

Rule-Based Conversion of UsiXML-based GUI Description

Plamen Paskalev, Ilka Serafimova

Eurorisk Systems Ltd.

31, General Kiselov Str., 9002 Varna, Bulgaria

E-mail:

ppaskalev at eurorisksystems dot com

iserafimova at eurorisksystems dot com

ABSTRACT

The paper considers a realization of an approach for converting a UsiXML-based GUI definition to a proprietary format of a WEB-based application. The conversion is realized as a rule based solution using CLIPS engine. The advantages of the selected approach and the technical realisation are discussed. The results from implementing the solution are debated. The integration with an existing GUI handling solution, storing the definition in a database is discussed.

KEYWORDS

GUI description, UIDL, UsiXML, Rule based, CLIPS, 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. Implementation of the main obligatory parts of the GUI (such as showing or hiding controls or groups of controls, making fields read-only, etc.) leads to program code which is hard to read and support. GUI design is especially complicated in case of complex client-server oriented systems, where large amounts of data have to be shown. The programmer tasks were simplified with introduction of high level programming languages (e.g. Java, C# etc.) and technologies for web based application development (e.g. JSF, ASP.NET, etc.). Nevertheless, the design is still a challenge in case of complex applications. The problems like different types of hardware on client

machines, different resolutions and multi-language support are combined with problems of grouping the data in correspondent logical hierarchy, implementing the logic behind the dialog layer and controlling the input. The approach of development of 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 [15], the UIDL concept allows designers to describe the interactive behaviour in a high-level form, which gives a higher level of abstraction over the input – output devices.

Another aspect, which adds complication to the user interface handling, is inclusion of some aspects of Intelligent User Interfaces (IUI) in the application logic. The attempts to trace user behaviour, to determine the user expertise level, to maintain a knowledge base with most common user activities, action sequences and failures and, based on this knowledge, to undertake some dynamic changes in the GUI in order to help the user in his interaction with the application make the GUI design even more difficult and complex.

The basic theoretical matters and related works will be covered by the end of the introduction. The technologies used in the proposed solution will be discussed in the next two chapters. Further, the prototype tasks and goals will be defined. A detailed module description will follow. And finally, conclusions about the research will be made and future works will be stated.

Graphic user interface generation

The concept of user interface description language (UIDL) allows designers to abstract the description of the GUI and thus separate it from the application business logic. This abstraction gives the possibility to use the description in different manners. It can be converted during application release building to a static user interface implementation or it can be interpreted at runtime. The implementation of the

user interface management systems (UIMS) based on the used UIDL is crucial for the application design. The former lightens the designers work, but also allows easy GUI consistency support across the application [19]. However, problems like isolating the designer from the control of the low level details in the visualization and other UIMS usage problems, as well as the standardization of user interface components did not allow a wide acceptance of the UIDL concept [15].

In the next years the concept of UIDL solved some of the above-mentioned issues like different screen resolution, multi-platform applications, multi-language support with the introduction of new interaction technologies and devices.

The objectives of UIDL implementations are [11]:

- 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.

Related works

The usage of XML as a UIDL base is very widespread. There are numerous suggestions such as UIML [1], XIML [18], TERESA XML [14] etc. The solutions [7] using data base description are not universal but tightly task-dependant and are used in applications with many different users.

The usage of database description of GUI was discussed in [17]. A solution for generation of unified GUI generation, based on modular components is proposed in [6].

GUI DEFINITION USING USIXML

One of the UIDL solutions and frameworks developed in recent years is UsiXML [10]. It is an XML-based context-dependent UIDL which provides various levels of abstraction: context-independence, platform-independence, etc. UsiXML allows specifying of multiple models involved in user interface design such as: task, domain, presentation, dialog, and context of use, which is in turn decomposed into user, platform, and environment. These models are structured according to the four layers of the Cameleon framework [2]: task and concepts, abstract user interface (AUI), concrete user interface (CUI), and final user

interface (FUI). Intermodel mapping is used to support relationships between these models [8].

