A Domain Model-Driven Approach for Producing User Interfaces to Multi-Platform Information Systems

Julien Stocq
Ethias
Avenue de l'Astronomie, 19
B-1210 Brussels, Belgium
Phone: +32-2/227 98 11 – Fax: +32-2/227 98 02
julien.stocq@ethias.be

Jean Vanderdonckt
Université catholique de Louvain (UCL)
School of Management (IAG)
B-1348 Louvain-la-Neuve, Belgium
Phone: +32-10/478525 – Fax: +32-10/478324
vanderdonckt@isys.ucl.ac.be

ABSTRACT

User interfaces to information systems can be considered systematic as they consist of two types of tasks performed on classes of a domain model: basic tasks performed on one class at a time (such as insert, delete, modify, sort, list, print) and complex tasks performed on parts or whole of one or several classes (e.g., tasks involving various attributes of different classes with constraints between and establishing relationships between). This paper presents how a wizard tool can produce user interfaces to such tasks according to a model-driven approach based on a domain model of the information system. This process consists of seven steps: database selection, data source selection, building the opening procedure, data source selection for control widgets, building the closing procedure, setting the size of the widgets, and laying them out. The wizard generates code for Visual Basic and eMbedded Visual Basic, thus enabling to obtain support for both stationary and mobile tasks simultaneously, while maintaining consistency.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *elicitation methods (e.g., rapid prototyping, interviews, JAD).* D.2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces.* H.2.4 [**Database Management**]: Systems – *transaction processing.* I.3.6 [**Computer Graphics**] Methodology and Techniques – *i*nteraction techniques.

General Terms: Design, Languages, Human Factors.

Keywords: Code generation, data base, information system, model-driven approach, multi-platform, RAD, wizard.

1. THE *WIZARD* APPLICATION

1.1 The Production Process

Since our goal is to generate a ready-to-run user interface (UI), the retained strategy consists of defining a set of specific rules for each platform to be supported. So, rather than using general models (as in [4,5]), WOLD (Wizard fOr Leveraging the Development of multi-platform user interfaces) uses specific ones. Right now, it produces UIs both for Visual Basic and eMbedded Visual Basic platforms. Although this approach compels us to define a different set of rules for each platform, we believe that it is the best solution to generate effective running UIs that are adapted to a particular platform. Designing UIs that manipulate database is a quite complex task and requires many capabilities. The *wizard* we have developed aims to provide the developer with task assistance by breaking this job into a sequence of subtasks and presenting her one step at a time (Fig. 1) [1]. In addition, the wizard exploits inference rules to propose at step *n* a solution that takes into account inputs that were given on step *n-1*. We have also paid attention to navigation's issues and our tool has been designed in a way that require the minimal amount of work by controlling if a step has to be proceeded or not. For instance, Step 4 in Fig. 1 will omitted if there is no need to work with `Combobox` or `ListBox` controls.

