



Nexirus GmbH
Sonnhaldenweg 66
CH-4450 SISSACH
Switzerland
Tel: +41 61 973 01 23
marcel.baumann@nexirus.ch

Architectural Overview



Application Framework for Java™

This document briefly describes the architecture of ***jnex***

Author: Marcel Baumann
Version: 5.2
Date: 6/18/2009 6:29:00 PM
File: C:\marcel\projects\src\jnex\docs\ArchitecturalOverview.doc

1	<i>Introduction</i>	3
1.1	Who shall read this document?	3
2	<i>What is the benefit of jnex?</i>	3
2.1	<i>jnex</i> und Swing™	4
2.2	<i>jnex</i> and HTML	4
2.3	<i>jnex</i> und Software Design	4
2.4	<i>jnex</i> is not epidemic	5
2.5	<i>jnex</i> is not a code generator	5
3	<i>The Model View Controller Pattern</i>	6
3.1	Benefit	6
3.2	<i>jnex</i> and MVC	7
3.3	Nested DataModels	8
3.3.1	Using Swing LayoutManagers and JComponent Properties	9
3.4	Mapping of jnex data models to appropriate editors/viewers	9
3.4.1	Events	10
4	<i>Swing Applications</i>	11
4.1	Pulldown Menu and Tool Bar	11
4.2	Internationalization	11
4.3	Dialog Windows	12
4.4	<i>jnex</i> Servlets (Web-Applications)	13
5	<i>Persistence</i>	16

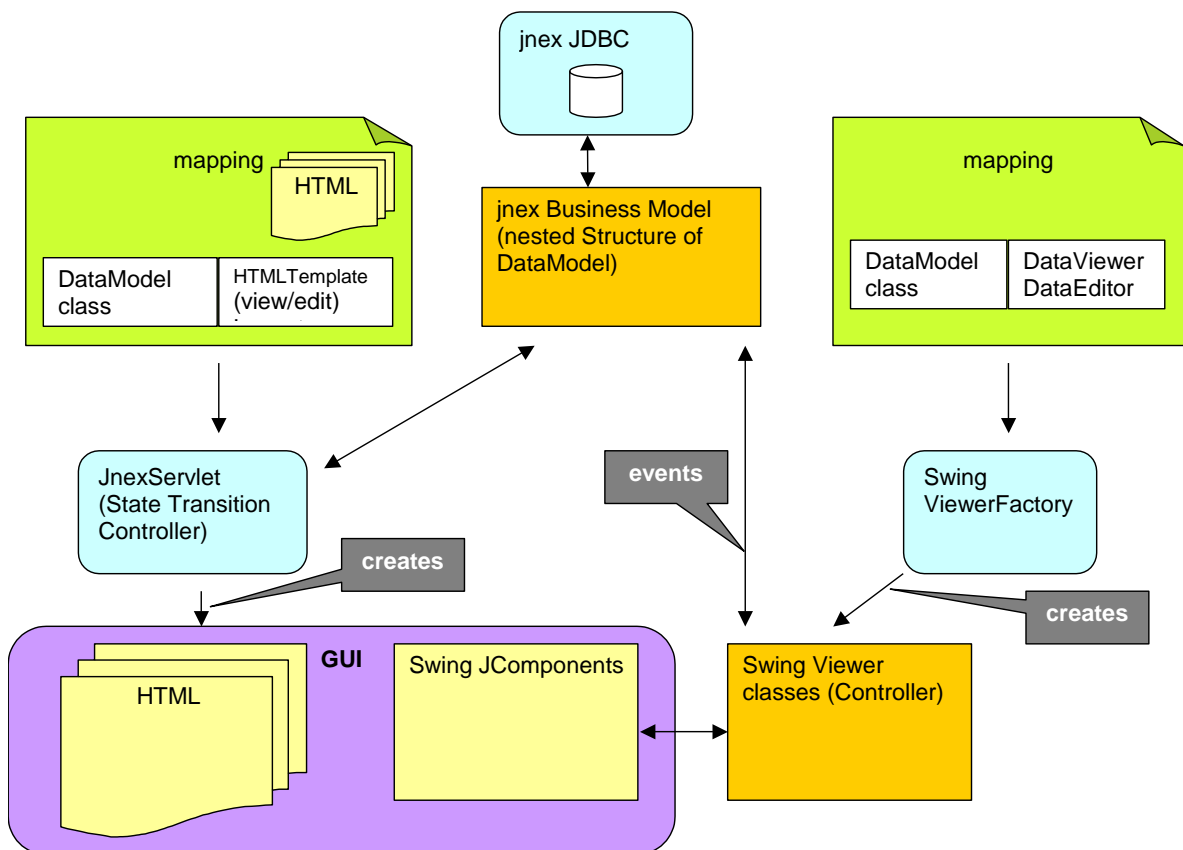
1 Introduction

This document describes the architecture of *jnex* and its major classes as well as their interaction. *jnex* can be used as a base framework for Java application development. The consequent use of *jnex* speeds up the development time and enhances the code quality and structure.

The reason is the consequent separation of the business model from the user interface. The same data models can even be used as a basis for three different rendering technologies

- ❑ Swing rich client applications
- ❑ thin HTML web-applications
- ❑ Canoo ULC applications

The data models of *jnex* can be arbitrary nested and it is very easy to store (load) them to relational databases with JDBC technology.

