Create database applications fast

Rapid Database GUI Designer

Documentation

Create database applications in minutes

© 2000-2009 Lothar Behrens

$Revision: 1.2 $

Table of contents

Introduction

Creating database applications is not easy. Especially the creation of database applications, like CRM systems or a simple CD cataloging system.

With this project you will have a starting point in creating database applications faster. The method to use design tools is a great help. Those design tools eases the development because you begin modelling your database view in combination with some informations for the form design.

Also the system helps you to fastly present a first solution what is called the prototype. With that prototype you can discuss with your customer in more detail, because they see a real working application, thus maybe see gaps in the logical design very early.

Also with the prototype you can test the application before any line of code has been written and with a real database.

Concept

The prototype application can either created dialog based what is a bit more complex, but it can be attended in more detail settings were may not yet supported by additional modelling tools.

Or you use a designer that understands to create XML files that principally could be imported.

Included in this software package is a UML modelling tool ([UML](#)) that I support with an import function (template).

But there also could other tools used that supports an usable XML format. With usable format I mean that the modelling is useful and also could be translated into the internal format.

Quickstart

In this chapter you learn, how easy it can be to start with this tool. The samples of the UML models are based on version 4.9.3 Mac. The Windows version is identical and should be replaceable.

Creating an UML model

To create an UML model you start the included and installed software BoUML. In the 'Project' menu you select 'New' and enter the project name. Enter 'BoUML'. You then will see the following warning message:



Close the window and select 'C++' as I prefer always.:
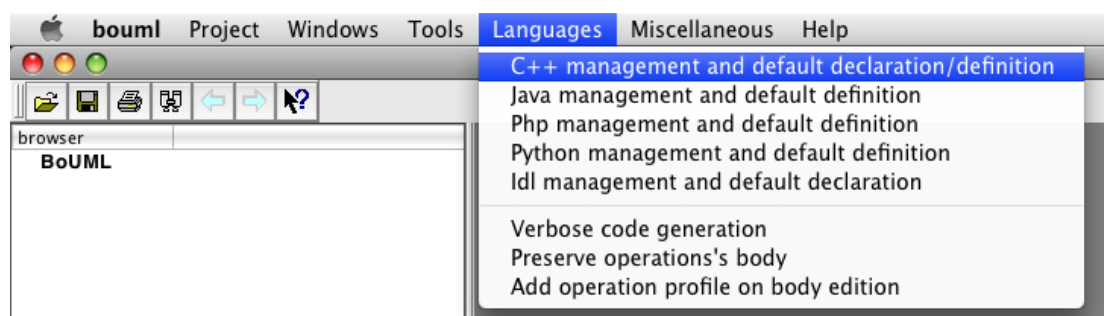
This setting relates to the internal code generation of the UML modeller and doesn't matter here.

Package names

There is an important difference between projectname and package name. Until now you have entered a project name that is used at the same time for the package name. Later you have to remember to the package renaming if you rename the project name.

Creating packages

To work with multible packages, you could create new packages inside an existing package. Here you will see, how:
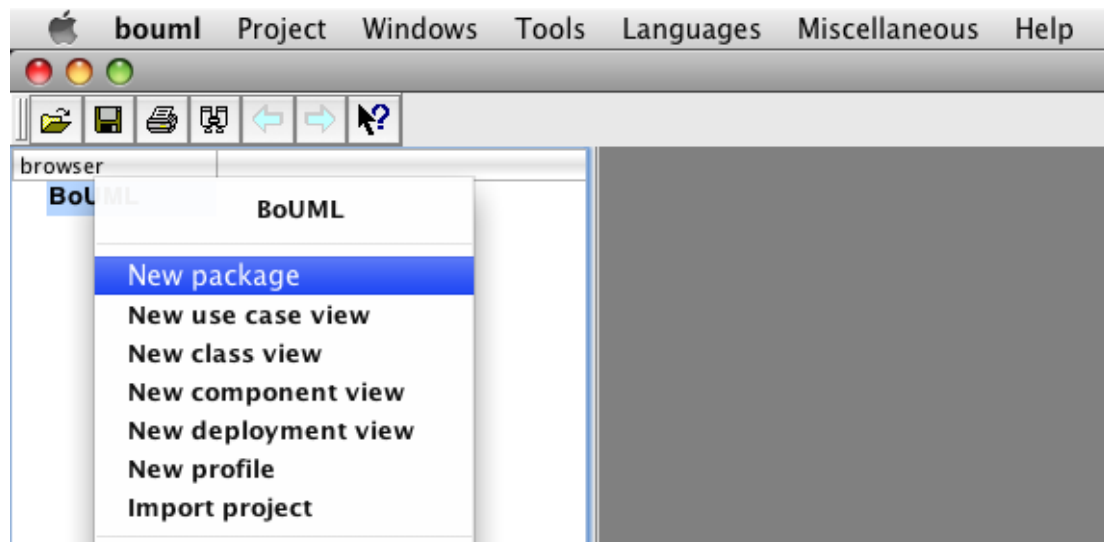
Multible packages are not supported yet or are not tested at least. If you use a revision control system like CVS, it is recommended to create a package to avoid renaming issues with the project related to the revision control system.

A package name is equal to the name of the application. The inner most package name is used for the application name. We now use a package name 'CRM'

Create a classview

After you have created a package, you create in there a class view. See the picture above to spot the next menu entry for creating class views.

The name of the class view is not relevant for modelling the application and can be named as you like. There is also the possibility to use multible class views to structure your design to logical pieces. Use 'Start' as the name.

Create a class diagram

To use graphical documentation and modelling you create class diagrams. This is done with 'New class diagram' as depicted in the picture above. Use the name 'Tutorial'.