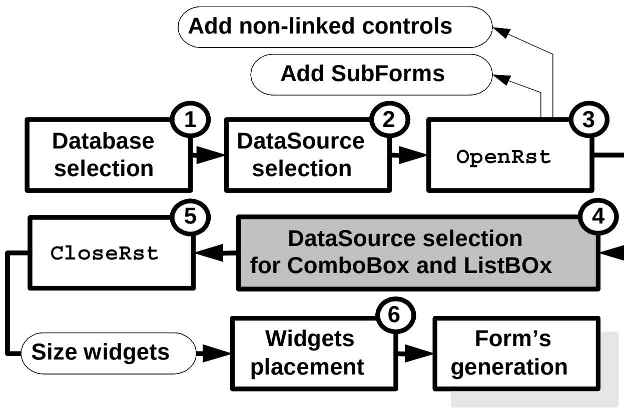


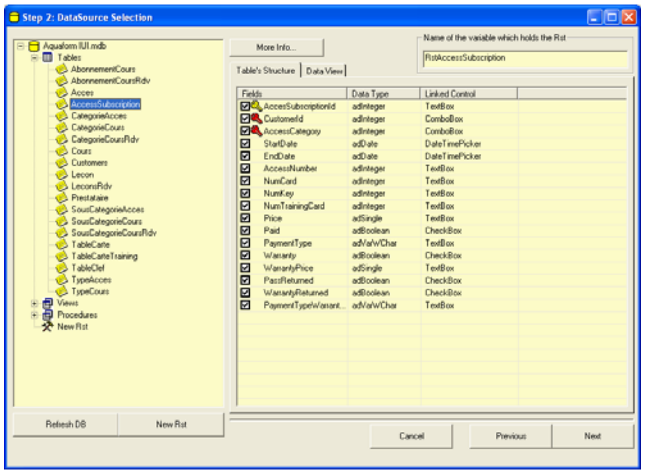
Fig. 1. Production steps of the Wizard

1.1.1 Step1: Database selection

The first step consists of selecting the database we want to interact with. It is also possible to specify which kind of UI has to be generated. Developers can choose between the list style and the datasheet style. The tool offers also the possibility to load a resource file that replaces the *wizard* in the context of a previously generated form. In this way, forms can be tailored to customer's needs trough incremental changes rather than with entirely new generations. This is in line with an iterative development model.

1.1.2 Step 2: Data source selection

The *wizard* gathers information from the database schema considered as a domain model and presents it to the user in a graphical way (Fig. 2). On the left side, the user chooses between four different data sources: the tables, the views, the procedures, and adding a new `RecordSet` used to create a new SQL query. Then, the *wizard* provides the user with information about its content in the right side of Fig. 2. There are listed all the fields that compose the data source, their data types and the classes of the controls by which they will be represented. Visual information will also point out if a field is primary key (with a yellow key icon) or foreign key (with a red key icon). The checkboxes enable the user to choose which field will be placed on the form.



Step 2: DataSource Selection

- Aquaform IUI.mdb
  - Tables
    - AbonnementCours
    - AbonnementCoursRdv
    - Acces
    - AccessSubscription
    - CategorieAcces
    - CategorieCours
    - CategorieCoursRdv
    - Cours
    - Customers
    - Lecon
    - LeconsRdv
    - Prestataire
    - SousCategorieAcces
    - SousCategorieCours
    - SousCategorieCoursRdv
    - TableCarte
    - TableCarteTraining
    - TableClef
    - TypeAcces
    - TypeCours
  - Views
  - Procedures
  - New Rst

Name of the variable which holds the Rst: RstAccessSubscription

Fields	Data Type	Linked Control
✓ AccesSubscriptionId	adInteger	TextBox
✓ CustomerId	adInteger	ComboBox
✓ AccessCategory	adInteger	ComboBox
✓ StartDate	adDate	DateTimePicker
✓ EndDate	adDate	DateTimePicker
✓ AccessNumber	adInteger	TextBox
✓ NumCard	adInteger	TextBox
✓ NumKey	adInteger	TextBox
✓ NumTrainingCard	adInteger	TextBox
✓ Price	adSingle	TextBox
✓ Paid	adBoolean	CheckBox
✓ PaymentType	adVarWChar	TextBox
✓ Warranty	adBoolean	CheckBox
✓ WarrantyPrice	adSingle	TextBox
✓ PassReturned	adBoolean	CheckBox
✓ WarrantyReturned	adBoolean	CheckBox
✓ PaymentTypeWarrant...	adVarWChar	TextBox

Fig. 2. Domain model contained in database structure

The *wizard* uses the a decision tree technique (Fig. 3) to select which control will present each field in the UI. In addition to these four widgets, the user can also work with the `Option Button`, `ListBox` and `MonthView` controls.

