Runtime Generation of a User Interface, Described in a Database

Plamen Paskalev, Ilka Serafimova

Abstract: *The paper considers a realization of an approach for defining GUI for WEB-based application using description, located in a data base. The concept of the UIDL, together with some of the most recent works in this direction, is overviewed. The selected approach, the challenges and appropriate technical solutions are discussed. The problems and the benefits are listed. The developed test application, built using the discussed approach is debated.*

Keywords: *GUI description, UIDL, Real time GUI generation, Computer Systems and Technologies*

INTRODUCTION

The user interface is an essential part of every application. The development of GUI has to answer many requirements – from reasonable and robust handling of the information shown or received to user-friendly, predictable and consistent design. The task of GUI creation, although well known, arises many questions for investigation. Especially in case of complex client-server oriented systems, with large amounts of data to be shown the GUI design is a challenge. The problems, connected to different types of hardware to be used as client machines, different resolutions and multi-language support are combined with problems of grouping the data in corresponding logical hierarchy, implementing the logic behind the dialog layer and controlling the input. The approach with developing a separate solution for every special case is not an option due to the enormous amount of work. To answer these problems, the concept of UIDL (User Interface Definition Language) has been introduced. With its foundations in User Interface Management Systems (UIMS), introduced in 1980s [1], the UIDL concept allows designers to describe the interactive behavior in a high-level form, which gives a higher level of abstraction over the input – output devices.

This article describes an approach, used for realization of the GUI of a WEB based application using description, located in the database. The reasons and the advantages of the approach are classified. The structure of the description, main problems and technical challenges in the realization are discussed.

INTERFACE GENERATION

The concept of a user interface management system (UIMS) allows designers to specify interaction in a high-level user interface description language (UIDL), which abstracts the communication details. This specification can be then automatically translated to user interface implementation either during the building of the executable program or via interpretation during the run time. The choice of a UIDL model and methods is a key ingredient in the design and implementation of an UIMS. The goal of user interface management systems was not only to simplify the development of user interfaces but also to promote consistency across applications as well as the separation of user interface code from the application logic [2]. However, the standardization of user interface elements, together with some problems in using UIMS, like isolating the designer from the control of the low-level details in the visualization, didn't allow the wide acceptance of this concept [1].

In the next years new devices and interaction techniques were introduced. Some of the challenges (different screen resolution, multi-platform applications, multi-language support, etc.) can be solved with the concept of UIDL.

The goals of implementing UIDLs are ([3]):

• To capture the requirements for a user interface as an abstract definition that remains stable across a variety of platforms.

• To enable the creation of a single UI design for multiple devices and platforms.

• To improve the reusability of a user interface.

• To support evolution, extensibility and adaptability of a user interface.

• To enable automated generation of user interface code.

Several UIDL solutions and frameworks have been developed in recent years – UIML [4], XIML [5], UsiXML[6], TERESA XML [7], Plastic User Interfaces [8]. As the names imply, most of them are XML-based. The solutions [10] using data base description are used to handle user preferences and permissions, which affect the GUI. These solutions are not universal but tightly task-dependent and are used in applications with many different users.

SOFTWARE SOLUTION

1. TASKS AND GOALS

The discussed approach is an attempt to simplify the work of the UI designer of complex applications and in the same time to add some additional degrees of freedom in the way, the application is handling its user interface during the runtime. Since the existing hierarchies can develop in the future and/or new ones can be described with their own structure, the solution should be able to maintain their GUI representation without recompiling the presentation layer. The application should be able to expand the possible control types with minimal effort. Using the UIDL concept, this approach is based on description of the GUI, placed in a database. Decision to use database as a container for the GUI description leads to the following advantages:

