity having the **folder** facet assigned becomes its own home directory. On the picture below the **Icons** entity having the key **_juwhpT_txc_SIN45Mt48Tt3FIEOc** owns the folder of the same name.



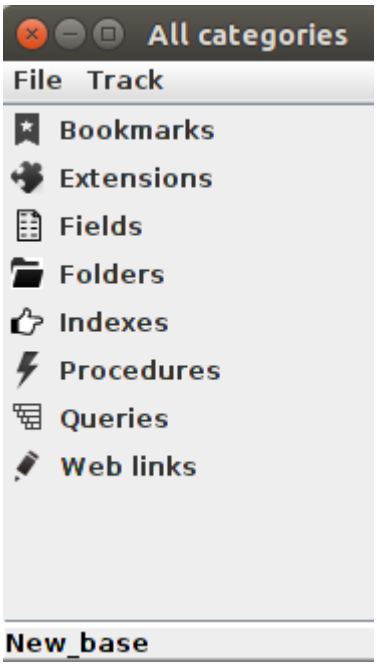
7.The maintenance of the database.

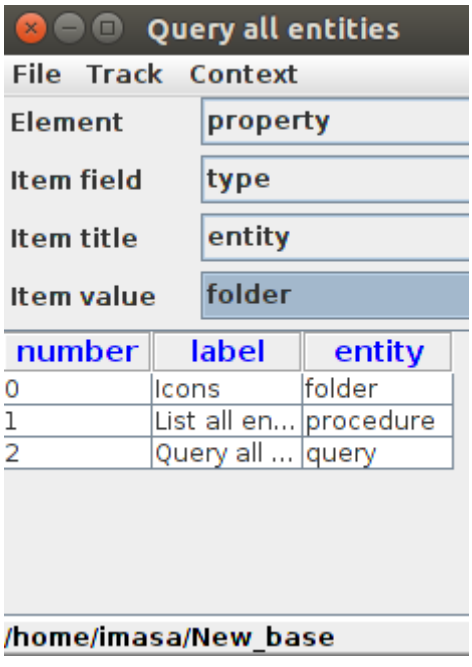
We distiguish three groups of maintenance activities in the **JEntigrator** : data management, extension development and application development.

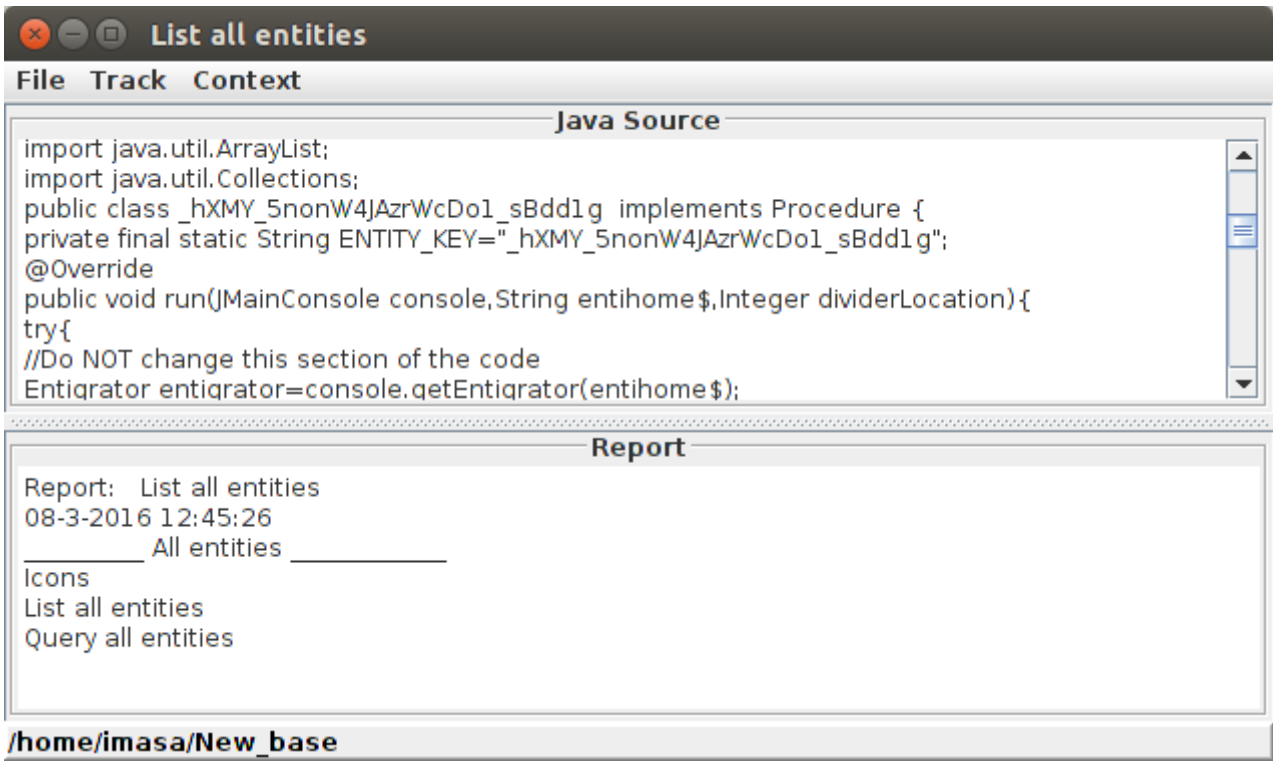
The data management belongs to the competence of the end-user. The end-user takes over the installation of necessary extensions, making data design, creating and deleting data objects, modifying data objects content and structures during their lifecycle and so on. Most of this work can be done without any coding through the graphical user interface of the **JEntigrator**.



A user opens the existing database or create a new one and get access to data objects within the database. For example, the **JBaseNavigator** panel displays the list of all types of data objects in the database. A user can add custom procedures and queries using source code of providing templates, which requires some programing in java.







The new created database contains very few of general-purpose facets. To be really useful the database should include one or more **extensions –** the data objects containing libraries which provide a set of facets reflecting the business logic of user objects. The author provides source code of the **community** extension that can be used as a template. One can say that the extension adapt the database to the user's business area in the same sense as the schema does it in the RDMS. That's true but there are some essential differences. The extension does not prescribe the structure of data objects but gives possibilities to add new facets to them. Many independent extensions can be included in the database. Adding an extension does not impact existing data objects and does not violate the consistency of the database. The end user can install extensions from the **JEntigrator** interface without any programing.

The application **JEntigrator** is published as open-source code available under a GPL license. The application development means improving this code.

8. Conclusion.

The **JEntigrator** is an extensible open-source java application based on the original single-user database management system. The application exploits semi-structured evolving data objects which are built as a composition of facets. A facet contains two components: data fields and piece of code (handler) processing these fields. The end-user implements the whole data objects management including design, structure evolution and content changing through the application GUI without programing. Data objects having the **extension** type contain additional libraries extending the set of avalilable facets. They can be imported into the database in order to adapt the application to the user's business area.