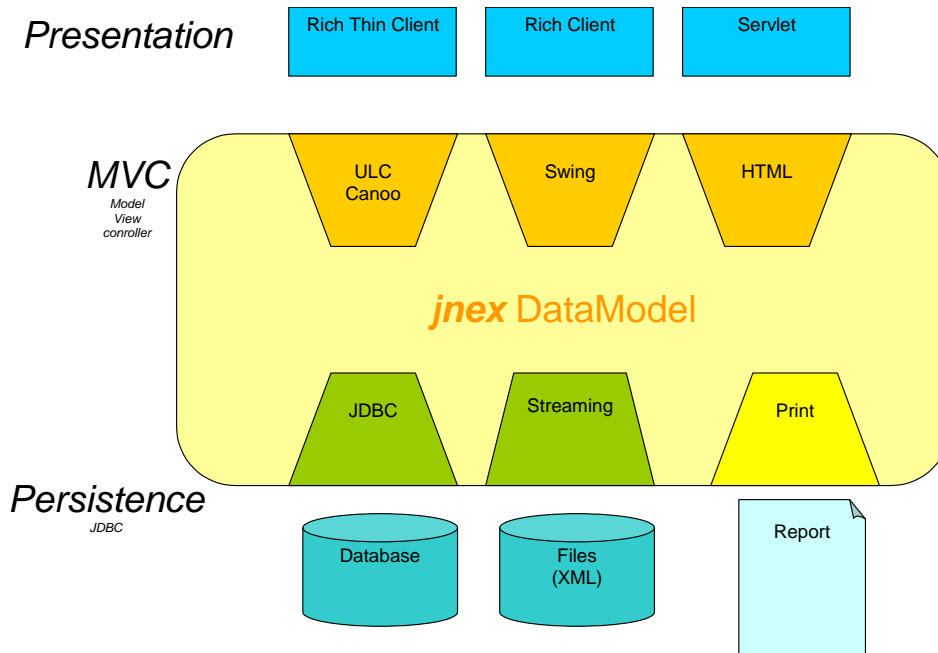


1.1 Who shall read this document?

The reader of this document is getting an overview of the *jnex* technology and how it can help to structure complex software projects.

2 What is the benefit of *jnex*?

When you decide to use *jnex*, then you can build one set of business models, which can be used on the database layer for simple JDBC access, on the middleware as simple access objects and on the presentation layer as GUI model. Other technologies map data from one similar structure to the next which leads to a situation where a simple extension of a business model with one attribute triggers code changes in all architectural layers.



2.1 *jnex* und Swing™

Swing™ is part of the Java Developer Kit from Sun Microsystems. It is a class library to build windows, text input fields, buttons and more. Using Swing is not trivial. Apart from that, in big projects similar GUI components are reused over and over to display similar content.

jnex eases the use of Swing and reduces the amount of redundancy.

Every *jnex* data model can be automatically translated into the appropriate Swing component, which serves as an editor or viewer of the specific information.

The *jnex* data model assures the input value consistency while editing. Micro validation is used for basic data models (StringModel, DateModel, IntModel, CurrencyModel, ...). Macro validation is used to validate data model containers (StructModel, ArrayModel).

For the predefined *jnex* data models appropriate data viewer and data editors are defined. The user can easily extend the given library and define new data models and their appropriate viewers and editors.

We put a lot of effort into overcoming the memory leaks, which are not easy to avoid in a Swing environment. The *jnex* viewers are attached to the *jnex* models listener lists as soft references so they do not influence the life cycle of the associated viewers and editors.

2.2 *jnex* and HTML

The *jnex* data models can be translated automatically into HTML. The user can define HTML templates with place holder strings, which will be translated into the appropriate input field.

Even nested *jnex* data models can be translated into complex HTML forms which automatically map their values to (from) the associated data model.

The control flow of the *jnex* servlet is handled with a simple state transition table. Each state in the state transition table represents more or less a HTML page and holds a data model instance (in the user session) and a reference to a HTML template, which is used to render the associated state. Optionally a command instance can be defined for each transition.

2.3 *jnex* und Software Design

jnex changes the focus in software development. The programmer uses its time with designing the business model and with the control flow. Only in a second phase he concentrates on the GUI. *jnex* applications can even be tested on the model layer. This makes a lot of sense, because the actual GUI components are much more volatile and you need a tool to automate GUI testing.

2.4 ***jnex*** is not epidemic

Epidemic frameworks do not easily integrate with other parts, which are not based on the framework. ***jnex*** is not like that. You can at any time mix up parts, which are not using the framework with ***jnex***. The base library of data model types and their associated editors and viewers can be extended with own implementations, which perfectly fit into the philosophy of ***jnex***. Based on the fact that ***jnex*** is open source, this task is really simple because you can look at the code of the existing models and viewers.

