Morfik: A new approach to web applications

Michaël Van Canneyt

July 27, 2006

Abstract

Morfik is a new product which has the potential to make development of web-applications as easy and straightforward as desktop application development. What sets it apart from other web-application development platforms is it's unique architecture, which will be explained here.

1 Introduction

A new trend in web-application development is using AJAX. AJAX stands for Asynchronous JAvaScript and XML. It's a technique for creating web applications which behave almost as regular desktop applications. Using a clever combination of XHTML, CSS and JavaScript, it is possible to create applications that run in a web-browser, and which behave almost as a desktop application: The javascript is used to update parts of the HTML, or retrieve extra data from the server. Obviously, this requires a standards-compliant browser: in practice, there are only 2 such browsers.

Now, most people are stuck at this level: Writing AJAX code, possibly with the help of some clever IDE, and some code templates for doing standard things. At the server side, many approaches exist to coding the server part of the application: One can use PHP, Java, .NET (AJAX enabled ASP) and so forth. All these have 2 things in common: they clearly separate the server part and the client (JavaScript) part, and the programmer is very much aware of this.

This is all fine and dandy, and works well. It's a definite improvement over what existed before. However, it's still too limited. Any desktop application programmer will stare in horror at what needs to be done to produce an application that more or less behaves and responds like a desktop application. One needs knowledge of XML, HTML, Javascript, CSS and so on: the classical ingredients for a web application.

Morfik takes it one level further, in 2 steps:

First, the web-application is programmed like one would program any other client/server desktop application. The programmer programs as if there was only a desktop application, communicating with a server application via webservices. The fact that the code will run in a browser is completely hidden from the programmer (well, almost).

Secondly, it is principally possible to program in any high-level language: currently Object Pascal, Visual Basic, Java and C# are supported. The morfik IDE compiles this to a server application (a standalone HTTP server or an apache module) and client-side JavaScript, which is sent to the browser by the server.

This means no knowledge is needed of Javascript, XML, AJAX, HTML, CSS, which is a definite plus: any Delphi or VB programmer can start programming right away.

The end result is something which can be deployed as a real web application (i.e. integrated in an existing website) and which can be deployed as a regular desktop application: the

application will behave in exactly the same way. The desktop application is simply the server application which will simply spawn the web-browser and point it to itself, and will then proceed to behave as it's 'web' cousin.

The next sections show how this is achieved in practice.

2 Installing Morfik

Currently, the Morfik IDE runs only on Windows. A free trial download is available from the morfik website:

`http://www.morfik.com/`

(the website is a morfik webapp itself) and should install smoothly on any Windows platform, with a single caveat: If Firebird or Interbase are already installed, this may conflict with Morfik: Morfik uses embedded Firebird (2.0) as a default back-end for all its web applications. The installation will conflict with an existing Firebird installation, requiring some manual tweaking of the firebird installation that comes with Morfik.

Should this be the case, the Morfik website contains some help on how to handle this, all one needs to do is to search the forum. Mainly what needs to be done is to:

1. Copy a version of gds32.dll next to the morfik application binary.
2. Edit the firebird configuration file of Morfik so it will listen on another TCP/IP port as the standard installed firebird or interbase.

This will surely be fixed in the upcoming installs of Morfik.

3 The Morfik IDE

All development is done in the Morfik IDE, or AppBuilder, as it is called. The look of the IDE is loosely inspired on the MS-Acces look, although clearly influenced by other IDE's as well.

When started, the IDE looks as in figure 1 on page 3, granting immediate access to recently opened projects, and some examples from the morfik website. Clicking the 'New Project' menu item in the 'File' menu, will show the dialog in figure 2 on page 3. Depending on the chosen mode, more or less questions will be asked on how the project should be configured. All dialogs have in common that they first ask for a name and location where the project files should be stored.

After the wizard has finished it's job and has set up some files, the IDE main window will look something like in figure 3 on page 4. It shows a window with all the Project items in it. By default, this is only a form called 'Index' (most webservers use 'index.html' as the default page to be shown).