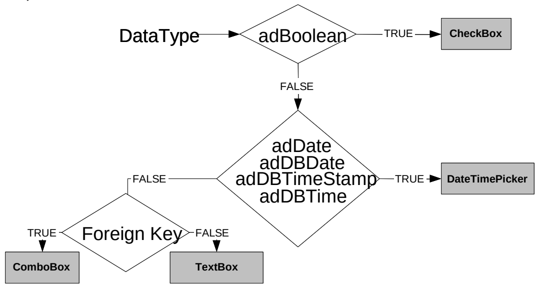
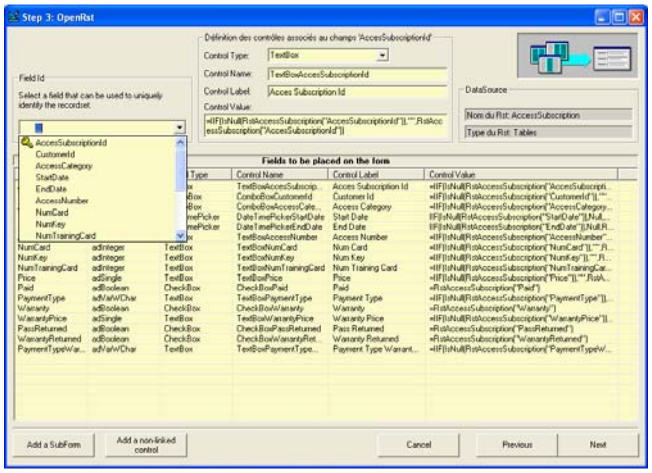


Fig. 3. Decision tree for selecting controls

1.1.3 Step 3: Building the `OpenRst` *procedure*

The third step determines the properties of the controls that will be placed on the form. These include the class of the control, its name, its label and its value when the form is opened. We have already seen how the class is defined. The *wizard* gets the name by simply concatenate the class of the control and the name of the field it represents. The label is simply the name of the underlying field. However it is automatically split in distinct words when it contains upper cases. The value is the database underlying field.



Step 3: OpenRst

Field Id

Select a field that can be used to uniquely identify the recordset.

- AccesSubscriptionId
- CustomerId
- AccessCategory
- StartDate
- EndDate
- AccessNumber
- NumCard
- NumKey
- NumTrainingCard

Définition des contrôles associés au champs 'AccesSubscriptionId'

Control Type: TextBox

Control Name: TextBoxAccesSubscriptionId

Control Label: Acces Subscription Id

Control Value: =IIF(IsNull(RstAccessSubscription('AccesSubscriptionId')),"",RstAccessSubscription("AccesSubscriptionId"))

DataSource

Nom du Rst: AccessSubcription

Type du Rst: Tables

Fields to be placed on the form

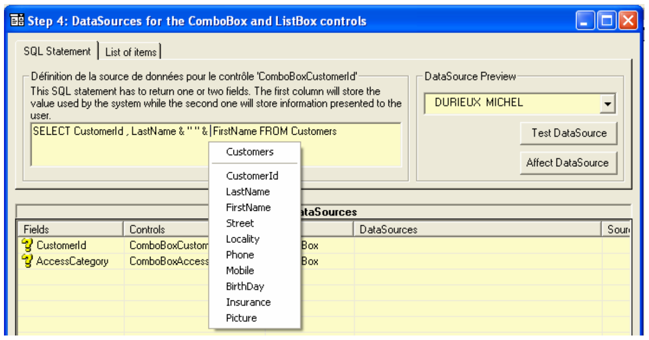
		l Type	Control Name	Control Label	Control Value
		x	TextBoxAccesSubscrip...	Acces Subscription Id	=IIF(IsNull(RstAccessSubscription('AccesSubscripti...
		Box	ComboBoxCustomerId	Customer Id	=IIF(IsNull(RstAccessSubscription('CustomerId')),""...
		Box	ComboBoxAccessCate...	Access Category	=IIF(IsNull(RstAccessSubscription('AccessCategory...
		mePicker	DateTimePickerStartDate	Start Date	IIF(IsNull(RstAccessSubscription("StartDate")),Null,...
		mePicker	DateTimePickerEndDate	End Date	IIF(IsNull(RstAccessSubscription("EndDate")),Null,R...
		x	TextBoxAccessNumber	Access Number	=IIF(IsNull(RstAccessSubscription('AccessNumber"...
NumCard	adInteger	TextBox	TextBoxNumCard	Num Card	=IIF(IsNull(RstAccessSubscription('NumCard')),"",R...
NumKey	adInteger	TextBox	TextBoxNumKey	Num Key	=IIF(IsNull(RstAccessSubscription('NumKey')),"",R...
NumTrainingCard	adInteger	TextBox	TextBoxNumTrainingCard	Num Training Card	=IIF(IsNull(RstAccessSubscription('NumTrainingCar...
Price	adSingle	TextBox	TextBoxPrice	Price	=IIF(IsNull(RstAccessSubscription('Price')),"",RstA...
Paid	adBoolean	CheckBox	CheckBoxPaid	Paid	=RstAccessSubscription('Paid')
PaymentType	adVarWChar	TextBox	TextBoxPaymentType	Payment Type	=IIF(IsNull(RstAccessSubscription('PaymentType')),...
Warranty	adBoolean	CheckBox	CheckBoxWarranty	Warranty	=RstAccessSubscription('Warranty')
WarrantyPrice	adSingle	TextBox	TextBoxWarrantyPrice	Warranty Price	=IIF(IsNull(RstAccessSubscription('WarrantyPrice')),...
PassReturned	adBoolean	CheckBox	CheckBoxPassReturned	Pass Returned	=RstAccessSubscription('PassReturned')
WarrantyReturned	adBoolean	CheckBox	CheckBoxWarrantyRet...	Warranty Returned	=RstAccessSubscription('WarrantyReturned')
PaymentTypeWar...	adVarWChar	TextBox	TextBoxPaymentType...	Payment Type Warrant...	=IIF(IsNull(RstAccessSubscription('PaymentTypeW...

