The JEntigrator – the local data management system.

Alexander Imas. Darmstadt 2016.

The **JEntigrator** is a single-user, local, file-based, NoSQL data management application implemented as a java program. It is published as an open source project under the GNU GPL licence on the **github** platform. Sources and documents can be downloaded here:

https://github.com/imas-alex/Jentigrator/tree/master.

Artifacts can be downloaded from the **maven** platform here:

https://oss.sonatype.org/content/groups/public/com/github/imas-alex/JEntigrator/0.0.2-SNAPSHOT/JEntigrator-0.0.2-20160428.124125-1-jar-with-dependencies.jar

https://oss.sonatype.org/content/groups/public/com/github/imas-alex/Community/0.0.2-SNAPSHOT/Community-0.0.2-20160428.124445-1.jar

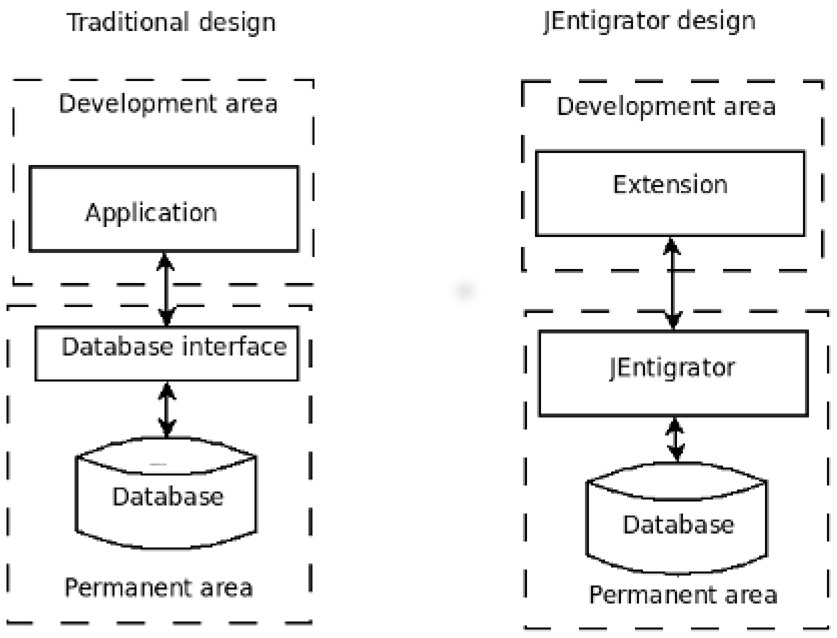
The concept.

Existing database management systems have problems to reflect agile data in the database, as already discussed in the article Evolution of Data Objects Within Database Applications. Such data objects can change content , structure and relations during their life cycle. When content of data can be changed through the standard user interface by the end user, then the structure changes (the evolution) enforce involvement of database designers (in case of relational database management system) and application developers. In both cases the application code must be modified. This makes the evolution process expensive and time consuming.

The compositional data model offers the way to delegate the design and the evolution of data objects to end-users without any programing. Data objects are represented as compositions of more elementary components – facets. A facet comprises two parts: a portion of data within the data object and a portion of code within the application. The evolution means to add/remove the facet data in/from the data object and the portion of code in/from the application. Adding the facet data to the data object is a standard content changing operation. Adding the facet code to the application is accomplished through the extensions mechanism. An extension is a data object containing a library with the facets code. It's enough to import the extension object in the database to make the facet available for the application. The import a data object in the database is also a standard operation under the responsibility of an end-user. Of course , somebody must create the library , so there are code developers behind the scene. The benefit of the extensions mechanism is that the application code stay intact. The development of a facets library is much easier task than upgrading the whole application.