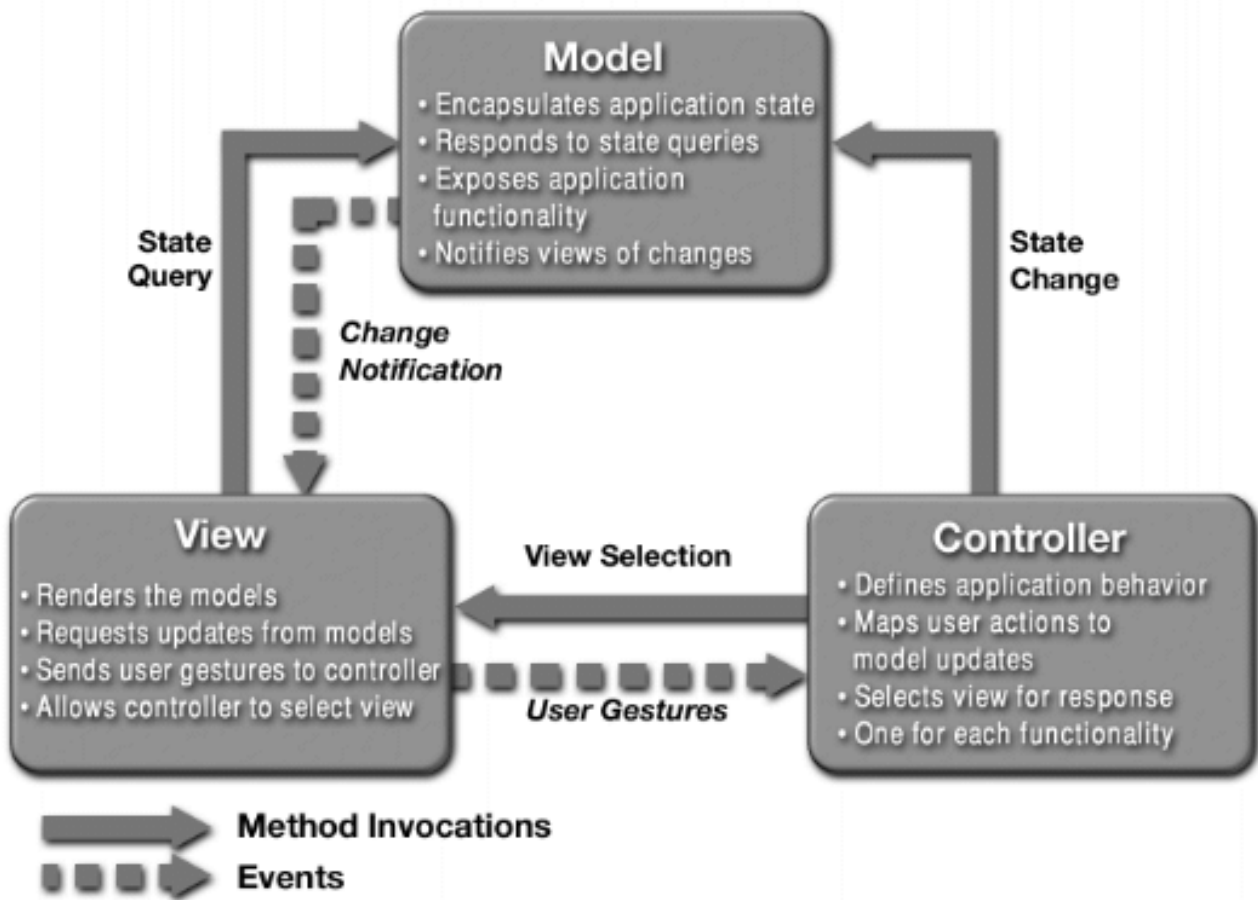
2.5 ***jnex*** is not a code generator

Code generating frameworks do normally convince in very small example applications. The point is that really complex needs come up in the real world (far away from a hello world or a pet store). ***jnex*** is not using code generation, because this would hinder the extendibility. Code generators also force the programmer to learn new syntax based on weird XML schemas.

3 The Model View Controller Pattern

3.1 Benefit

The main target in MVC is the total separation of the business models from the GUI, which is rendering it. For this purpose the controller controls the interaction of the user and the update by managing changes on either side via events. *jnex* models are reactive. This means that they produce events upon any change of the value or the structure. The controller in *jnex* is always coupled to the associated viewer (JComponent). Complex (nested) data models use a controller (viewer), which generated other controllers for nested instances of the associated model.