Fig. 4. Incorporation of the procedure

To populate the list showed in Fig. 4 the *wizard* loops through all the fields the user has selected during the second step and applies the set of rules we have described. During this step the user designates which field uniquely identifies the record. This field will be used during the both `OpenRst` and `CloseRst` procedures to specify on which field has to be applied the `Id` passed as argument. Finally, the user has the opportunity to add a subform and non-linked controls to the UI.

1.1.4 Step 4: Datasource selection for `ComboBox` *and* `ListBox`

When the *wizard* detects whether a field is foreign key, the decision tree depicted in Fig. 3 will automatically assign a `ComboBox` to represent this field. The purpose of the fourth step is to define by which elements `ComboBox` and `ListBox` will be populated at run time. Naturally this step will be ignored if no control of these classes has to be placed on the form.



Step 4: DataSources for the ComboBox and ListBox controls

Définition de la source de données pour le contrôle 'ComboBoxCustomerId'

This SQL statement has to return one or two fields. The first column will store the value used by the system while the second one will store information presented to the user.

```
SELECT CustomerId , LastName & " " & FirstName FROM Customers
```

DataSource Preview: DURIEUX MICHEL

Fields	Controls		DataSources	Sour
CustomerId	ComboBoxCustom	Box		
AccessCategory	ComboBoxAccess	Box		

Fig. 5. Automatic assignment of widgets to queries

Fig. 5 illustrates how this step works. First the list is automatically populated with all the controls whose classes are `ComboBox` or `ListBox` types. Second when the user selects an item, the *wizard* provides him with the SQL statement that will populate the control at run time with accurate values. For this purpose, the *wizard* analyses the database's schema. For instance, for the `ComboBox IdClient` the *wizard* will look at the table `AccessSubsription` and then sees that the values should come from the table `Customers` (Fig. 6). The *wizard* will then propose the following SQL statement: `Select CustomerId FROM Customers`

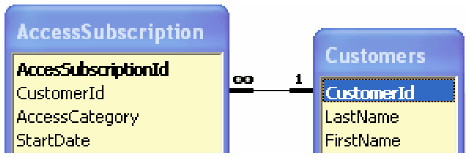


Fig. 6. Table definition in MS Access

Since this string returns the customer's Id, which is not relevant for the final user, the developer can modify the provided statement in order to return more significant information. In general, while the system will still work with the customer's Id, the final user will better work with their names, such as in:

```
SELECT CustomerId, FirstName & " " & LastName
FROM Customers
```

The above instructions return a second column containing the customer's first and last names. To increase the predictability of generated results, WOLD is equipped with a system that enables the user to see immediately the result of her actions. Although this concept has been applied every time it was possible, this step shows well how the *wizard* provides the user with immediate feedback. In Fig. 5, the `ComboBox` placed on the right side shows the result of the SQL statement entered on the left side. We can also observe that some guidance [2] is given that refines the SQL statement trough a context menu showing all the fields from the underlying table. This prevents developers to remember the domain model all the time. The *wizard* also checks that the SQL statement does not return more than two columns. Another feature is the possibility to add an element in the `ComboBox` that will serve to add a new item. In Fig. 7, the "Add New" item is located on top of the selection list. At run time, adding an element which is not on the list (a new customer for example) is then allowed.