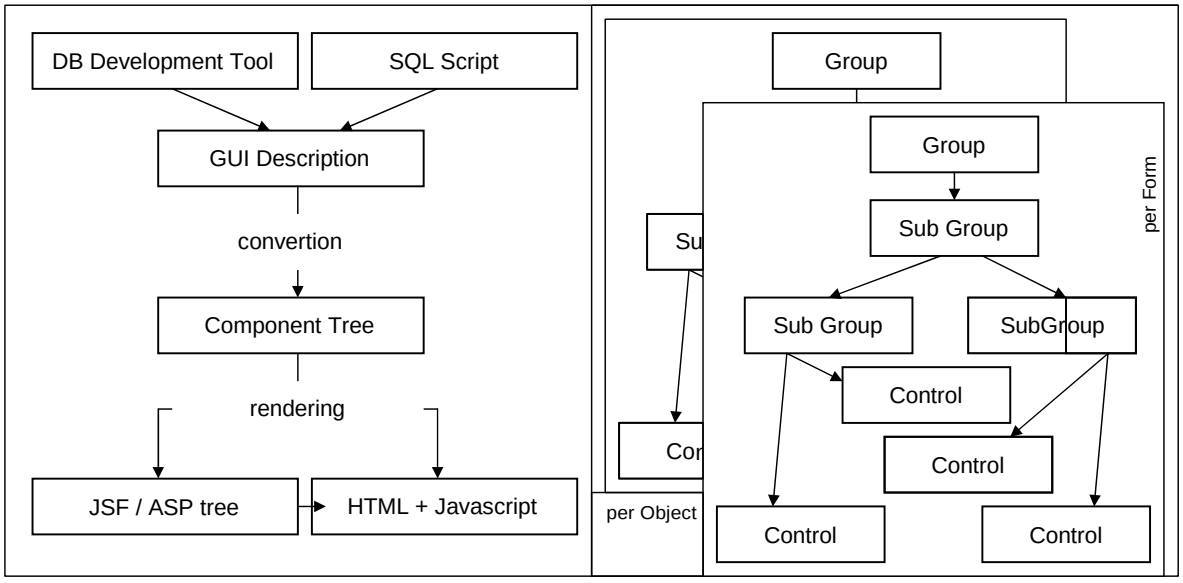


Figure 1. a) Generation process steps; b) The structure of description objects.

• The description is platform independent abstract and, in the same time, flexible enough to contain all the needed information about visualization [9].

• The parsing of the description is robust, fast (SQL statements) and gives the developers the opportunity to implement some additional steps in the business logic layer, implementing some aspects of intelligent user interfaces, etc. [9].

The main idea behind the discussed approach is to build the GUI not from separate controls, but from groups of logically connected controls, which then can be re-used.

Having defined basic sets of controls, grouped logically, the designer can then build quite complex GUI structure using them as building blocks, thus implementing some kind of *encapsulation* of the user interface objects. Special rules for excluding controls or sub-groups from groups for an instrument or a form are available. Thus, the approach realizes some kind of *polymorphism* too.

2. REALIZATION

The approach, discussed in this article consists of user interface description in the database, GUI builder and a renderer, realized in Java (fig. 1a). It is part of a WEB application server solution, designed to work with financial information (financial instruments, environments, etc.). The description is organized in components, which facilitates the reusing of groups or sub-groups of controls in visualization of different instruments. Although designed for visualization of financial information, the approach can be used for any area, where complex data is to be displayed, certain logic is to be realized and the model of data visualization allows reusing of components.

The GUI description is based on the following structure of objects (fig.1b):

- The control objects (leafs in the hierarchy) describe the normal GUI controls, used in WEB application: Edit boxes, Drop down lists, Check boxes, Labels, etc.
- The controls are grouped together in groups. The groups can be members of higher level groups, etc. The groups on the highest level represent tab controls in a common tab holder of a form.
- The form objects (defined with their identifiers) describe the views of the application.
- The instrument objects define the financial instrument types to be shown on the GUI. They are also identified by unique Ids and can be used for applying of different kinds of conditional displaying of groups, sub-groups or single controls.

Both controls and groups are determined with their identifier numbers. Additionally, the relations between the groups and their members (sub-groups and controls) are defined. The groups, belonging to the certain instrument type and groups, included in a certain form are defined as well.