Now you will see the following:
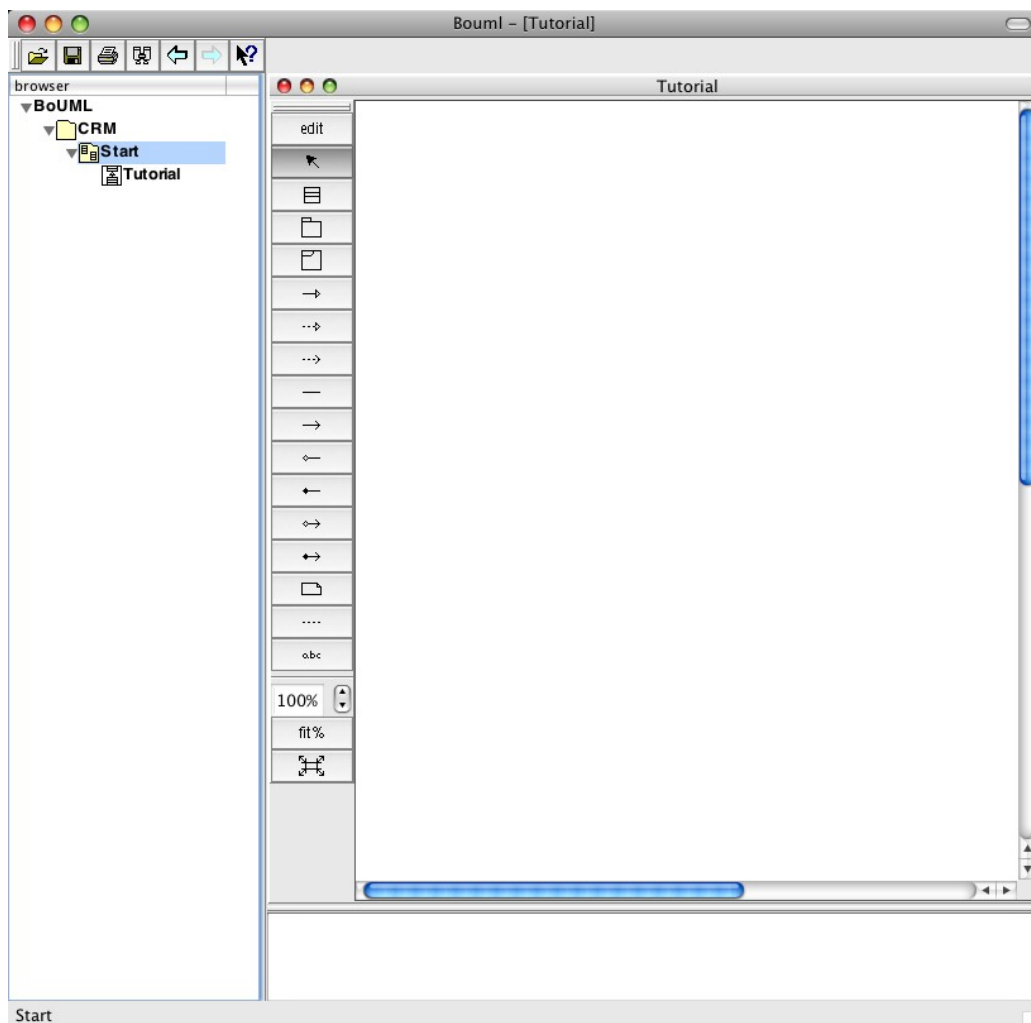
Start modelling

Now you have created a project that contains a class diagram with them you could start modelling. Save the complete project directory as a template for new projects.

Create some classes

You create new classes with the symbol [⊟]. Thereof you get the following class dialog to enter the new class name.

Uml
enter class's name : [   ]
(OK)   (Cancel)
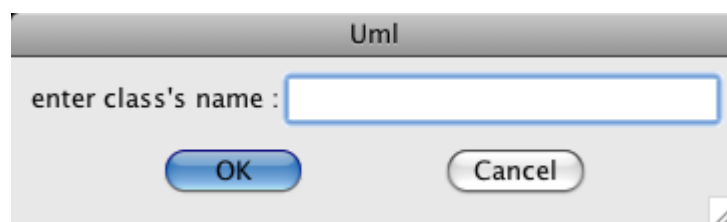
Do not use a classname multible times. You could use the same class multible times in a class diagram. This is useful for an overall overview of classes without all the details in each class.

Relate classes

Classes usually have relations to other classes. An important relation in our modelling is the following one: ◇—
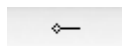
The side with the diamond applies to the referenced class. The referenced class therefore is the primary class and the other is the foreign.

In the following sample we will create a class named 'Kunde' (customer) wich is referenced from a second class named 'Bestellungen' (orders).

It is then recognized as a one to many (1 to N) relation. Here we have orders that are related to customers. Therefore the diamond is on the side of the customer class.

There are other presentaion or modelling alternatives to achive the same result, but those are not yet supported in the prototype application import method.

Therefore you use this notation for database relations.

Until now the model looks like as follows:
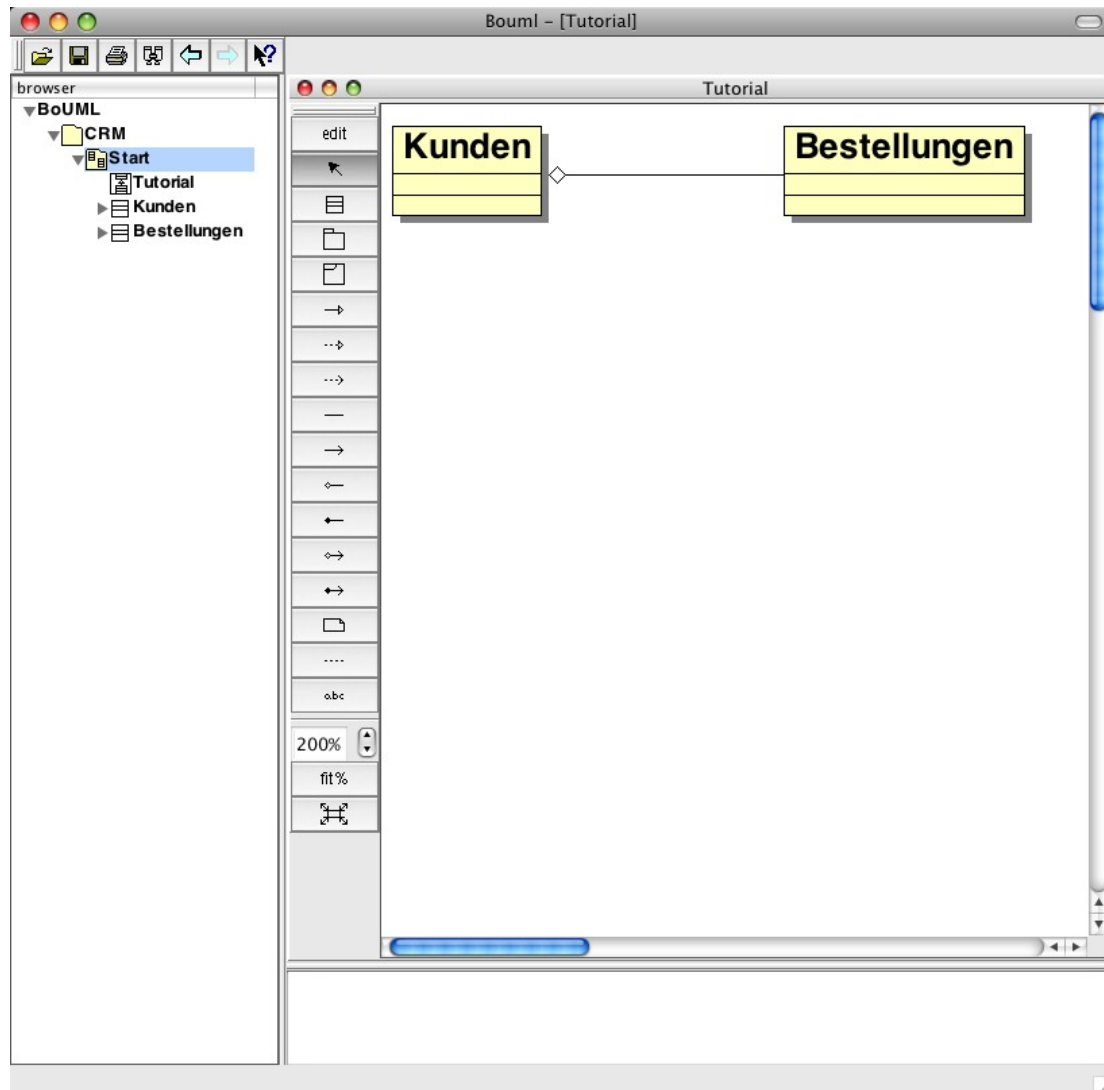
As you see, the tree structure has been growed. You now could right click on these classes (tree or diagram) to do the following actions:
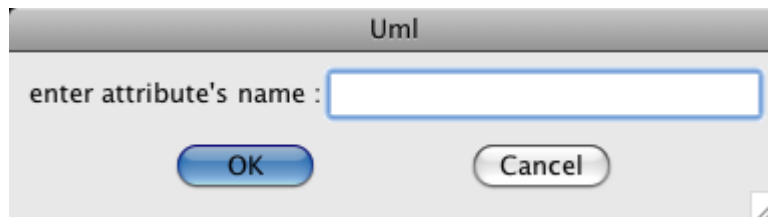