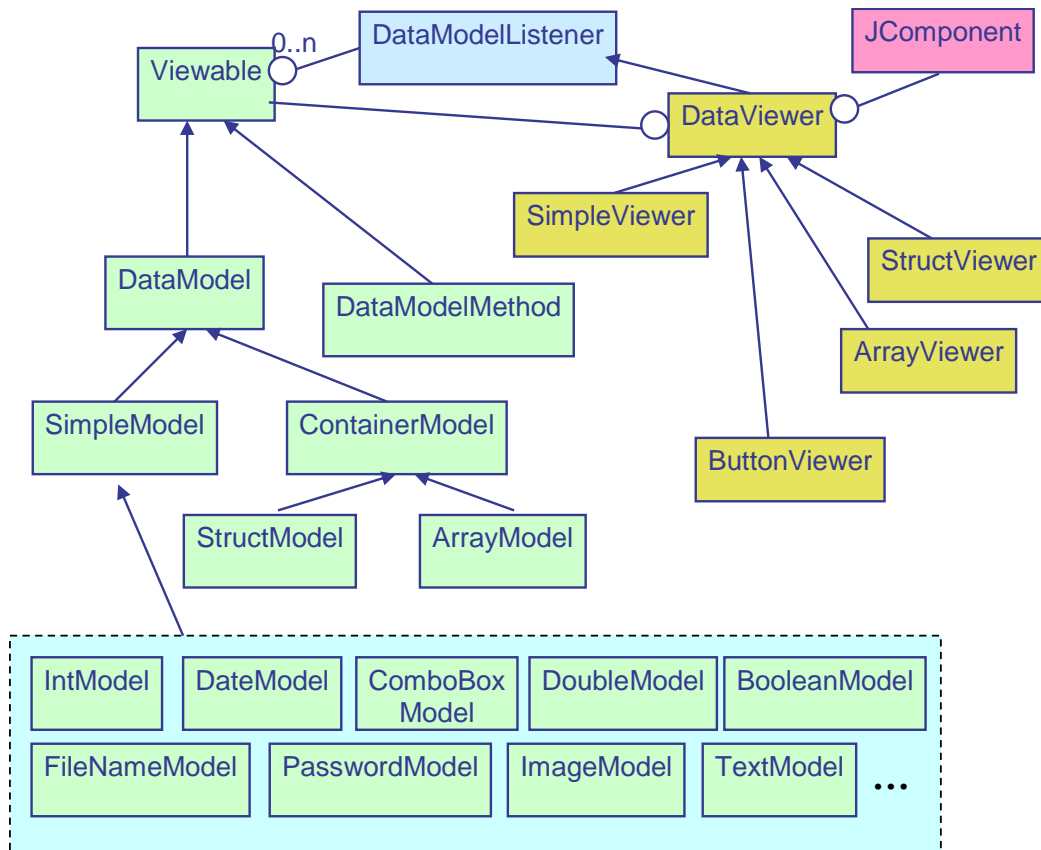


3.2 jnex and MVC

jnex supports data models for primitive types (SimpleModel) and data models for composite types (ContainerModel). Simple models are StringModel, IntModel, BooleanModel, ComboBoxModel, DateModel, CurrencyModel ...

ContainerModels are ArrayModel or StructModel. An Array holds a dynamic number of similar DataModel instances. The instances (children) are accessed by index. StructModels hold a fixed number of DataModel instances (children), which are accessed by name.

For every DataModel the user can attach methods (DataModelCommand). The commands are typically rendered as push buttons.



Type (Data Model)	Default Viewer	Default Editor
String	Label	Text field
Integer	Label	Text field
Floating point number	Label	Text field
File Name	Label	File selection dialog
Bitmap file name	Image viewer	File selection dialog
One out of a list	Label	Combo box
Boolean	Label	Check box
Array	Table	Table with New, Edit, Delete
Method		Push button
Structure	Viewer panel	Editor panel
Password	Label (displaying *)	Password editor (displays *)