The control and group descriptors contain information about their relative positions in X- and Y- directions. Information about validation and data binding as well as some visualization details (style, converting type, access type – read only etc.) are included. Complex controls like tables and charts can be maintained using this approach as well.

3. DB STRUCTURE

The discussed solution includes description of the GUI in several data tables in Oracle 10 database. The structure of the data tables and the relations between them are shown on fig. 2. The database solution consists of the following tables:

INSTRUMENT_TYPE – nomenclature table, which contains the definitions of all the instruments, the GUI is capable to handle. The instruments have Type / Subtype and are identified with unique identifier.

FORM_TYPE – nomenclature table, containing the definitions of the views, the application consists of. Every form is identified with an unique identifier.

GROUP_DATA – the table, containing the data for the groups in the hierarchy. It consists of identification information (unique identifier, group type) and visualization information (relative positioning in both directions, group style, label, etc.).

CONTROL_DATA - table, collecting the controls' definition. The table contains information about the type, relative position, identifier of the control, validating, type of the conversion of the input, access (read-only, read/write), etc.

GROUP_GROUP_REL – Relational data table, used to define the hierarchy of the groups.

INSTRUMENT_GROUP_REL, INSTRUMENT_CONTROL_REL_OPP – relational tables, giving additional constraints and rules for displaying or hiding groups and controls per instrument respectively.

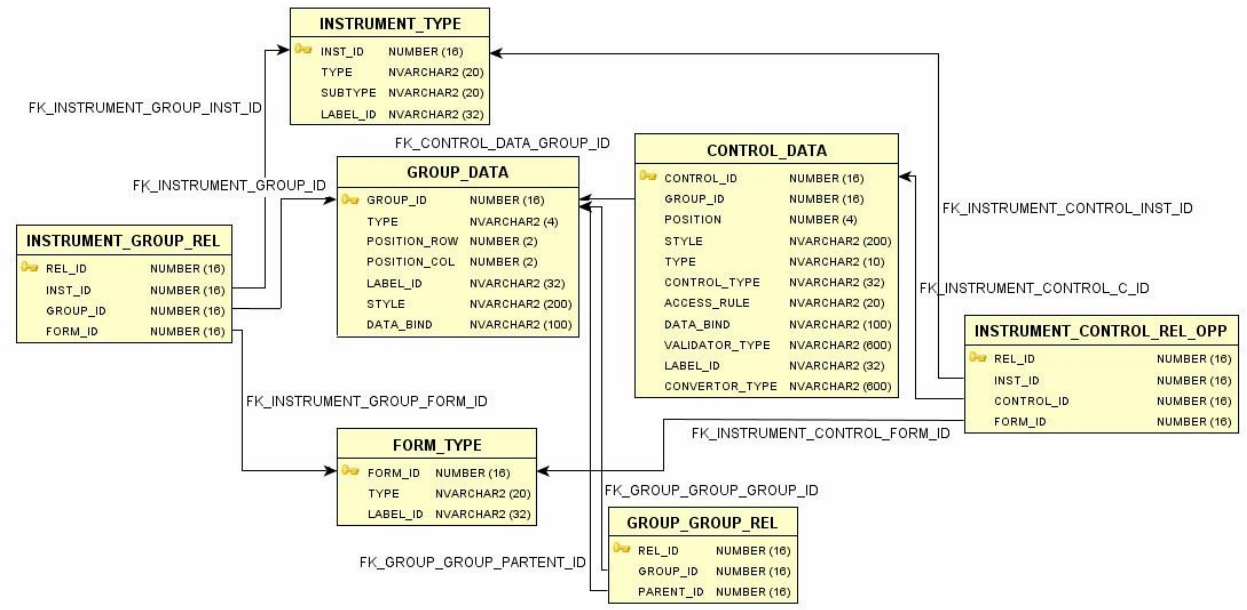


Figure 2. The DB tables and relations

4. RETRIEVING AND RENDERING THE GUI

After the database description is converted to a component tree the next step is to refactor it into a GUI. The builder which transforms the component tree can be implemented to generate different output: from plain HTML and Javascript to the flexible and sophisticated ASP.NET component tree or JSF component tree - any technology that can render an application GUI. The discussed solution is realized using JSF technology which is chosen for many reasons:

- Easy to run on any server supporting java container like Tomcat, Apache, Jboss, etc. (as opposed to ASP.NET which needs paid software to run);
- Powerful - supports special features like parameterized standard and custom converters and validators;
- View tier focused (compared to Struts for example) [11];
- Easy to maintain custom components [12].

The solution is implemented as a custom component (represented by a tag and a state manager) and a builder. This way the presentation and transformation logics are separated. The component simulates a Tab control rendering a certain set of data depending on the user selection or input.

The builder receives the component tree and converts it to a tree of standard JSF controls such as panel grids, command links, text inputs, etc. The generated tree has the following structure:

root represents the custom component in the standard JSF view tree;
  `-*1st level* children represent tab views;
    `- *2nd level* children represent layout tables;
      `-… nested layout tables or groups;
        `- *leaves* represent the actual data in labels/inputs/outputs/data tables.

The builder uses factories for the control tree generation. The factories know what type of component to create and which properties to set depending on the given source node. They can produce:

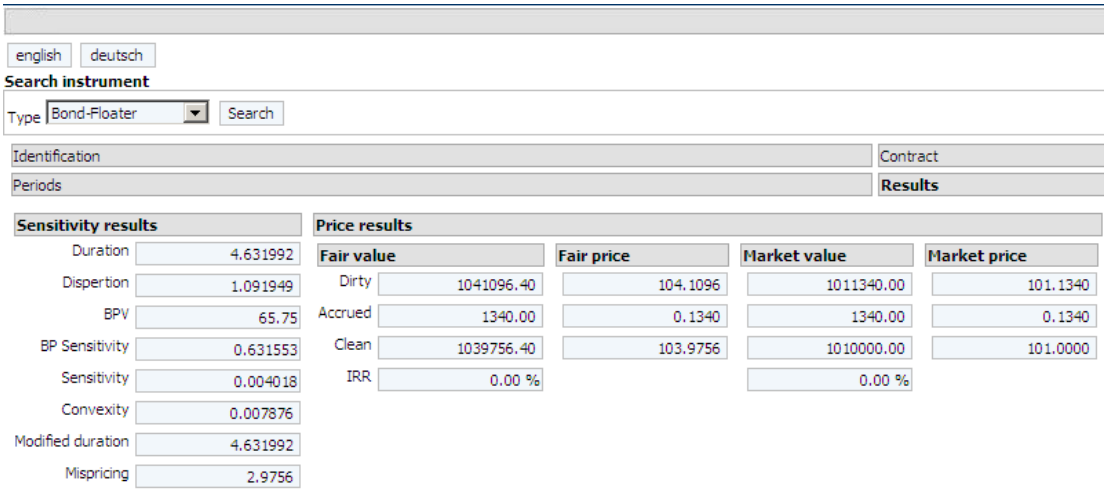


Figure 3. Generated tab control: "Results" tab selected

- Layout tables to arrange the elements on the page. The elements know their table coordinates – column and row number.
- Pairs of labels and inputs/selects for single value elements. They may have specified CSS style, converters and validators, collections for filling the select items, etc.
- Data tables for collection display. The data tables develop collections by columns (rows) and display their properties in rows (columns) – the database description allows table direction management.
- Action components which can invoke custom methods from the model.

The used factory pattern is flexible and easy to expand with new types of components and new properties.

This GUI generation approach adds only the necessary components to the view during the rendering phase, unlike the standard one. The latter is to declare all the possible components and/or combinations and show the proper ones depending on checks performed by the model. This burdens the model with unnecessary and inappropriate logic. The discussed solution simplifies the readability of the view.