At the left, a set of icons can be seen which represents the various types of items (called 'Documents') that make up the application. They are the following:

Tables Any morfik application is connected to a database. Morfik is database centric, and it uses the database also to store things in it. By default, it uses embedded firebird as a database back-end. The tables can be designed visually in the IDE, much as in MS-Access, or OpenOffice Base.

The tables can be filled with data right at once in the Morfik IDE.

Figure 1: Morfik when started

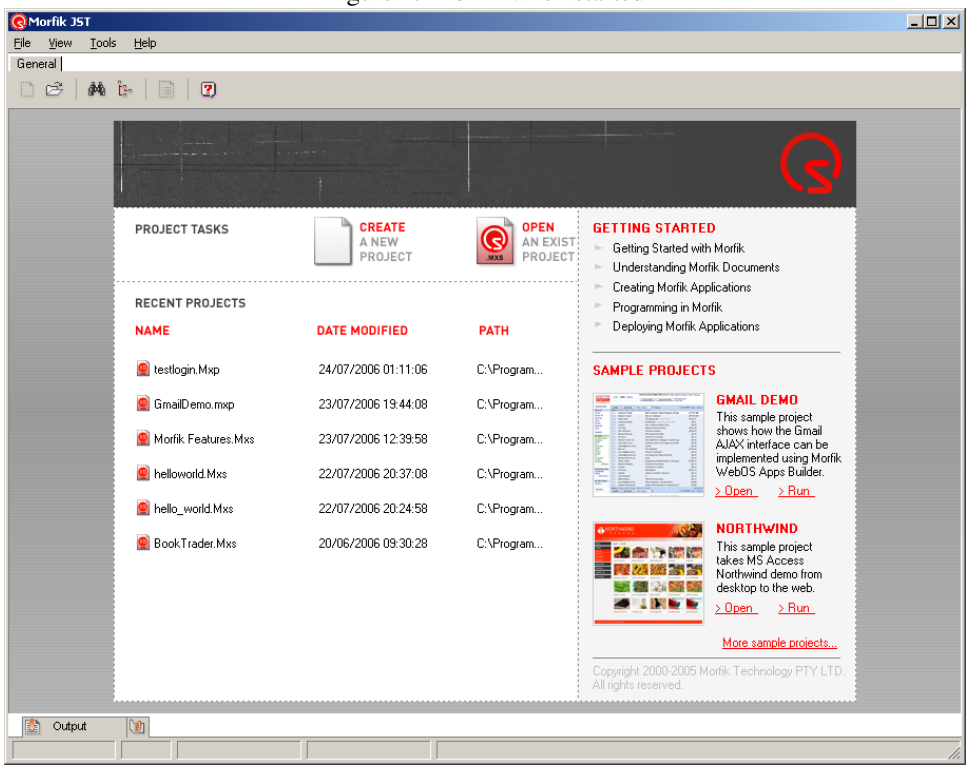


Figure 2: The 'New Project' wizard

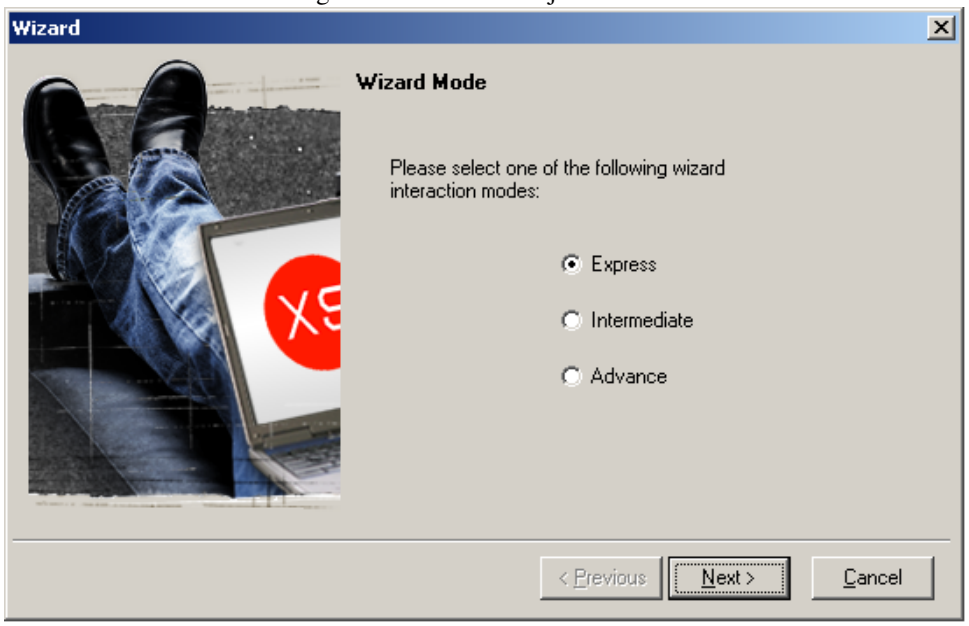
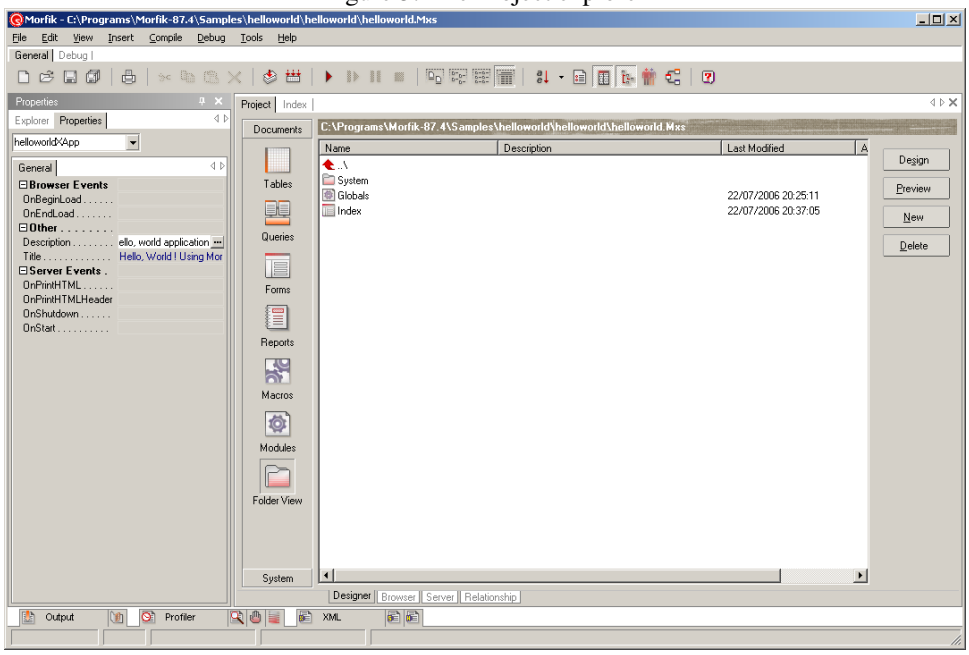


Figure 3: The Project explorer



Queries Queries are pre-defined queries just like they can exist in MS-Access, with the difference that the only possible queries in Morfik are select queries. There is a Visual Query Editor which allows to design the queries in a simple and straightforward way. Anyone familiar with MS-Access, or OpenOffice base will have no trouble understanding the query editor. The queries can have parameters, just like in Delphi. For the query to work in the designer, the parameters should be given default values.

Like with the table editor, the result of the query can be displayed and viewed.

Forms Forms are much like Delphi forms or forms in Visual Basic. The major difference is that they will be displayed in a browser. The design process is similar to that in Delphi or Visual basic: a simple drag and drop interface, and an object inspector to set various properties.

In terms of Delphi, Morfik's forms hold the middle between a `TForm` and a `TFrame`: A form can be displayed by itself, completely filling the browser window, but can also be displayed as part of another form, in which case Morfik speaks of a 'SubForm'.