- Create attributes
- Create operations

Attributes are columns in the database table. Therefore the class name results into the equivalent table name where the data is stored later.

Operations are handled as controls at the form you later will see. Until now there are supported only few opertions. One of them is the execution of a stored procedure.

Creating attributes

Because the fact that database forms are looking empty, we now create some attributes we think to be required for now. To do this you right click on the customer class and there click on 'Add attribute'. Then you will see the following dialog:

Assign to the customer the following attributes:

Firma (company name): string

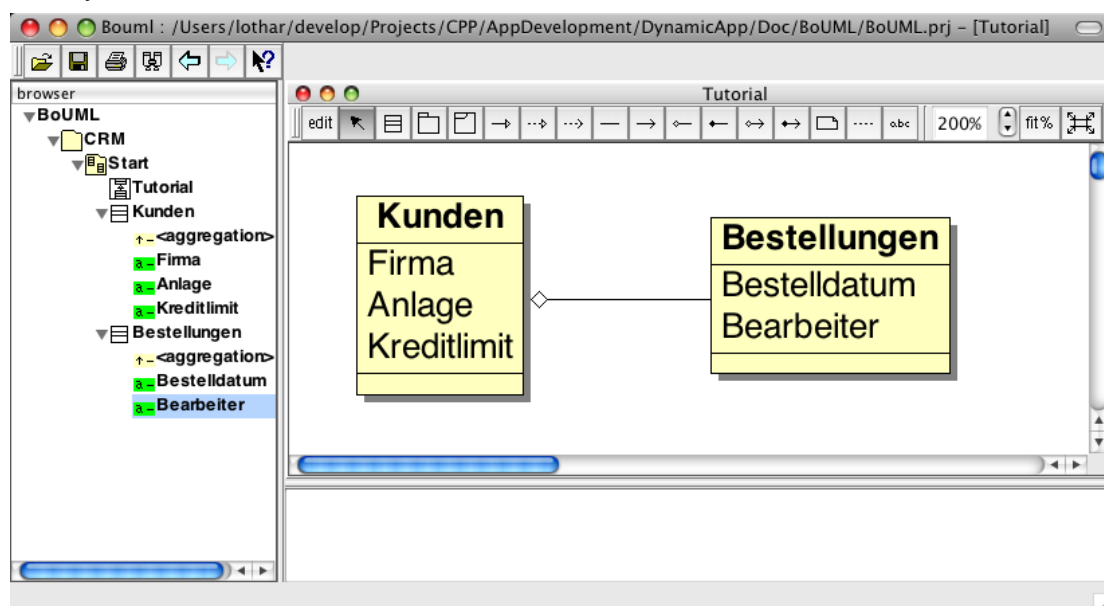Anlage (creation date): date

Kreditlimit (credit limit): float

The class orders becomes the following attributes:

Bestelldatum (order date): date
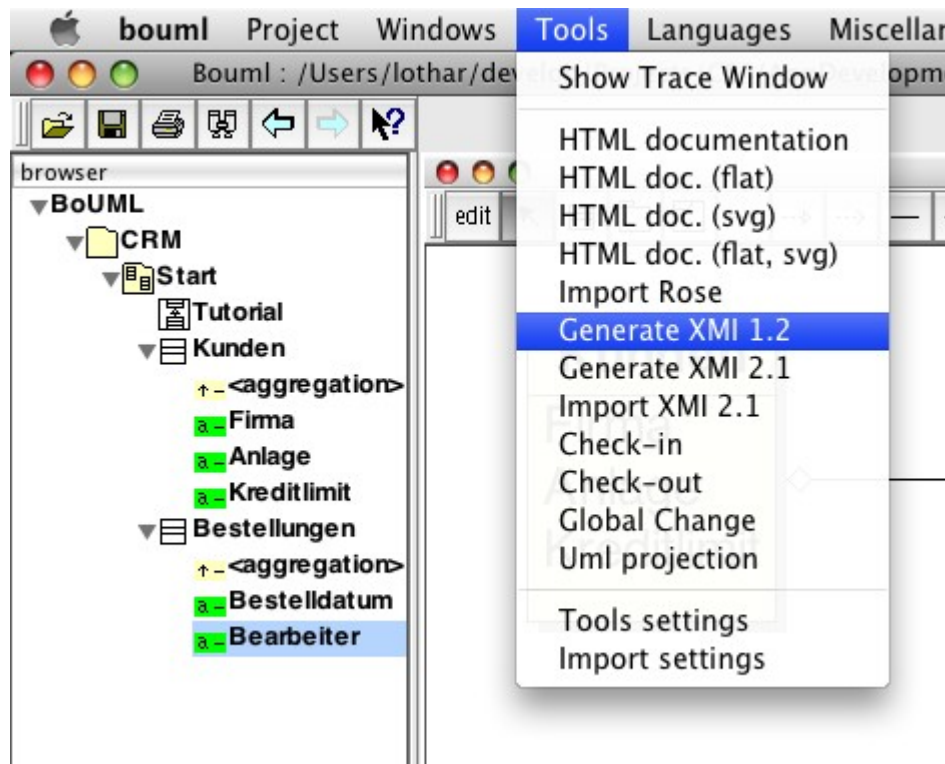
Bearbeiter (sales rep.): string