5. ADVANTAGES OF THE SELECTED SOLUTION

The implemented GUI generation approach is very flexible. It allows easy change of the builder output to any GUI technology or another GUI description language (e.g. XIML). It allows simple addition of new controls and attributed with little effort. The building process saves the application from many checks for displaying/hiding a component which very much improves the readability of the view. The DB description changes immediately reflect on the generated HTML. Moreover, it allows building graph structures and reusing existing descriptions in many hierarchies unlike XML.

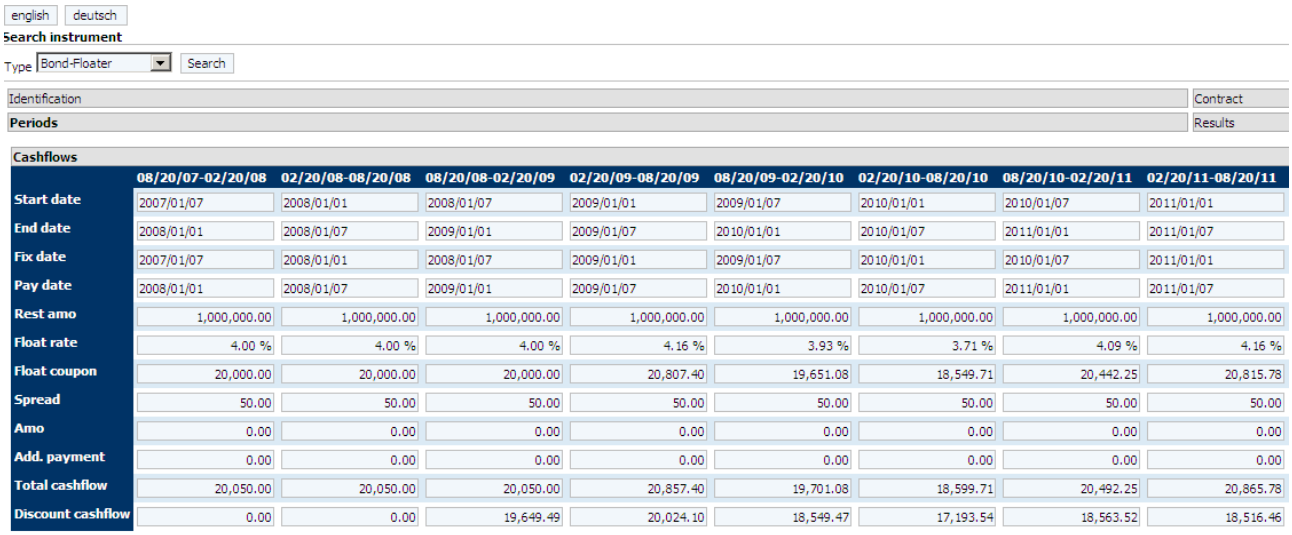
CONCLUSIONS AND FUTURE WORK

The discussed solution was developed and tested using sample data. Some results can be seen on fig. 3, fig. 4. The following conclusions were made:

- The development of a dynamic interface is more time consuming, compared to direct hard coding the GUI. However, this investment will pay itself back in later changes and adaptations of the system because of its higher flexibility.
- The database-oriented approach benefits from the easier maintenance and higher availability of the data center.
- It is fast, no delays were encountered due to the database interaction.
- It is more flexible and simple than using the standard show/hide controls approach;
- It allows reuse of groups of components with a single add. Changes in the group take place in all referencing hierarchies (one change per all hierarchies);
- Adding new controls types or properties to the description is easy but requires

refactoring and recompilation of the presentation layer;

• Database description does not allow definition of the used language like XML with DTD or XSD Schemas.