A form can be made data-aware by connecting it to a query or table; In this case it can be used to display data from the query or table.

Reports As a database centric application builder, Morfik has a complete report-designer built in. The reports are output and shown in the browser as high-quality PDF documents, ready for saving or direct printing. The reports are designed much as the forms are designed, and work pretty much like any banding report designer.

Macros Macros are the term the Morfik IDE uses for WebServices. They are a simple RPC mechanism: this way the client (i.e. the browser) can execute code on the server. Because all communication happens via the HTTP protocol, it is only natural that the webservices are implemented using the SOAP protocol, which means that they should be accessible from any application which supports SOAP. In Morfik, the webservice is implemented as a class which exposes the input/output parameters as

variables; all that needs to be done is to implement the 'Execute' procedure, and the webservice is ready. For the client side of the morfik application, a class with a single method 'HandleResponse' must be implemented.

Modules Modules are simple files with code in them, they can be used on the client or on the server, or on both - this is a choice to be made when a new module is started. They can be compared with units in Delphi, or simple basic files in VB.

Clicking each of the items will display the available items of that type in the list view.

4 A primer: 'Hello, World!'

To show how it is done, the ubiquitous 'Hello, World!' application will be created in Morfik: the simplest of applications. After choosing the 'New Project' menu item, the wizard will ask which wizard interface should be used. Choosing 'Express' will ask the least questions: It will ask for the type of application and the name. There are various types of application:

WebOS Application This is a 'normal' Morfik web application. Morfik will create a database for you, add the embedded database engine and so on.

Classic Web Application This is a web application that does not use the internal embedded database engine, but connects to an external database.

AppExchange S-Control This is for creating simple modules which can be re-used in other projects. It can be compared to Delphi packages or OCX-es for Visual Basic; in .NET this would correspond to assemblies.

For the 'Hello World!' application, a 'WebOS application' is suitable. After giving it an appropriate name, the IDE will present the view as in figure 3 on page 4. The index form is by default the main form of a Morfik applicaton (although this can be changed in the project options) So we double-click on it and it opens in the designer. Pressing the F11 key will pop up the 'Properties' pane (the Object Inspector in Delphi-speak) which can be pinned down. From here on, it's like designing a visual application: Dropping a TextLabel from the toolbox on the form, and setting some properties: Caption, Color, font size. Hitting the 'Preview' button will compile and run the application.

Obviously, not only textlabels can be dropped on a form. By default, most common HTML elements can be dropped on a form, as well as some others.

Edits An edit is either an single line Edit (plain Input) or a multiline edit (textarea). Common for Visual Basic programmers, confusing for Delphi programmers, where an edit and a memo are different beasts entirely. Additionally, there are date/time edits and combo-box edits, and a file-upload.

Buttons These include normal buttons, check buttons and radio buttons.

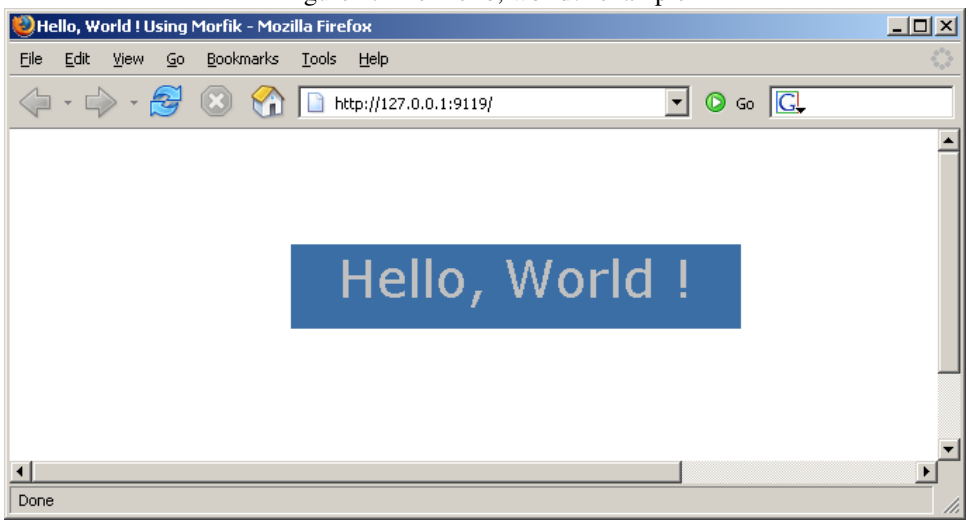
Images plain images or flash files can be included. A special paint box can be used to draw custom images on the web-page.