3.3 Nested DataModels

```

public class PersonModel extends StructModel {
    private StringModel name;
    private DayMonthYearModel birthdate;

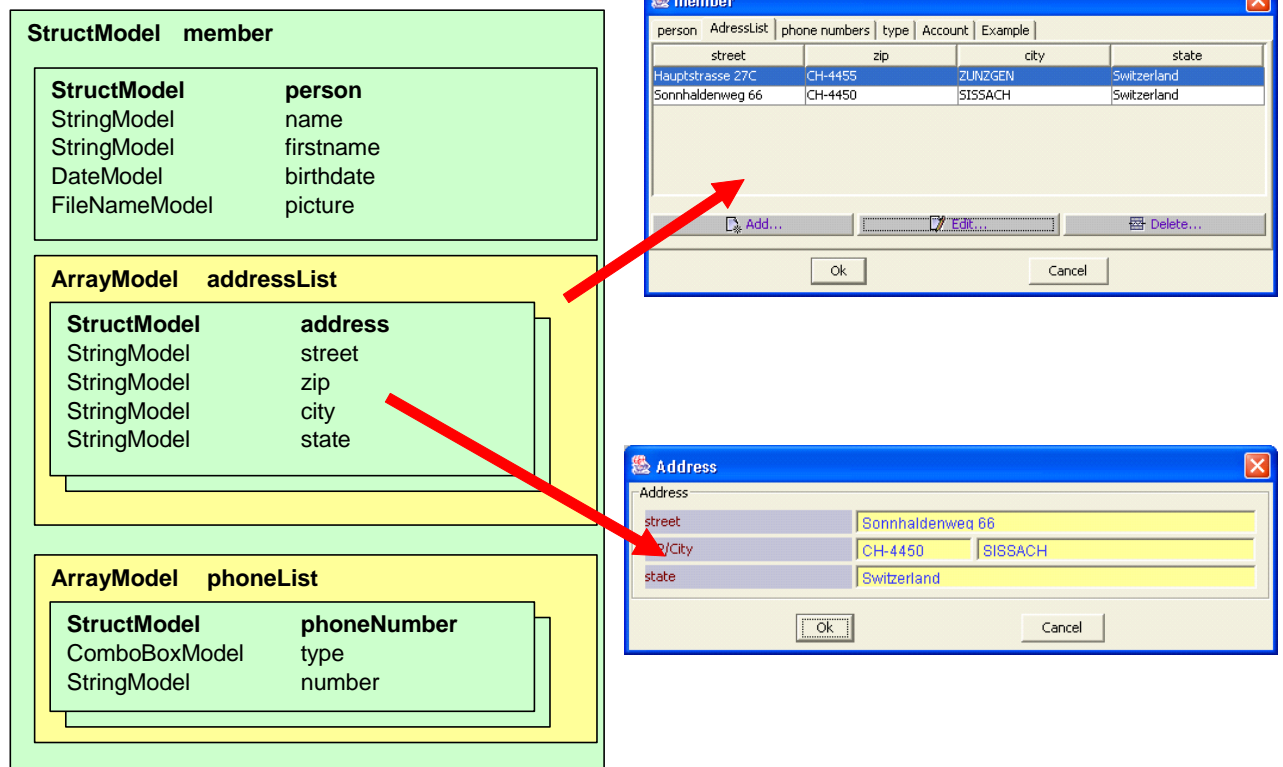
    public PersonModel() {
        super("Person");
        init();
    }

    public PersonModel(String nameString, Date birthdateDate) {
        this();
        name.setText(nameString);
        birthdate.setDate(birthdateDate);
    }

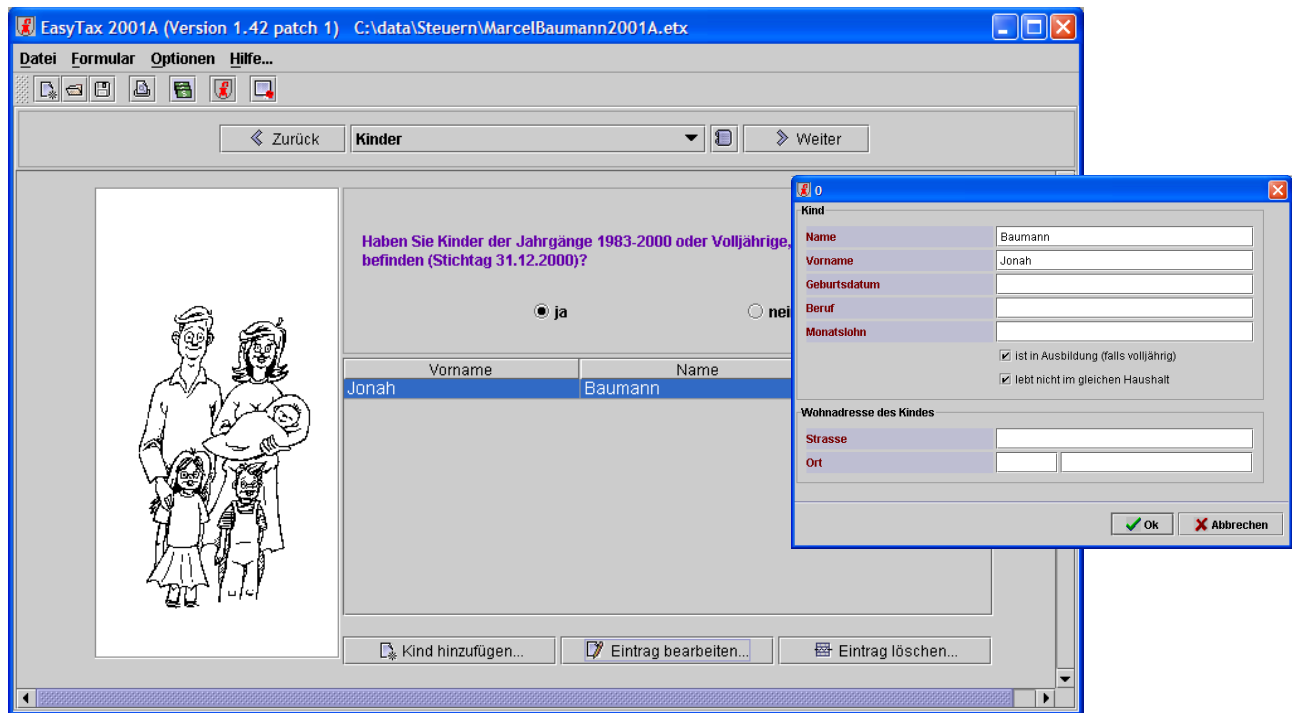
    private void init() {
        name = new StringModel("", "name");
        append(name);
        birthdate = new DayMonthYearModel(new Date(), "birthdate");
        append(birthdate);
    }
}