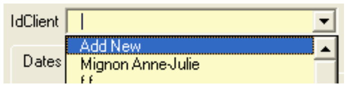


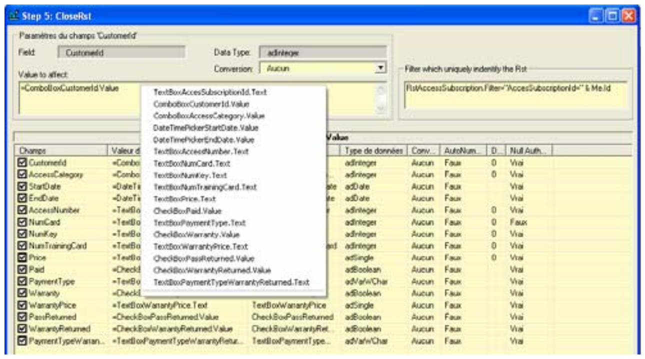
Fig. 7. Expandability of combo boxes

1.1.5 Step 5: Building the `CloseRst` procedure

As previously mentioned, the role of the `CloseRst` procedure is to update the database with modifications made by the end user. Depending on the `OpeningMode`, the procedure will update a specific record or add a new one. During this step, the *wizard* converts parameters entered into first sections of code. First, it draws up a list with all the fields of the `RecordSet`. Second, it assigns to each of them the value of their linked controls. The following listing shows one element of the built list.

```
RstAccessSubscription("CustomerId") =
ComboBoxCustomerId.Value
```

This is the syntax of the ActiveX Data Object. The left side contains the name of the field to be updated while the right side contains the value to be assigned to it (Fig. 8). The field `IdClient` will receive the value of the `ComboBox IdClient`.



Step 5: CloseRst

Paramètres du champs 'CustomerId'

Field: CustomerId

Data Type: adInteger

Conversion: Aucun

Value to affect: =ComboBoxCustomerId.Value

Filter which uniquely indentify the Rst: RstAccessSubscription.Filter="AccesSubscriptionId=" & Me.Id

TextBoxAccesSubscriptionId.Text
ComboBoxCustomerId.Value
ComboBoxAccessCategory.Value
DateTimePickerStartDate.Value
DateTimePickerEndDate.Value
TextBoxAccessNumber.Text
TextBoxNumCard.Text
TextBoxNumKey.Text
TextBoxNumTrainingCard.Text
TextBoxPrice.Text
CheckBoxPaid.Value
TextBoxPaymentType.Text
CheckBoxWarranty.Value
TextBoxWarrantyPrice.Text
CheckBoxPassReturned.Value
CheckBoxWarrantyReturned.Value
TextBoxPaymentTypeWarrantyReturned.Text

Champs	Valeur d	Value	Type de données	Conv...	AutoNum...	D...	Null Auth...
✓ CustomerId	=Combo		adInteger	Aucun	Faux	0	Vrai
✓ AccessCategory	=Combo		adInteger	Aucun	Faux	0	Vrai
✓ StartDate	=DateTi	ate	adDate	Aucun	Faux		Vrai
✓ EndDate	=DateTi	te	adDate	Aucun	Faux		Vrai
✓ AccessNumber	=TextBo	r	adInteger	Aucun	Faux	0	Vrai
✓ NumCard	=TextBo		adInteger	Aucun	Faux	0	Faux
✓ NumKey	=TextBo		adInteger	Aucun	Faux	0	Vrai
✓ NumTrainingCard	=TextBo	ard	adInteger	Aucun	Faux	0	Vrai
✓ Price	=TextBo		adSingle	Aucun	Faux	0	Vrai
✓ Paid	=CheckB		adBoolean	Aucun	Faux		Vrai
✓ PaymentType	=TextBo		adVarWChar	Aucun	Faux		Vrai
✓ Warranty	=CheckB		adBoolean	Aucun	Faux		Vrai
✓ WarrantyPrice	=TextBoxWarrantyPrice.Text	TextBoxWarrantyPrice	adSingle	Aucun	Faux		Vrai
✓ PassReturned	=CheckBoxPassReturned.Value	CheckBoxPassReturned	adBoolean	Aucun	Faux		Vrai
✓ WarrantyReturned	=CheckBoxWarrantyReturned.Value	CheckBoxWarrantyRet...	adBoolean	Aucun	Faux		Vrai
✓ PaymentTypeWarran...	=TextBoxPaymentTypeWarrantyRetur...	TextBoxPaymentType...	adVarWChar	Aucun	Faux		Vrai