Others There are also tab-page controls, containers and subforms, even a timer.

There exist a lot of AJAX gui elements out there; Morfik is able to use these with the help of some specialized import routines. Examples of this can be found on the morfik Website, where several GUI component packs can be downloaded.

The application starts the browser, and the result is shown in figure 4 on page 6.

Figure 4: The 'hello, world!' example



5 Using a Database

Obviously, a 'Hello world' example can be made with small effort in most IDE's and programming languages. It is more interesting to program something with a database behind it, as that is what most real-life web applications are all about. To illustrate the concepts, a small login-like application will be developed.

So, a new project is started (testlogin) and in the 'Tables' view, a new table is created, with some fields in it:

U_ID a unique userid, an autonumber field.

U_LOGIN the login name of the user.

U_PASSWORD the password of the user.

U_FULLNAME full name of the user.

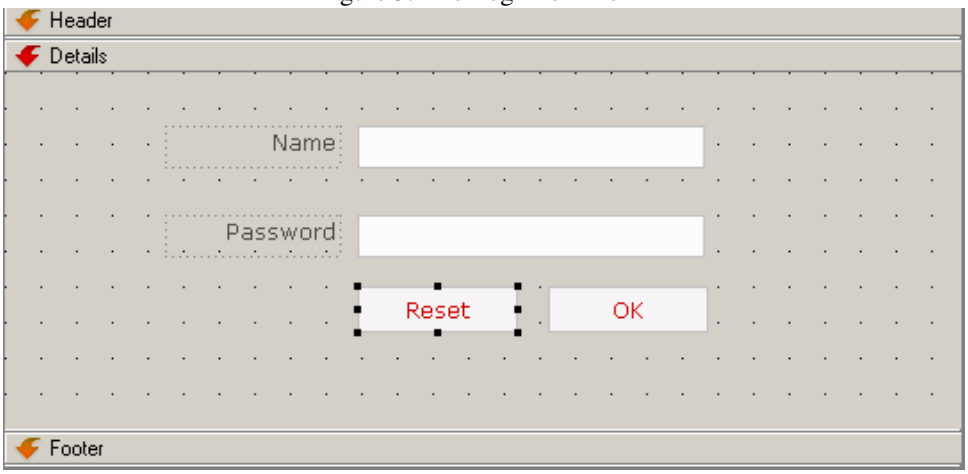
These fields can be defined in the table editor just like one would define them in OpenOffice base, or MS-Access, the interface is very similar. The primary key of the table is set to the `U_ID`, field, and a unique index is created on the `U_LOGIN` field. The table is renamed to 'USERS'. The `Preview` button shows the table in a tabular format, and data can be entered at once.

Once the table is defined, a query can be defined to authenticate a user. The query will be named `QAuthenticate` and looks as follows:

```
SELECT
     U_ID,
     U_FULLNAME
FROM
     USERS
WHERE
     (U_LOGIN = :Login) AND (U_PASSWORD = :PWD)
```

The syntax of the query is obvious, and any Delphi programmer will see at once that `Login` and `Password` are parameters for this query. The Morfik IDE also detects this,

Figure 5: The LoginForm form



and prompts you to enter default values for the parameters. To be able to work with the query in the IDE, it's mandatory to enter some values.