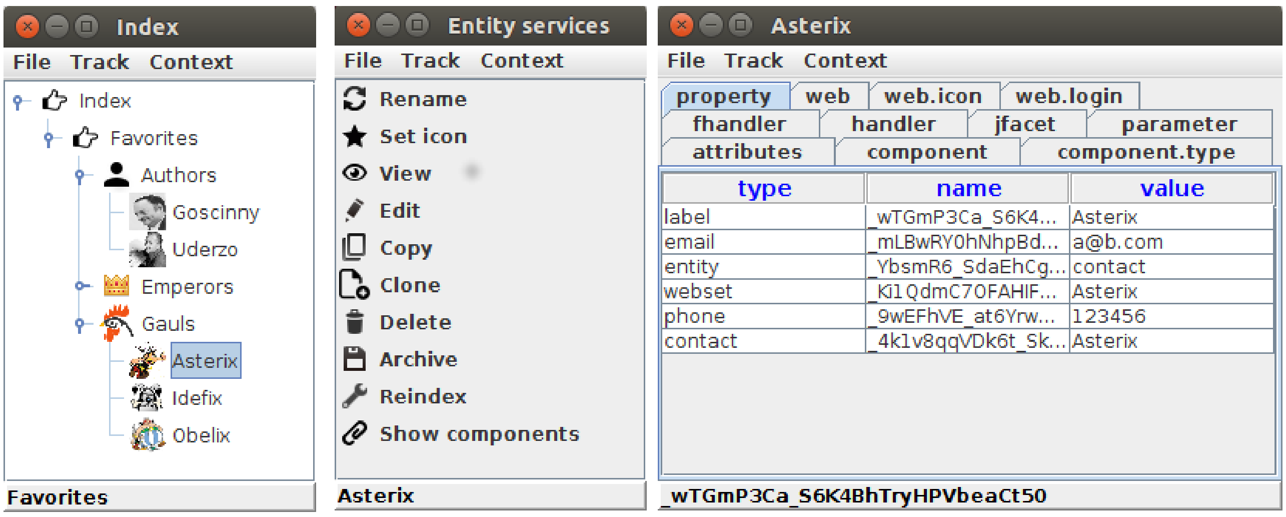
Actually, the idea of the compositional data model is very similar to the Feature Oriented Software Development (FOSD) paradigm, where programs are considered as a composition of the basic program and incrementally added features.

The **JEntigrator** is the application based on the compositional data model. It can be considered as an universal data manager providing general-purpose database and facets services. The difference between design of the traditional database application and the **JEntigrator** is shown on the picture below. We can see that the **JEntigrator** is located in the same abstraction level in the data flow as the database interface. Theoretically the common database interface provides full control on data objects through the SQL or command line (assume the user has admin rights) . Practically the ordinary user has no necessary knowledge and time to do it apart from danger to damage data. The **JEntigrator** provides full control on data through the user friendly and safe GUI.



The user interface.

The design of user interface in the **JEntigrator** is not straigforward. The traditional application possess the whole information about data objects during all the time. Developers can use the known structure of a data object to program corresponding control elements at the design time. The **JEntigrator** get the information about data objects only at the runtime by loading the extension libraries. It is impossible to provide the **JEntigrator** with embedded control elements for data objects which structure is unknown. In order to solve the problem the **JEntigrator** follows the same idea as the compositional data model – it delegates the control elemens to facets. The basic element of the **JEntigrator**'s GUI is the context panel - the place holder for facets controls. The context control will be loaded into the context panel by accessing the corresponding facet for the selected data object. The picture below shows context examples.



type	name	value
label	_wTGmP3Ca_S6K4...	Asterix
email	_mLBwRY0hNhpBd...	a@b.com
entity	_YbsmR6_SdaEhCg...	contact
webset	_Ki1QdmC70FAHIF...	Asterix
phone	_9wEFhVE_at6Yrw...	123456
contact	_4k1v8qqVDk6t_Sk...	Asterix

_wTGmP3Ca_S6K4BhTryHPVbeaCt50

Key features of the application.

- **Flexibility -** full control on content and structure of data objects. Data design and evolution are delegated to end users.
- **Extensibility –** data objects are represented as free compositions of facets. The facets pool can be extended by usage of addional facets libraries.
- **Regularity –** the **JEntigrator** uses the unified internal data structure called 'entity' for all data objects. An entity is a container for any number of named fields, facets and files. They can buid container–component relations with each other. All data objects, libraries, procedures and queries are wrapped into entities and located in the database. The only external file is the **JEntigrator** itself.
- **Universality –** embedded facets provide general-purpose tools to build unspecific data objects and structures . The **JEntigrator** can (and must) be adapted to the user's business area by the corresponding extension .
- **Portability –** entities can be transferred from one database to another by the copy-paste procedure. The **JEntigrator** resolves all dependencies and transfer all accompanying entities together with selected entities.
- **Efficiency –** some named fields within an entity can be qualified as properties. They are

included into the global database index. Entities can be quickly found wth a criteria search.

- **Accesibility -** comprehensive graphical user interface. Entities design and filling can be fulfiled directly from GUI. All additional programing only in java. The open source code provides developers with the necessary API. The open source sample extension **Community** can be used as a reference example.

- **Openness –** provides direct access to the operating system functionality from a context control. For example, the entity's home folder can be opened in the system file manager.