```

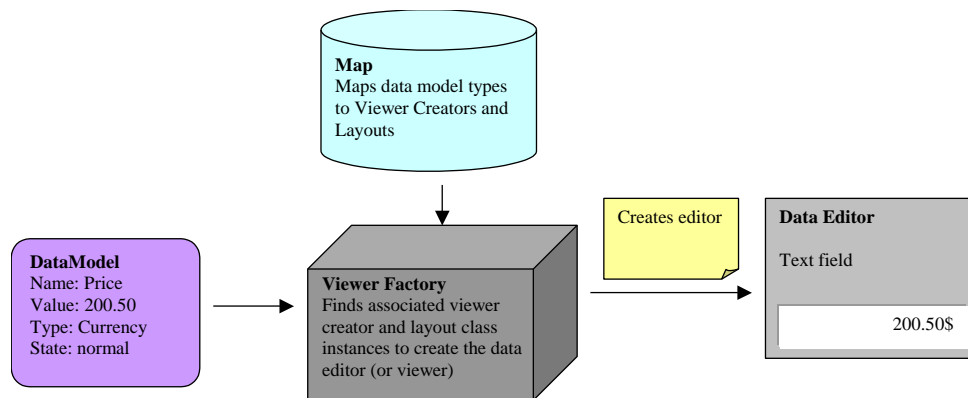
Nested DataModels are translated into GUI components (viewers) using the ViewerFactory. For each DataModel class the user can register associated ViewerCreators with the ViewerFactory. Based on this map (which is initially filled with default viewers and editors for the predefined models) the ViewerFactory generates viewers (or editors) for each DataModel of a specific class.



3.3.1 Using Swing LayoutManagers and JComponent Properties

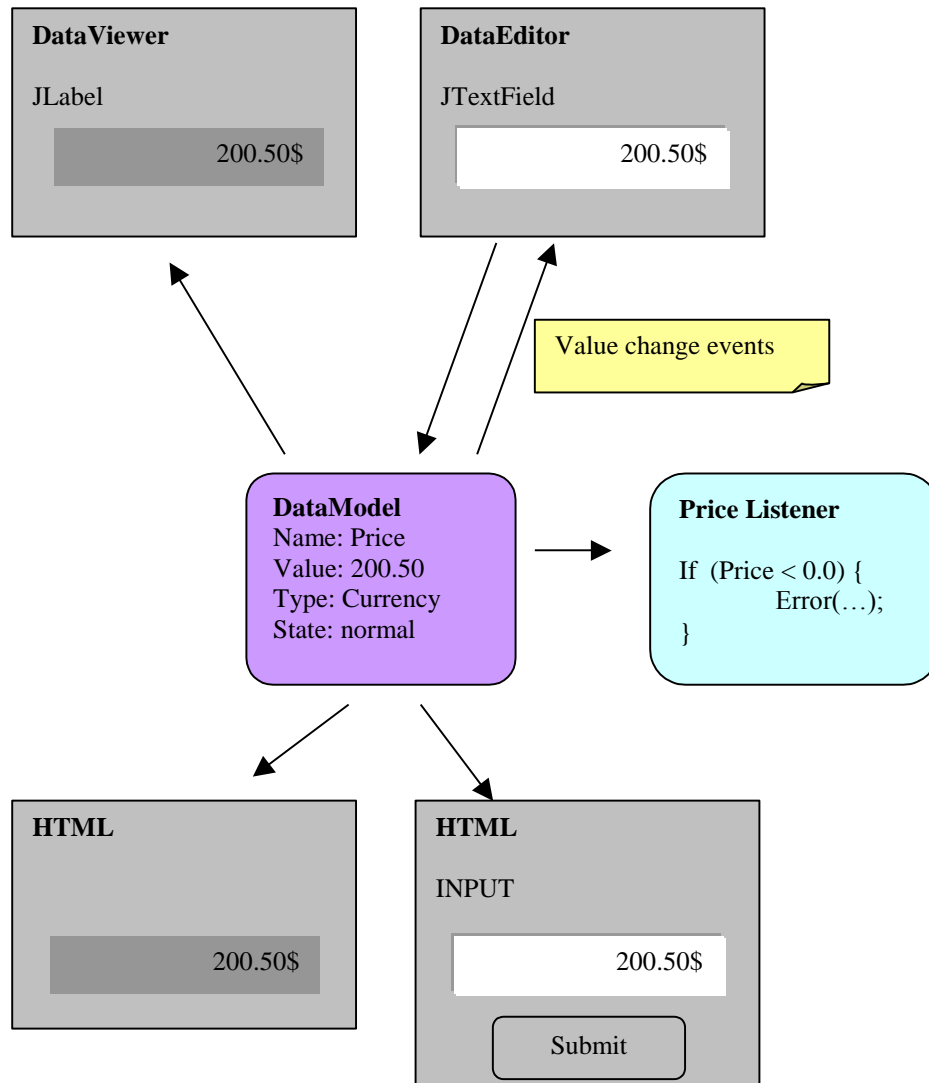


3.4 Mapping of *jnex* data models to appropriate editors/viewers



3.4.1 Events

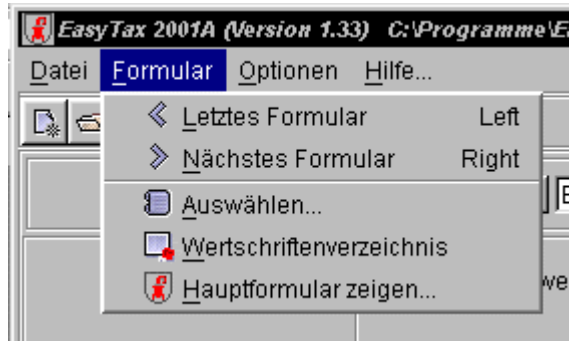
Events handle the synchronisation of DataModels and GUI Components. They can also be used to re-calculate fields, which represent a formula (e.g. the sum of other field). With this mechanism it is trivial to achieve a behaviour like in spread sheets where the user changes the value of one field and the dependent formula fields are automatically updated.