6 Designing a login form: Subforms

Now that the database backend is ready for a login, the application itself must be designed. In designing the main form, an important feature - indeed, a key aspect - of the Morfik way of designing web applications is demonstrated: a form can be placed on another form. The location where a form can be placed on a parent form is set at design time: On the parent form, a 'SubForm' control is dropped, which acts as a placeholder for the subform. A default form can be set in the designer. At runtime, the morfik application will display the parent form, and place the second form in the designated placeholder.

Keeping this in mind, the login application will be split in 3 forms:

index This is the main form of the application, and has a single 'Subform' control on it.

loginform This form contains the controls needed for login. It is the default form for the 'SubForm' control on the main form.

MenuForm this form will be displayed in the subform as soon as the user is logged in. It contains a greeting message, and can be enhanced with a menu, or whatever should happen after the user has logged in.

The index form is rather simple, it just contains a 'subform' component, which the name 'MainForm'. The property 'Form' can be set to the name of the form that should by default be created; It has a dropdown list with the names of defined forms in it. For this example, it should be set to `LoginForm`. That's it; nothing else is needed for the main form of our application.

The login form is slightly more populated than the login form; It should look something like figure 5 on page 7. The names of the buttons and edits are not really relevant. The OnClick handler of the 'Reset' button can be coded very easily:

```
Procedure LoginForm.Button2Click(Event: TDOMEvent);
Begin
  EName.Text:='';
```

```
  EPassword.Text:='';
End;
```

Which is very similar to what one would do in Delphi. Note that the 'Sender' parameter which is customary in Delphi is replaced by a 'Event' parameter of type 'TDomEvent'.

When the OK button is pressed, the form should attempt to authenticate the user and then display the main menu form. To do this, the server must be queried, the database needs to be consulted. To do this, a webservice is needed.

7 Authentication: A webservice

To be able to authenticate a user, the database must be consulted. Obviously, this is not possible from within the browser: Only the server part of the web applicaton has access to the database. So, the browser must query the server and ask it to authenticate a user. This is done by a implementing a webservice.

A webservice - or Macro, as it is called in the Morfik IDE is easy to implement: In the 'Macros' view, a new macro can be created using a wizard. The Morfik IDE will ask for a name of the macro, and then offers the possibility of specifing the parameters for the macro.

Based on what has been entered, the Morfik IDE creates an empty Macro skeleton based on the parameters that were specified. A macro (or webservice) consists of a client-side class, and a server-side class. In both, all parameters to the webservice are defined as fields of the class. In both classes, one method must be implemented to make the webservice functional.

For the login form, we need a webservice that receives 2 input parameters: a login name and a password. It should return a user ID, and the user's full name. If the login name and password were incorrect, -1 will be returned as the user ID, which acts as the default value of the parameter.

As can be seen from the code below (generated by Morfik) the server class needs a single method, called Execute:

```
WSAuthenticate=Class(Macro)
  Login    : String;
  Password : String;
  UserID   : Integer;
  FullName : String;
Public
  Procedure Execute; override;
End;
```

The `Login` and `Password` variables are the input parameters of the webservice. The output parameters are `UserID` and `FullName`. Currently, Morfik does no checking whether the code writes to input variables or not, so it's possible to abuse these variables. But they will not be transmitted back.

So, for the login example, the code could look as follows:

```
Procedure WSAuthenticate.Execute;
Var
  RecordSet  : TRecordSet;

Begin
  UserID:=-1;
```

```
  FullName:='';
  RecordSet := SoapServer.CreateRecordSet('QAuthenticate');
  RecordSet.Prepare;
  RecordSet.ParamByName(':Login').AsString:=Login;
  RecordSet.ParamByName(':PWD').AsString:=Password;
  RecordSet.Active:=True;
  If Not RecordSet.EOF then
    begin
    UserID:=RecordSet.FieldByName('U_ID').AsInteger;
    FullName:=RecordSet.FieldByName('U_FULLNAME').AsString;
    end;
  RecordSet.Active := False;
  SoapServer.DestroyrecordSet(RecordSet);
End;
```

Let us examine the code a bit closer: After initializing the output variables, the routine creates a recordset, which is the equivalent of a `TDataset` in Delphi. The `SoapServer` object is a global server object, responsible for handling soap requests. The `CreateRecordSet` method accepts a single parameter, which can be the name of a predefined query or a complete SQL statement. In case of the example: the pre-defined `QAuthenticate` is used. Morfik will create a new recordset and will copy the SQL statement from `QAuthenticate`.