english deutsch

Search instrument

Type Bond-Floater Search

Identification Contract

Periods Results

Cashflows

	08/20/07-02/20/08	02/20/08-08/20/08	08/20/08-02/20/09	02/20/09-08/20/09	08/20/09-02/20/10	02/20/10-08/20/10	08/20/10-02/20/11	02/20/11-08/20/11
Start date	2007/01/07	2008/01/01	2008/01/07	2009/01/01	2009/01/07	2010/01/01	2010/01/07	2011/01/01
End date	2008/01/01	2008/01/07	2009/01/01	2009/01/07	2010/01/01	2010/01/07	2011/01/01	2011/01/07
Fix date	2007/01/07	2008/01/01	2008/01/07	2009/01/01	2009/01/07	2010/01/01	2010/01/07	2011/01/01
Pay date	2008/01/01	2008/01/07	2009/01/01	2009/01/07	2010/01/01	2010/01/07	2011/01/01	2011/01/07
Rest amo	1,000,000.00	1,000,000.00	1,000,000.00	1,000,000.00	1,000,000.00	1,000,000.00	1,000,000.00	1,000,000.00
Float rate	4.00 %	4.00 %	4.00 %	4.16 %	3.93 %	3.71 %	4.09 %	4.16 %
Float coupon	20,000.00	20,000.00	20,000.00	20,807.40	19,651.08	18,549.71	20,442.25	20,815.78
Spread	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00
Amo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Add. payment	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Total cashflow	20,050.00	20,050.00	20,050.00	20,857.40	19,701.08	18,599.71	20,492.25	20,865.78
Discount cashflow	0.00	0.00	19,649.49	20,024.10	18,549.47	17,193.54	18,563.52	18,516.46

Figure 4. Generated data table. The array of data is developed by columns

REFERENCES

[1] Myers, B., Hudson, S., Pausch R., Past, Present, and Future of User Interface Software Tools, ACM Transactions on Computer-Human Interaction, Vol. 7, No. 1, March 2000.

[2] Shaer, O. , Green, M., Jacob, R., Luyten, K., User Interface Description Languages for Next Generation User Interfaces, Workshop, CHI 2008 Proceedings, Florence, Italy, 2008

[3] Luyten, K., Abrams, M., Vanderdonckt, J., Limbourg, Q. Developing User Interfaces with XML: Advances on User Interface Description Languages, Advanced Visual Interfaces, Italy, 2004.

[4] Ali, M.F., Perez-Quinones, M.A., Abrams, M., Building Multi-Platform User Interfaces with UIML, Multiple User Interfaces, John Wiley and Sons, UK, 2004.

[5] Puerta, A. and Eisenstein, J. XIML: A Common Representation for Interaction Data, Sixth International Conference on Intelligent User Interfaces, 2002.

[6]Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, 9th IFIP Working Conf. on Engineering for Human-Computer Interaction, Hamburg, Germany, 2004.

[7]Mori, G., Paternò, F. and Santoro, C., Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, vol. 30, issue 8, 2004.

[8]Thevenin, D., Coutaz, J. and Calvary, G. A, Reference Framework for the Development of Plastic User Interfaces, Multiple User Interfaces, John Wiley & Sons, , UK, 2004

[9] Paskalev, P., Nikolov, V., Multi-platform, script-based user interface, In proceedings of International Conference on Computer Systems and Technologies CompSysTech'04 IIIB.14, 2004

[10] van de Vegte, J., Som de Cerff, W.J., A Dynamic GUI for accessing the Netherlands SCIAMACHY Data Center, Montreal, Canada, May 2000

[11] McClanahan, Cr., Struts Or JSF? Struts And JSF?, http://blogs.sun.com/craigmcc/date/20040927#struts_or_jsf_struts_and , September 2004.

[12] Geary, D., Horstmann, C. S., Core Javaserver Faces [Jsf], Prentice Hall PTR, June 2004

ABOUT THE AUTHORS

Eurorisk Systems Ltd.
31, General Kiselov Str.
9002 Varna, Bulgaria
Ilka Serafimova, E-mail: iserafimova at eurorisksystems dot com
Plamen Paskalev, E-mail: ppaskalev at eurorisksystems dot com