Fig. 8. How to assign values to widgets at run-time

Since speaking about the 'value' can be misleading, we prefer to talk about the property's value that will be returned for each control. Indeed, for some controls, the property `.Value` should be used whereas for some other the property `.Text` should be used instead. To find out which property has to be used with a given control, the *wizard* looks in the decision tree of Fig. 9.

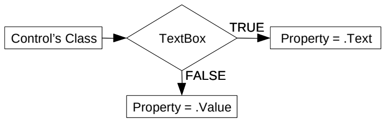


Fig. 9. Decision tree for property test

To foster mixed-initiative [2], the developer may then refine the proposals made by the *wizard* such as: adding conversion functions to some fields to ensure that they will match the require type of data. To assist the user in this task, the system provides for each field the following information: a name, the linked control, an assigned value, the data type, which conversion function is applied, whether it is auto numbered, a default value, and whether it accepts a null value. Finally, the user chooses through checkboxes which fields are updated when the form closes. This is particularly important because some fields if they are modified will raise errors. In any case, the *wizard* prevent the developer from modifying the field that serves to identify the record and those which are auto numbered.

1.1.6 Step 6: Setting the size of the widgets

By default, the size of each control is set to 200 pixels width and 20 pixels height. Nonetheless, the *wizard* queries the database's schema and adapts the width of controls whose fields' length is lower to 25 characters according to the following formula:

$$\boxed{W_{Ctrl} = Left_M + Len_{Field} + Right_M}$$

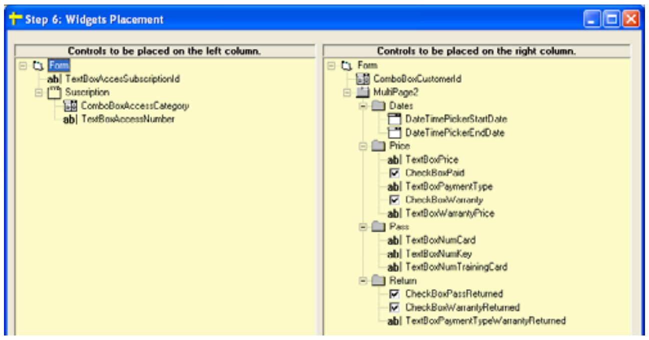
where:

- W_{Ctrl} stands for the control's width
- $Left_M$ and $Right_M$ are the left and right margin whose values are set to 2 pixels
- Len_{Field} is the length of the underlying field expressed in number of characters. One character equals 8 pixels.

By default all the controls and their respective labels are aligned on their middles except for the `ListBox` and the `MonthView` which are placed below their labels. As for the other steps the user can modify the default values and change the size of the controls as well as their alignment with their respective labels.

1.1.7 Step 7: Laying widgets out

Drawing the controls on the form is one of the most time consuming tasks during the design of a UI. So we have paid particular attention to that issue. The solution we have retained is to let the user choose the relative position of the controls. The *Wizard* computes the absolute ones and places it on the form. Fig. 10 shows how the user can organize the different controls by means of drag and drop operations. Icons are used to give visual clues about the type of the control to the developer. The user can distribute the controls amongst one or two columns and has also the possibility to add `Frame` and `MultiPages` controls to organize the different information on the form.



Step 6: Widgets Placement

Controls to be placed on the left column.