Now your UML model looks like this:

Export the UML model

After you have finished your model, it is time to test the application. Therefore you export the UML model as follows:

Note: This is an initial UML model. It has no detailed information as of modelling the forms and the database tables separately. Thus it is important to select 'Generate XMI 1.2'

Now there appears a dialog that has been created from a support application of BoUML. It's name is 'gxmi'. The application probably may not visible. If that is the case please look at the task list. Select an export filename or type in a new one in the field left of 'rows'. (Browse, the layout on Mac is broken):
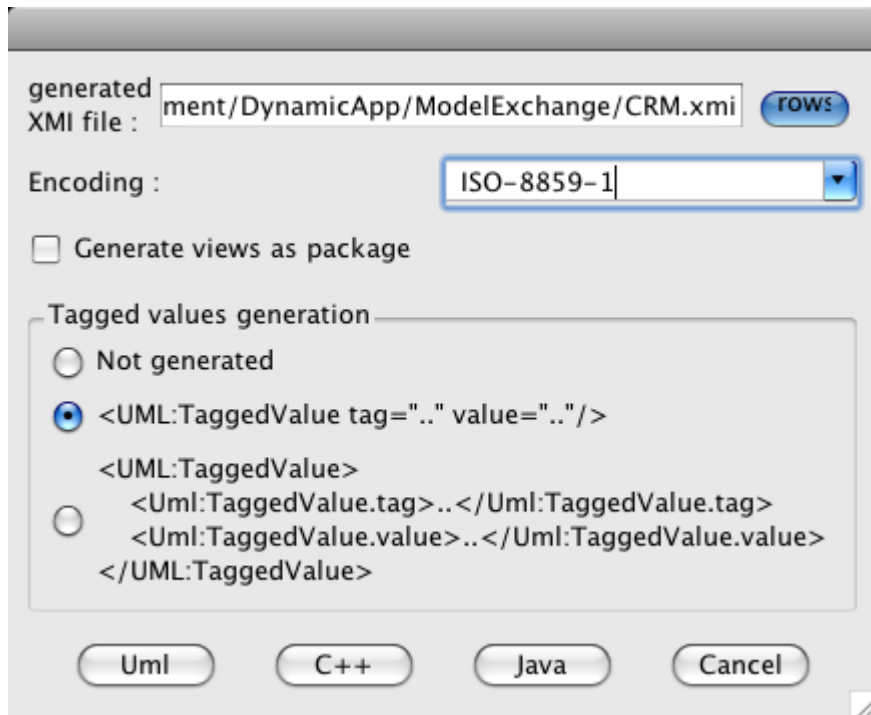
Take the same settings as depicted. Important here is the settings of the encoding. It can't be empty. The filename is later used at the import into the prototype.

Import UML model

To use an UML model created in BoUML, it is required to import it. Therefore you start the main application (wxWrapper). On the left side are some settings you need for the import. Fold in the first two groups. You don't need them now.

| Properties	
▶ Application Database settings	
▶ General	
▼ Transformation settings	
Ask for other XSL files	True
XSL file for application database	<application>
XSL file for settings	<settings>
XSL file for system database	<system>
▼ lbDMF Manager Import Definitions	
DB Name	CRM
DB Password	<dbpass>
DB Schemaname	public
DB User	<dbuser>
XMI UML input file	<UML XMI file>
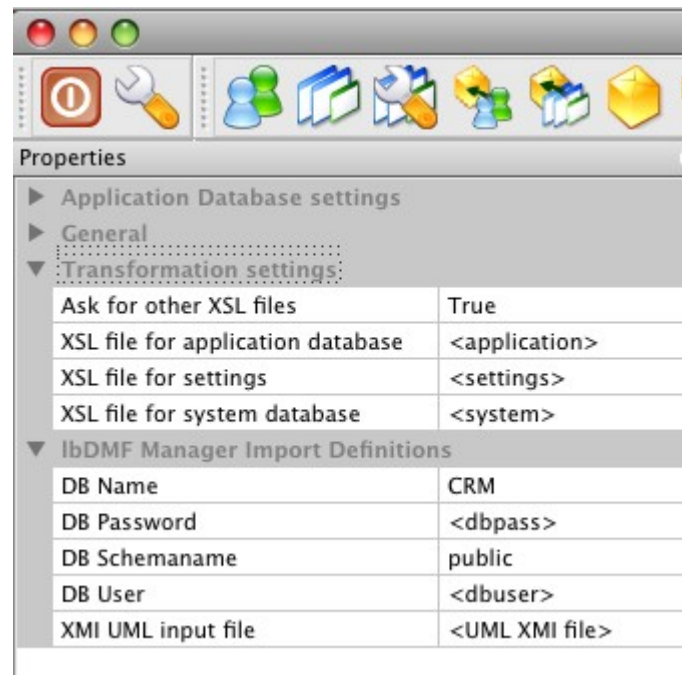
Choose the file you have created with BoUML in <UML XMI file>. It will be used in the import procedure.

To correctly import the applicatin model you need to know of what format your XMI file is. Shortly we have exported the UML model in XMI 1.2 format, thus the following settings are to be done in <application> and <system>:

<application>:   xmi1.2_2SQLScript.xsl

<settings>:   XMISettings.xsl

<system>:   xmi1_2_2_lbDMFSQLScript.xsl

These files are in the folder XSLT_Templates/XMIToDMF. The file XMISettings.xsl is to be entered manually in that directory if it doesn't exist. In that file are settings that controls how the import is done. Sa sample the type of the database system.