After preparing the statement, the 2 parameters should be filled in: Note that the colon is considered part of the parameter name. This is because Morfik follows the MS-Access convention that unknown identifiers in queries are treated as parameters, rather than the Delphi convention where parameters are indicated with a colon.

After activating the query and verifying that it is not empty (it will always return either a single record or no records at all), the full username and ID are retreived. Finally, the recordset is closed and freed again.

That's it. The server part of our webservice is finished.

The Morfik IDE has also generated the client part of the webservice:

```
WSAuthenticate=Class(Macro)
  Login    : String;
  Password : String;
  UserID   : Integer;
  FullName : String;
Public
  Procedure HandleResponse; override;
End;
```

The `HandleResponse` is automatically called by the Morfik generated code when the webservice has been executed on the server. The variables will be filled with the values that the webservice returned.

When the webservice has been executed, the client must take appropriate action. In case the username/password was wrong, a message should be displayed. In case the login was successful, then the menu form should be shown in the 'MainForm' subcomponent of the index form. This can be done using the following code:

```
Procedure WSAuthenticate.HandleResponse;

Var
  Params : String;
```

```
Begin
  If (UserID<0) then
    ShowMessage('Could not log you in. '+
                'Please check your username and password')
  else
    begin
    Params:='"UserID='+IntToStr(UserID)+'",';
    Params:=Params+'"FullName='+FullName+'"';
    OpenForm('MenuForm','index:MainForm',Params);
    end;
End;
```

The first part of this code is straightforward. The `ShowMessage` call exists in the client-side Morfik API. The real magic happens in the `OpenForm` call, which is declared as follows:

```
Procedure OpenForm(Const URL, Context, Params : String);
```

The parameters are pretty straightforward:

URL Determines what form to open: This can be a real URL, or the name of a Morfik form.

Context This is a string determining where in the browser the URL should be shown. This can be a string in the following format `form:subform`, but can also be a designated location such as `self:_` which would designate the current browser window, or `Blank` which opens a new browser window. More on this can be found in the Morfik IDE help, and the MenuForm contains some examples.

Parameters This is a series of parameters which will be passed to the form. This is a string of the form

```
"param1=value1","param2=value2"
```

Which can be used to pass parameters to the form which will be displayed. They will be explained in the following section.

Now that the client part of the webservice has been coded, the 'OnClick' event of the 'Ok' button in the login screen can be implemented: it should call the webservice. This is done as follows:

```
Procedure LoginForm.BOKClick(Event: TDOMEvent);

Var
  Params : String;

Begin
  Params:='"Login='+EName.Text+'",';
  Params:=Params+'"Password='+EPassword.Text+'"';
  XAPP.RunWebService('WSAuthenticate',Params,0);
End;
```

As can be seen, the code is very simple, and the `RunWebService` call, which calls the webservice, resembles the `OpenForm` Syntax. The first parameter is the name of the webservice to call on the server. The second is the list of parameters for the service, in the form

```
"param1=value1","param2=value2"
```

Just as for the `OpenForm` parameters. Not all parameters must be passed, if a parameter value is not specified, the default value will be used.

The call will return immediatly: this is due to the nature of AJAX server calls: they are asynchronous. It is also for this reason that the client part of the webservice must be implemented as a separate method: this method will be called when the result of the server call is returned to the client.

This means also that some extra state checking on the client may be necessary in some cases where multiple calls to the same webservice are issued to the server.

8 The menu: Form parameters

A form in morfik can have parameters associated with them. If a query is used as the dataset for this form, the query parameters are immediatly available as form parameters as well. This is handy to create master-detail relationships, and for passing information to forms when they are opened. In the login example, this functionality was used to pass the user-id and full user name to the main menu form.