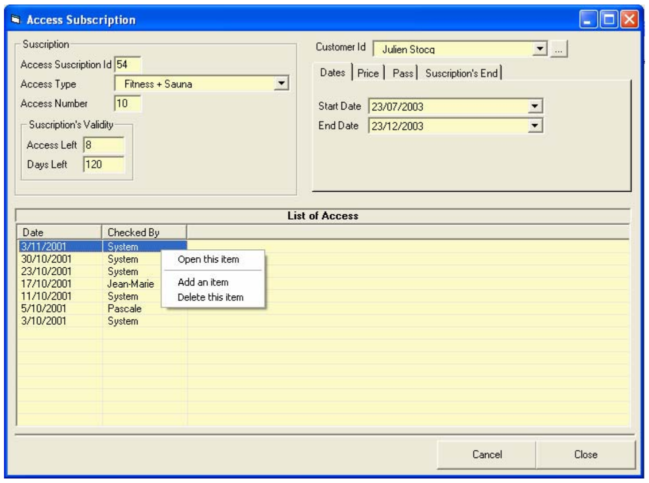
```
Form
  TextBoxAccesSubscriptionId
  Suscription
    ComboBoxAccessCategory
    TextBoxAccessNumber
```

Controls to be placed on the right column.

```
Form
  ComboBoxCustomerId
  MultiPage2
    Dates
      DateTimePickerStartDate
      DateTimePickerEndDate
    Price
      TextBoxPrice
      CheckBoxPaid
      TextBoxPaymentType
      CheckBoxWarranty
      TextBoxWarrantyPrice
    Pass
      TextBoxNumCard
      TextBoxNumKey
      TextBoxNumTrainingCard
    Return
      CheckBoxPassReturned
      CheckBoxWarrantyReturned
      TextBoxPaymentTypeWarrantyReturned
```

Fig. 10. Parameters for controlling the layout production

Once the user has determined the relative positions the *wizard* will proceed and generate the form represented in Fig. 11. When translating relative positions into absolute ones, the *wizard* respects a certain number of usability guidelines inspired from [3] which guarantee the form usability (Fig. 12). The *wizard* uses the notion of *area* to align and size the controls between themselves. An *area* is an portion of the form which presents a set of related information's to the final user. Inside an area labels are aligned between themselves as well as controls are.



Access Subscription

Suscription

Access Suscription Id	54
Access Type	Fitness + Sauna
Access Number	10

Suscription's Validity

Access Left	8
Days Left	120

Customer Id: Julien Stocq

Dates | Price | Pass | Suscription's End

Start Date	23/07/2003
End Date	23/12/2003

List of Access

Date	Checked By
3/11/2001	System
30/10/2001	System
23/10/2001	System
17/10/2001	Jean-Marie
11/10/2001	System
5/10/2001	Pascale
3/10/2001	System

Fig. 11. Generated UI for Visual Basic

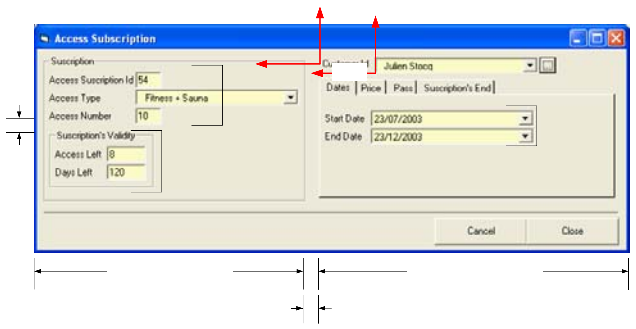


Fig. 12. Constraints satisfied by the layout algorithm

Furthermore, the widths of the various controls are balanced. Then, a second constraint consist of applying the same spacing between the different areas. And the third constraint is to align the different areas between themselves within the two columns. The *wizard* will draw the form so as to satisfy these constraints. The placement will start at the top left. Once all the controls will be placed the `Cancel` and `Close` buttons will also be added on the bottom right. Widgets can be laid out with and without automatic resizing as depicted in Fig. 13.