If you have entered the settings, you are ready for the import. In the following picture sou see how to start the import:
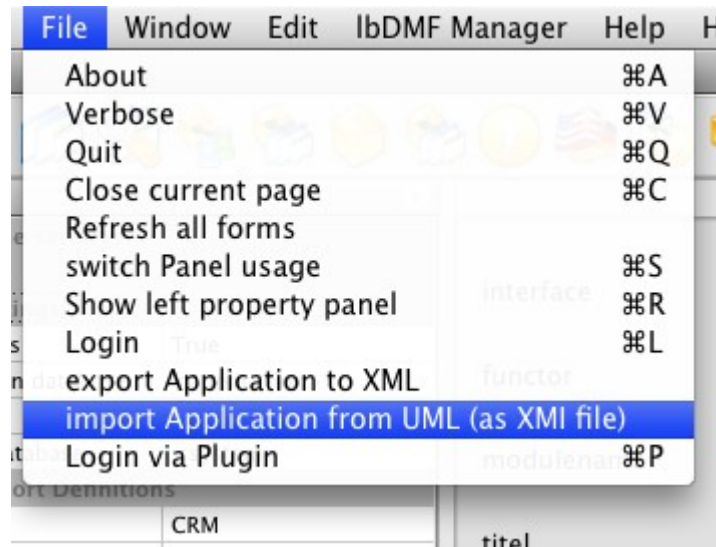
The import is done in two steps. The first step creates the application database that represents the physical datamodel of the UML model. The second step imports the application configuration into the system database. You could skip the first step if you have a database.

Note: If you want to create an UML model from an existing database, you create a dummy UML model containing only one class with a name 'Dummy' for sample. Import that UML model and skip the creation of the application database. You don't really need it now. Check in the application settings of the newly created application model the database user and the password. Also you need to deselect the usage of the database backend for the application database as shown below. Then export it when you have it running. This automatically tries to collect the database model and includes it in the exported XMI or XML file.
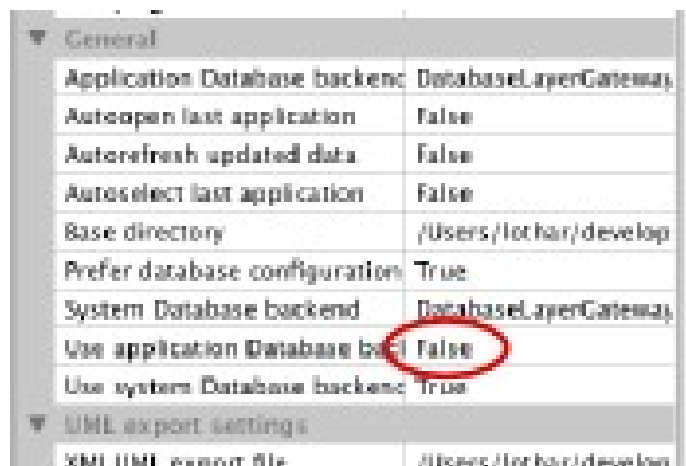
But using an existing database is explained later in more detail.

After you have imported the UML model ensure you have unchecked the menu entry 'Autoload application in the 'Edit' menu. Quit the application, start it again and login to the newly created prototype. In our sample the user is 'user' and the password will be 'TestUser'. The application you start will be 'CRM'.

The first prototype

The first prototype looks like the following picture if you click once in 'File' 'switch Panel usage' and have entered some data. The application does not contain a second toolbar because the UML modeller can't directly model this.
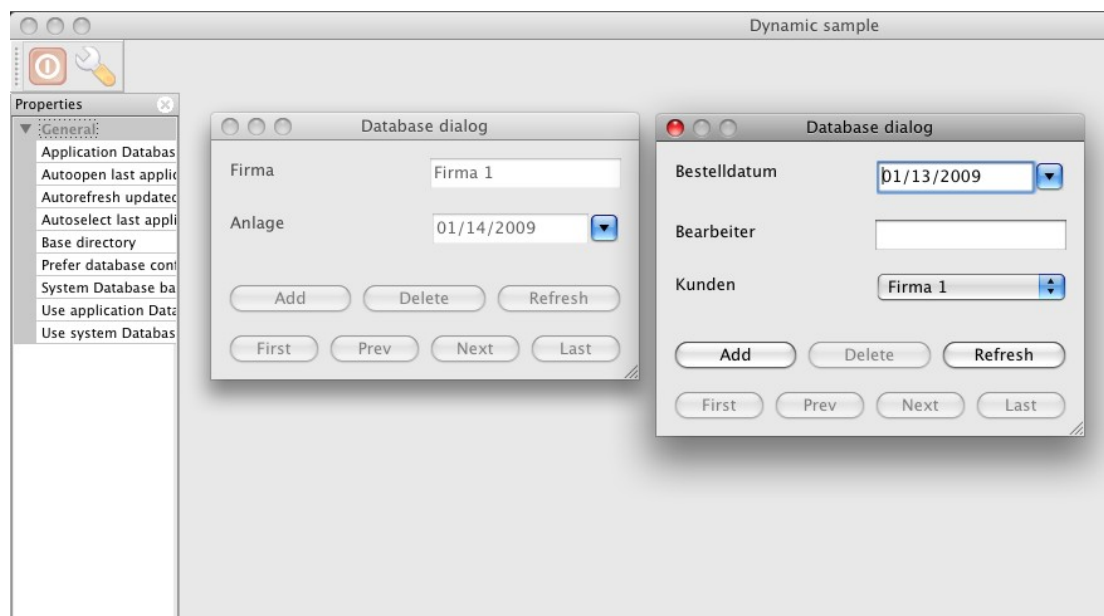
You have an existing database

With the included capabilities of the prototyper it is possible to create a prototype for an existing database application. There are many reasons why you want to do this.

- You want to replace an existing application
- You want to provide a separate application with parts only
- You want to create a web interface and therefore start with a prototype

There are truly more reasons, but the steps remain the same.

The sample Postbooks

As a sample here we create a prototype for the Postbooks application. It will show the capabilities of the prototyper. The following screen shots of extracting the database model from PostgreSQL are taken from the Windows platform and are not shown here. The equivalent Mac screenshots are comparable.

Setup the ODBC connection to the database

To use an existing database for a prototype you need to setup an ODBC connection to it. Use the informations of your database vendor. This description has been tested on a PostgreSQL database. Other databases are possible too, but there are probably required changes in the XSL file to do before you proceed. Developing XSLT templates or changing them is handled in a separate documentation.

Create a new UML model (Quickstart)

You need an empty UML project in that you add one class. Name the project and the package Postbooks. Have a look in the chapter Quickstart how this is done. Attend on the class name and do not name it like one of the tables in the database you like to create a prototype for. Name it 'Dummy' for sample.