4 Swing Applications

4.1 Pulldown Menu and Tool Bar

For Swing and ULC applications *jnex* has a mechanism to define the Pulldown Menu and the Toolbar. Icons, tooltip texts, and mnemonics are defined in Java property files.

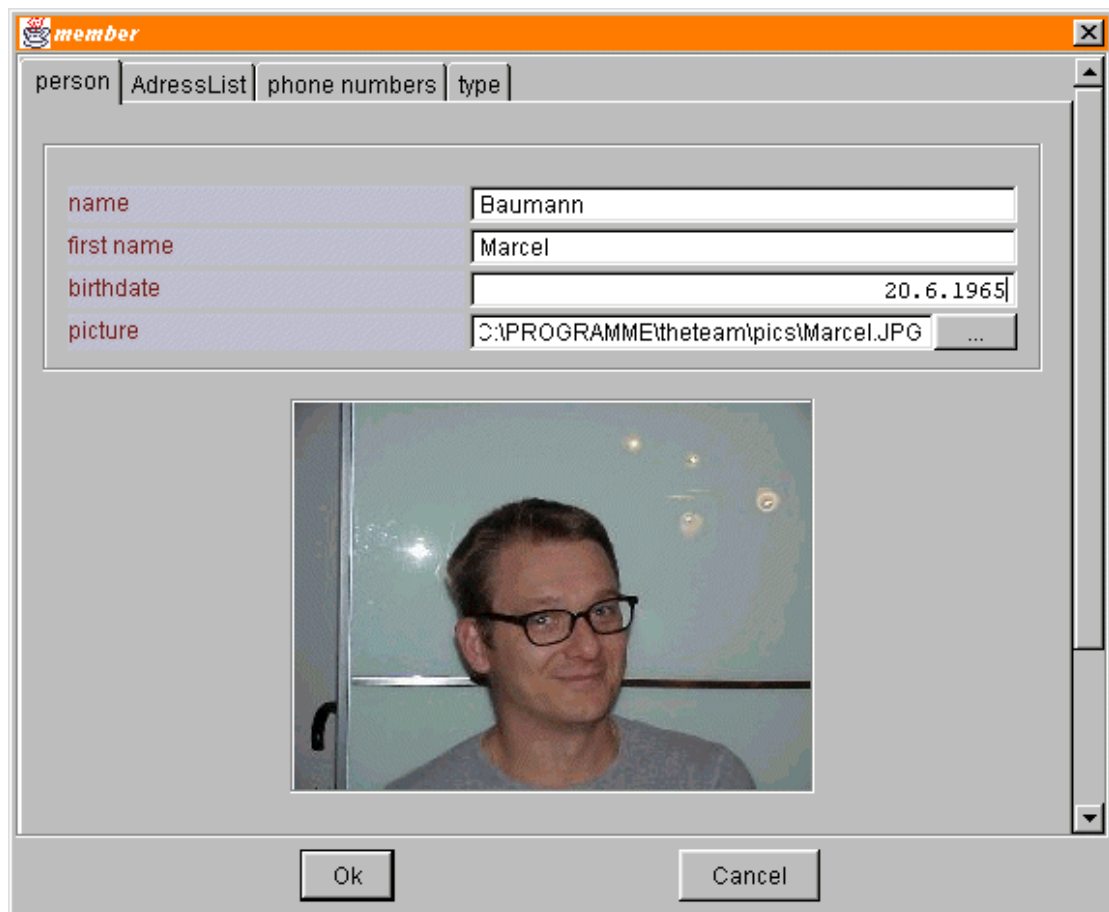


4.2 Internationalization

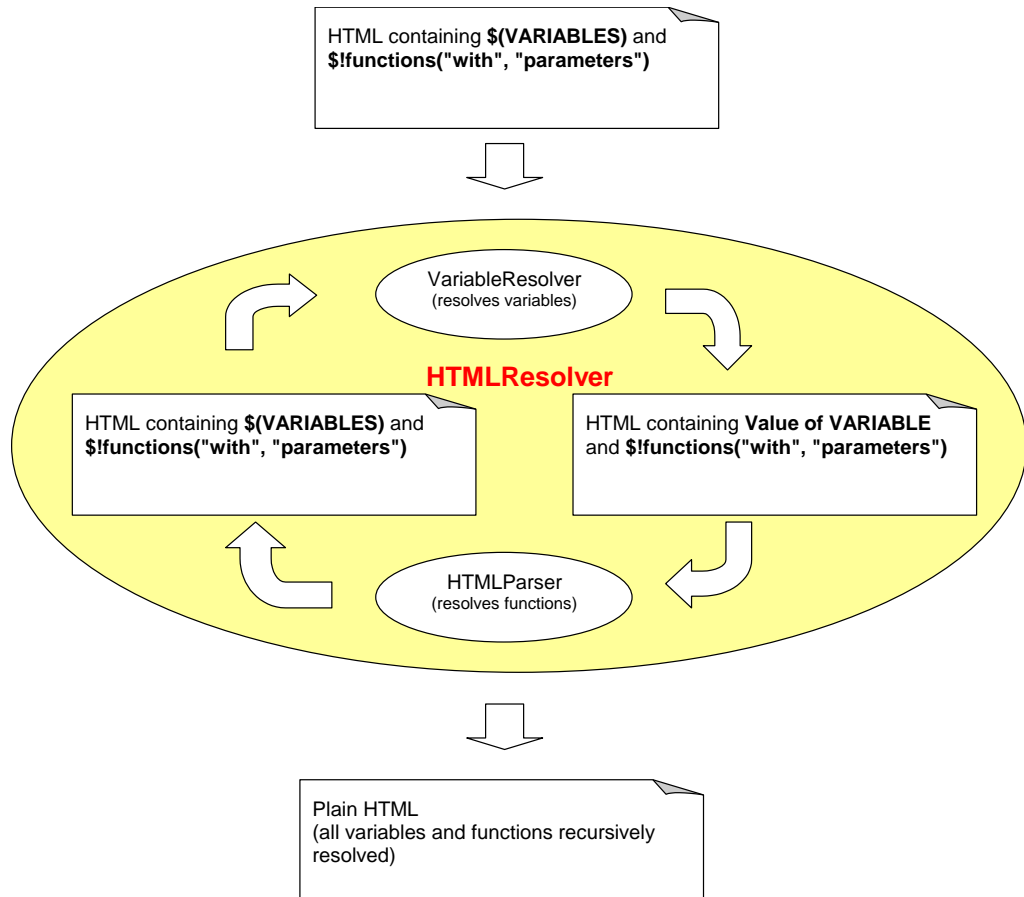
All of the predefined *jnex* GUI components are automatically loading textual information (like labels and tool tips) and icons from the Java properties files. The name of the resource is normally simply the field name of the associated DataModel.

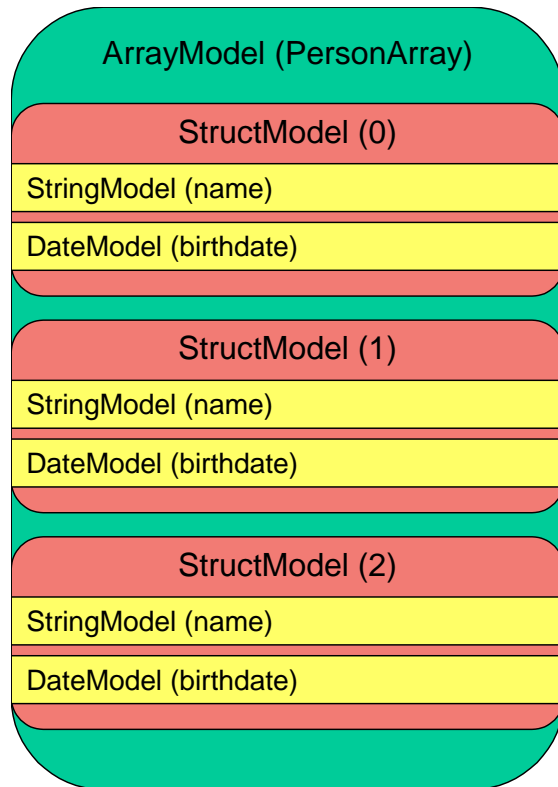
4.3 Dialog Windows

With *jnex* it is very simple to generate dialog windows for data models. Normally it takes exactly one line of code to pop up a window, which can edit the data model content and which is updating the associated data model instance only in case the OK button was pressed. In the background the original data model instance is cloned and dropped or copied in case of pressing the OK or the Cancel button.



4.4 ***jnex*** Servlets (Web-Applications)



**PersonRow**

```
<tr>
<td>!field("name")</td>
<td>!field("birthdate")</td>
</tr>
```

PersonArray

```
<table>
<tr>
<td>!translate("name")</td>
<td>!translate("birthdate")</td>
</tr>
!array(„this“, „PersonRow“)
</table>