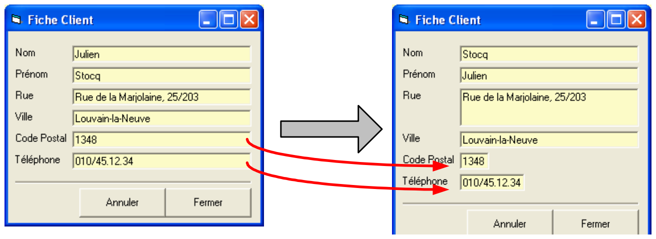
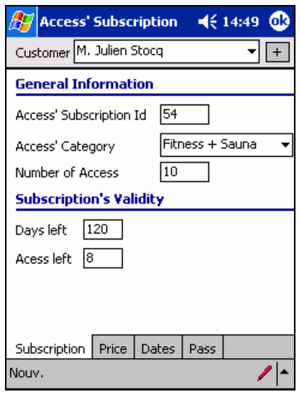


Fig. 13. Layout without and with automatic resize

1.1.8 Production for another Computing Platform

So far, we have seen how the *wizard* provides task assistance and speed up the realization of a running UI for Visual Basic. We present now the second one which is the generation, on the fly, of an UI targeted for the PocketPc platform. Fig. 14 shows an example of such a UI. When it comes to generate PocketPc UI, the *wizard* offers two main benefits: it lowers the effort needed to build the UI since information that where given during the six first steps can be reused. They consequently do not have to be entered a second time. Second, since the *wizard* uses a set of rules to translate the PC UI into the PocketPc one, it is possible to achieve more consistency amongst applications from the both platforms. If we refer to the Fig. 1, two more steps have to be proceeded before the *wizard* could generate the PocketPC interface. First, the user chooses which information should be placed on the form. Since the screen has more limited space, we usually restrict information on the hand-held device. Second, the *wizard* translates the PC UI into a PocketPC one. To achieve it, it analyses the form and determines how information can be presented. This analysis is done with the notion of *area* which has been introduced. Information belonging to the same *area* in the PC UI will be placed in a same area on the PocketPC UI. Since the PocketPC screen offers more limited space, the *wizard* will also check whether controls have room enough to be placed on the form. If this is not the case, i.e. the form is too small to receive all the controls, the *wizard* will come up with an alternative arrangement. Although we have not explored all the possibilities yet, an arrangement that has so far proven to be usable it the one depicted in the Fig. 14. As mentioned, information is arranged according to the different *areas* they belong to. When the amount of information items exceeds the screen resolution, the form is divided into different sheets. Within sheets information is no longer grouped into frame controls which consume a lot of space but rather by using blue labels and lines that emphasize elements and improve usability. End users seem also to be more comfortable when some key information is always displayed on the form so they know to what refer other elements trough which they navigate: in Fig. 14, the customer's Id is displayed in the gray area on the top of the form.



Access' Subscription 14:49 ok

Customer: M. Julien Stocq

General Information

Access' Subscription Id	54
Access' Category	Fitness + Sauna
Number of Access	10

Subscription's Validity

Days left	120
Acess left	8

Fig. 14. Generated PocketPC UI

ACKNOWLEDGEMENTS

The authors would like to thank Jean-François Hemmerlin and Lionel Neyts for their collaboration in the WOLD project. This work has been supported by "Salamandre" & "Cameleon" research projects and through the SIMILAR network of excellence.

REFERENCES

1. Dryer, D.C., *Wizards, Guides, and Beyond: Rational and Empirical Methods for Selecting Optimal Intelligent User Interface Agents*, in Proc. of IUI'97, pp. 265-268.
2. Horvitz, E., *Principles of mixed-initiative user interfaces*, in Proc. of CHI'99, 159-166.
3. Kim, W.C., Foley, J.D., *Providing High-Level Support And Expert Assistance in the User Interface Presentation Design*, in Proc. of InterCHI'93, pp. 430-437.
4. Paternò, F., Santoro, C., *One Model, Many Interfaces*, in Proc. of CADUI'2002, pp. 143-154.
5. Pizano, A., Shirota, Y., Iizawa, A., *Automatic Generation of Graphical User Interfaces for Interactive Databases Applications*, in Proc. of CIKM'93, pp. 344-355.

X Label
X Ctrl
Y Label
Y Ctrl
Area 1
Area 2
Area 3
Spacing between two areas
Left Column Lenght
Right Column Lenght
Spacing between the two columns