## *Export of the model in XMI 1.2 and import*

You export the model as of described here: Export the UML model and import it in to the prototyper. Thereafter you check the database access settings in the application settings ('Anwendungsparameter' in 'Anwendungen') in the newly created application model. The important settings are 'DBName', 'DBUser' and 'DBPass'. These settings must reflect your ODBC settings to access the database. **Please note that we still using XMI 1.2 format.**

Prepare for Reverse engeneering

To reverse engeneering a database model, you need to have an existing ODBC setup. Look here: Setup the ODBC connection to the database. Then you need to run the newly imported application. Click on 'Edit'->'Autoload application' to deactivate this feature. Then you have the ability to start another application. When you start exporting, the database metainformation will be retrieved and included in the export.

Export the application as XMI 2.1

If you have imported the dummy XMI 1.2 model you need to make the following settings in 'Use application Database backend': 'False'. Also setup the XSL files in the group 'UML export settings'.The base directory is XSLT_Templates/DMFToXMI. 'XSL file for UML export' is 'gen_DMFToXMI.xsl' and the file to be generated is 'XMI UML export file'. Name it as you like. Note also the XMISettings file.
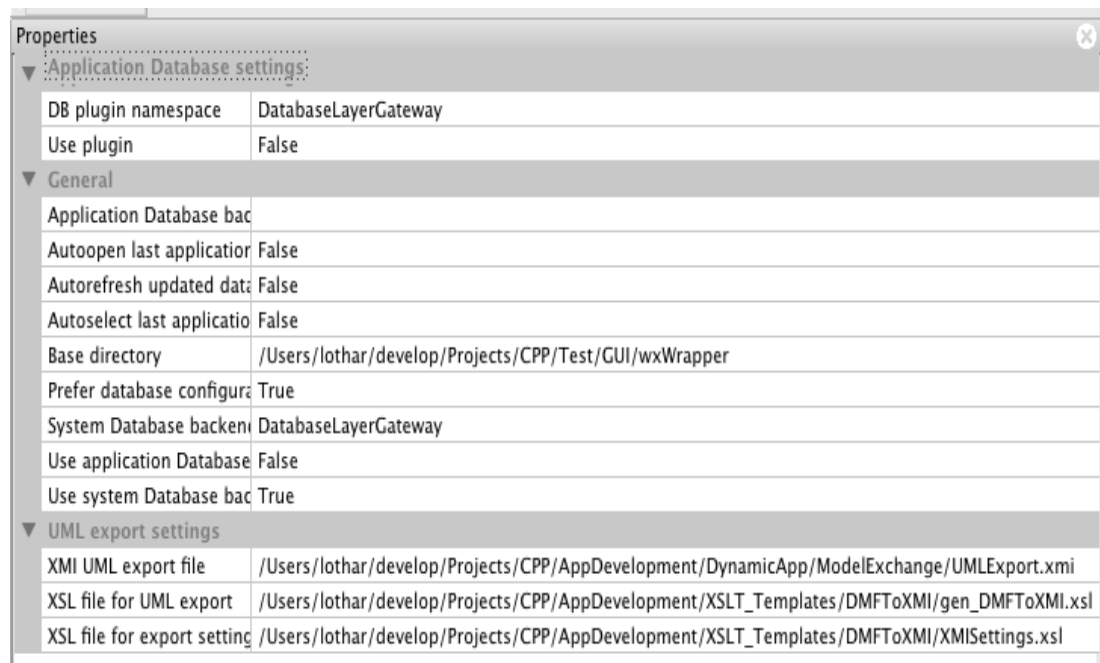
In your environment these files may be at another place. The sample screen shots are made in my Mac OS X development environment. On the Windows standard installation these files are located here: c:\lbDMF\XSLT_Templates\ DMFToXMI.

You could only create XMI 2.1 files with this export method. If you like another format, you need to change the template, but better create a new one.

There is currently also a limitation when you have created XMI 2.1 files. There is no way to reexport them as XMI 1.2 files. The XMI 1.2 files are used as a 'first starter' with enables simpler modelling.

Note: The sample database, here Postbooks has two tables with two primary keys each. The templates that will import this model from UML as XMI 2.1 files are not capable of handling multible primary columns. Remove the second <<key>> stereotype tags in each of the following Entity classes:

The classes: aropenco, payaropen.

Other problems haven't arised while my tests with Postbooks. But in general arising problems can be narrowed by using manual transformation and check the resulting files using XSLTPROC.

In any case you could correct things by modifying the templates.

Read more in the upcoming 'Developing XSLT templates'.

Import the XMI 2.1 model in BoUML

If you have exported the model, open BoUML to import it. You could use the dummy project created earlier, but delete the elements as follows:
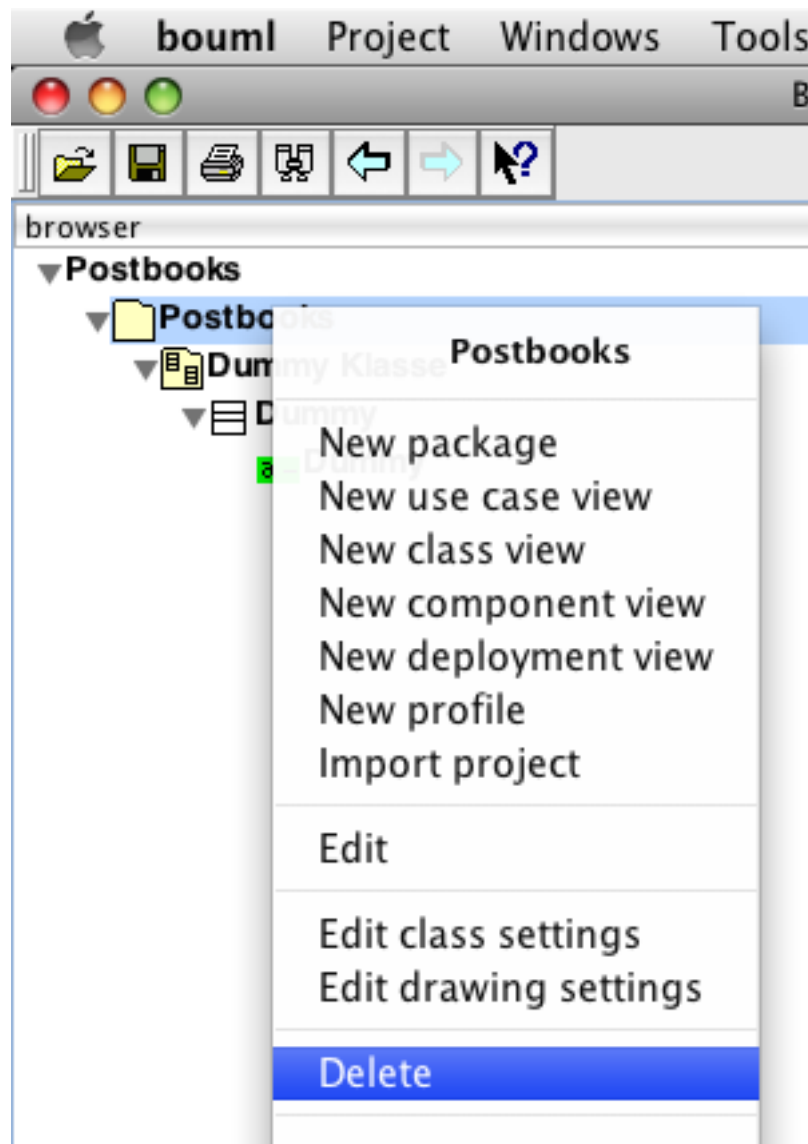