In the code of a form, the form parameters can be used as if they were normal form variables. In the login example, this can be used to display the user name in a nice greeting message when the MenuForm is shown. In the 'OnReady' handler of the form, the following code can be entered to display a nice greeting message:

```
Procedure MenuForm.WebFormReady(Var Ready: Boolean);
Begin
  LWelcome.Caption:='Welcome, '+FullName;
End;
```

The result can be seen in figure 6 on page 12 The other labels on the form can be clicked and will demonstrate the various OpenForm syntaxes.

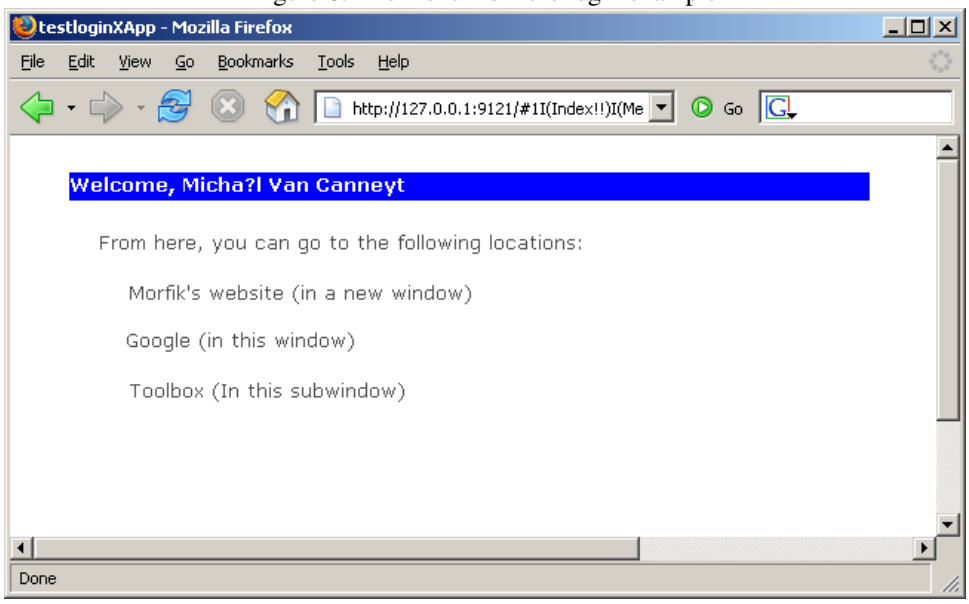
9 conclusion

It is known that AJAX-enabled ASP.NET applications can be made. Delphi offers technologies such as IntraWeb. Both suffer from the same thing: they are very server-centric. Client-side programming relies still on knowledge of JavaScript. Morfik RADically changes this by allowing to develop web-applications without any knowledge of HTML, SOAP, XML, JavaScript or AJAX: the Morfik framework takes care of all the gory low-level details. The programmer programs in his high-level language, as if he was developing a simple client-server system, with a fixed interface between client and server.

The Morfik IDE and framework are still on the drawing board, as many expected features are not yet fully functional. Nevertheless, the available test downloads will quickly convince any Desktop application developer that it is actually possible to create a web application which more or less reacts as a desktop application and which - far more important - can be designed and programmed as if it was a desktop application: in a RAD way.

Does this mean the end of the desktop application ? Far from it; Web applications suffer from a fundamental drawback: they are (by their very nature) executed inside the limited environment of the web-browser. This means that all the powerful API's from the desktop are not at the programmer's disposal; No DirectX, No Office-suite integration, no low-level

Figure 6: The menu from the login example



access to files, no shortcut keys, no control over closing the application: if the browser window is closed: bye-bye, web application. The list is long. The fact that web-applications do not work on all browsers is also a serious drawback (A morfik application does not run in Konqueror), endangering their widespread acceptance.

Nevertheless, a tool as Morfik pushes the limits of the web application development: As such, it is definitely showing the way in which web application development should and probably will evolve. So for applications that can be built within the boundaries of the browser: Morfik is definitely worth a deep investigation.