```

HTMLPage

```
<html>
<body>
!field(„PersonArray“)
</body>
</html>
```

Name	Geburtstag
Baumann Marcel	20.6.1965
Test Person	5.4.1970
Muster Hans	11.4.2001

5 Persistence

jnex has classes, which help putting data model instances into relational databases via JDBC. The JDBC commands are automatically generated. The mapping between data model field names and database attribute names is simple and can be configured very flexible.

BIRTHDAY_TABLE		
F_UID	F_NAME	F_GEB
92384758990	Baumann Marcel	20.6.1965
84705928538	Test Person	5.4.1970
28743648659	Muster Hans	11.4.2001

```
DatabaseTableMapping mapping = new DatabaseTableMapping("BIRTHDAY_TABLE", "F_UID"
    , new String[]{"name", "birthdate"}, new String[]{"F_NAME", "F_GEB"});

JdbcConnectionHandler.init(new JdbcConnectionHandler("jdbc:odbc:test", null, null));

JdbcConnectionHandler ch = JdbcConnectionHandler.instance();
ch.registerDatabaseTableMapping(PersonModel.class.getName(), mapping);

DatabaseTableMapping personMapping = ch.getDatabaseTableMapping(PersonModel.class.getName());
PersonModel person = new PersonModel();

personMapping.create(person);
personMapping.read(person);
```

jnex data models can be stored in text files because every data model can translate itself into a string which can be translated back into a data model instance.