After that the elements are still visible, but marked as deleted. You could revert the deletion, start with the import or reopen the project to remove the deleted elements.

Start the XMI inport into BoUML

After you have created an empty UML model or deleted all stuff therein you could start the import as follows:
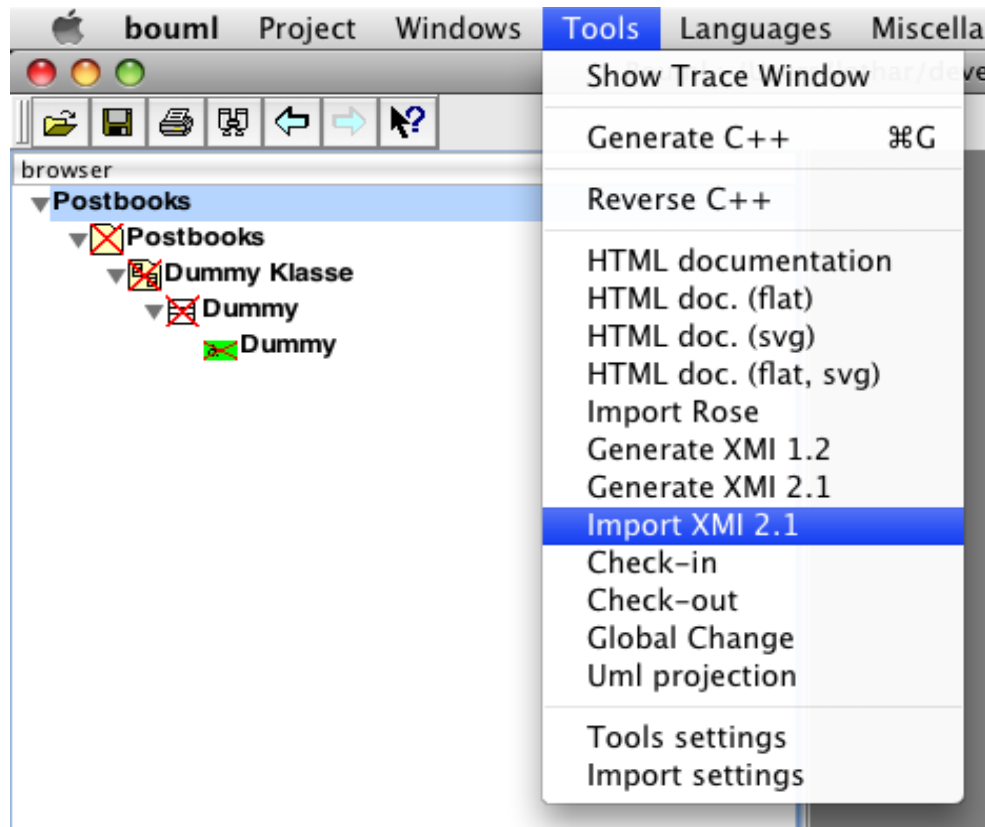
You see here, I haven't reopened the old model, but the import is possible. If you have done the import you will see how it looks like on the following page or like the next when you reimported the export from BoUML :
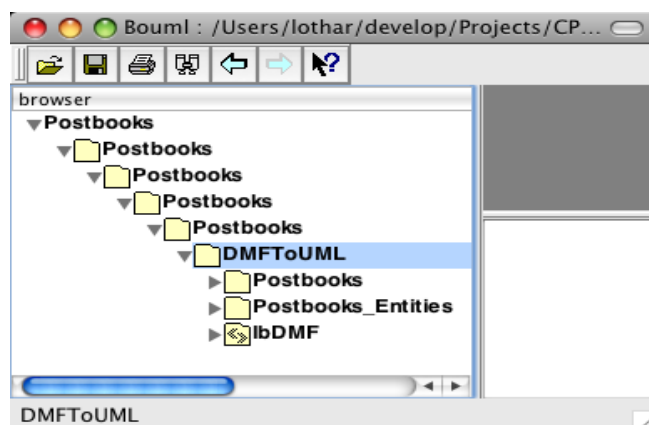

In that case you could move out the selected node into the outer most element and delete the other to correct that.

Postbooks application as UML model

Her eyou see some classes that have been reworked after the import into BoUML and are declared as forms (stereotyp <<form>>). Also you see the relations from forms to their entity classes (stereotyp <<entity>>). Entity classes represents the tables in the database. The forms are created by duplicating them and then unnessesary relations are removed, before starting creating relations. This is a BoUML feature I don't like here, because it is little too much work.

In that way the new application comes up based on an existing database. You could select specific tables only to provide forms in the new application (by duplication and moving them into the correct package).
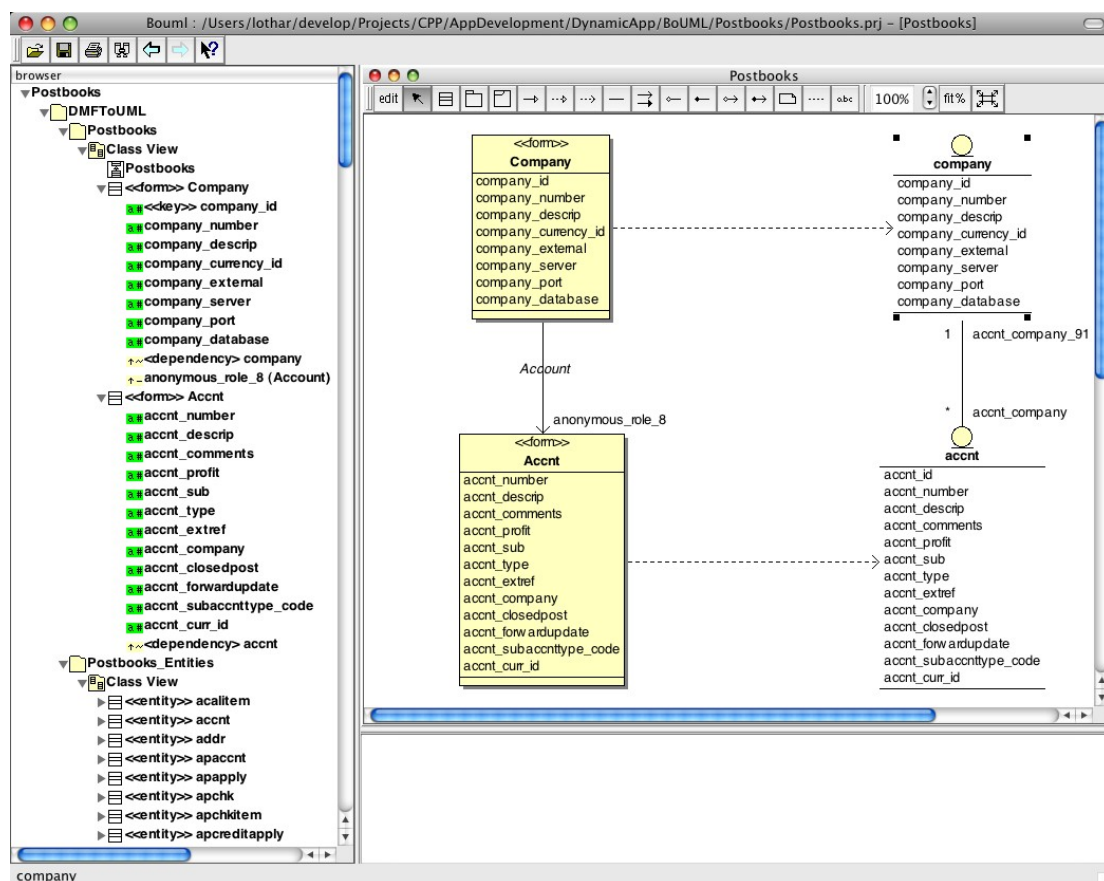
That in this way created UML model (firstly the package 'Postbooks' is empty) could be extended by your means. It could be created multible forms based on the same table (for sample for different WHERE clauses who currently get lost due to missing XML elements where to store the where clauses).

The tables are located in the package 'Postbooks_Entities' and the form classes are located in the package 'Postbooks'. In the diagrams you could show them together. Note that it will be better to create the diagrams in the entity package, as I have noticed there would be drawn all relations.

For an indeep information on UML modelling you could read the upcomming documentation about UML modelling.

Postbooks application as prototype

Here you will see two screenshots of company and accnt table in the prototype on Windows. I have currently no Unicode drivers for PostgreSQL on my Mac, thus the resulting screenshots are done using Windows.

The Company form shows some of the collumns in the table. Note the detection of different types:
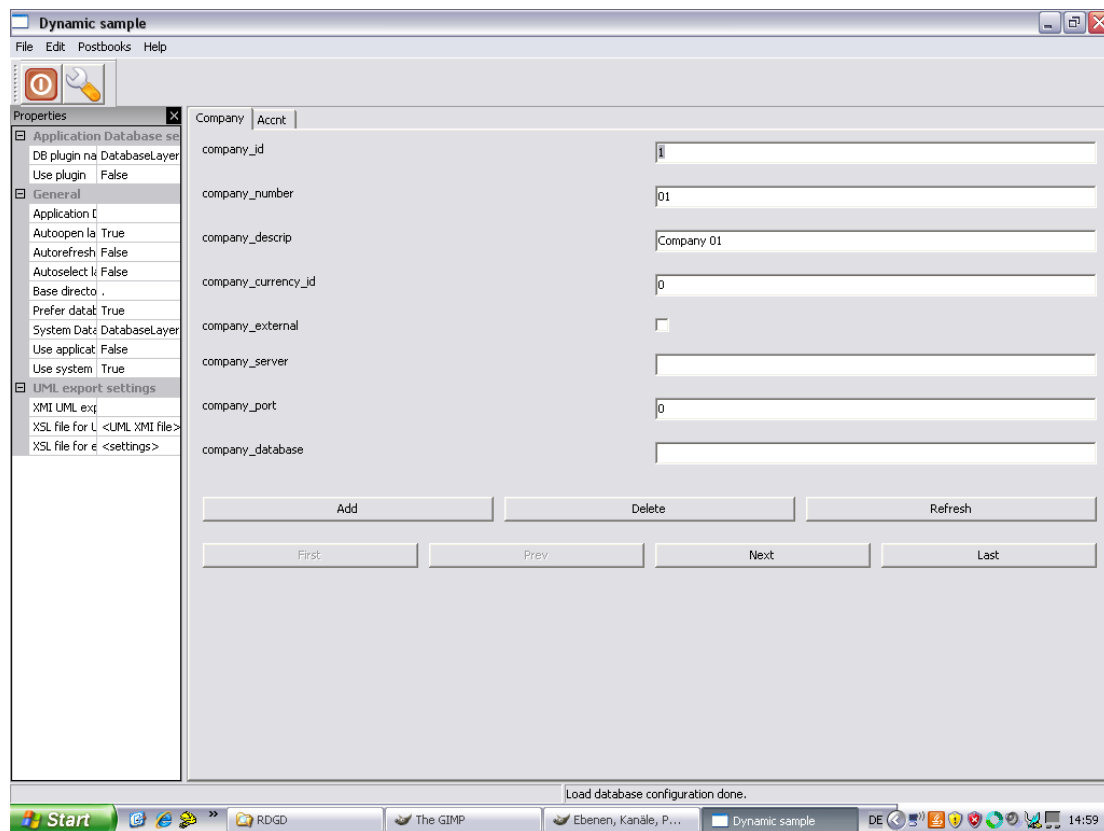
Note: The sample application shown here does not exactly represent the UML model shown above. There is an arrow from the 'Company' to the 'Accnt' form class (<<form>>). This will result into an additional button in the above form.

Here is the Account (accnt table) showing also some drop down boxes for the foreign keys that are shown:

Here I have not shown, that the application will ask you what field should be shown in the dropdown boxes for each foreign key. This happens once when you don't model this before in UML. If you reexport the prototype to UML (XMI) you will get an updated model from the gathered data for columns to be shown instead the foreign key itself.