A FUI lies at the bottom of the Cameleon framework. It is platform dependent and is either interpreted (e.g., through a Web browser) or executed (e.g., after compilation of code in an interactive development environment). A CUI abstracts the UI definition as independent from any computing platform, but it is environment dependent. A CUI is also considered a reification of an AUI at the upper level and an abstraction of the FUI with respect to the platform. An AUI abstracts the UI definition as independent of any modality of interaction (e.g., graphical interaction, vocal interaction, speech synthesis, etc.). An AUI is considered as an abstraction of a CUI with respect to modality. At the top of the framework is the Task and Concepts level where the interactive task carried out by the end user is defined according to their viewpoint. Task and Concepts are considered class instances representing the concepts manipulated [9].

UsiXML is a preferred technology for the research of this article because it is an XML based UIDL with a wide range of possibilities and several GUI abstraction levels. It has a number of user-friendly and easy-to-use editors (such as GraphiXML[12] and Sketchi-XML[3]) thus covering one of the main problems of the previous solution, discussed in [16, 17].

For the purpose of this experiment CUI description is used since the solution realizes a graphic user interface which is environment dependent. The concrete JSF interpreter is used as a testing tool for the generated or converted GUI descriptions.

USER INTERFACE DYNAMIC MODIFICATION

Applications which interact with people allow the introduction of intelligent and dynamic behaviour which improves the communication between the former and the latter. The details of user input can be gathered and afterwards examined using intelligence engine. The user interface is then rearranged on the bases of that analysis. The enhancements concern:

- Dynamic control of complicated application views with large amount of heterogeneous data to be shown (e.g. health information systems, financial software, complex industry controlling software, etc.) to reduce the level of complexity for easier development and testing.
- Automatic construction of intuitive user interfaces according to the user experience to facilitate their interaction with the system and decrease the necessity of regular on-line help and application description usage.

- Adaptation of the user interface to changing user expertise level to provide detailed information depending on the user knowledge and goals.

An approach, used for the realization of the GUI of a WEB based application which realizes some aspects of IUI was presented in [16]. The main goal of that project was to create an environment, which would allow an easy extension, manipulation and dynamic modification of the GUI, based on the

- a) data to be displayed
- b) investigation of the behaviour of the user

(See Fig. 1.)

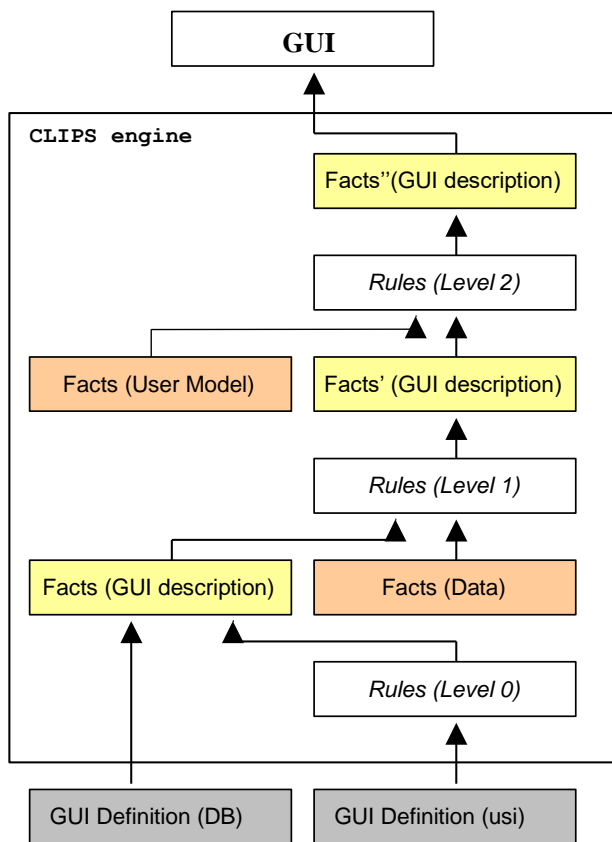


Figure 1: GUI Dynamic Modification project

The kernel of the project was realized as a rule-based solution (Fig. 1, [17]) based on a CLIPS [5] engine. The advantages of the CLIPS engine usage were discussed in [16, 17]. Several layers of rules were developed for this purpose. That prototype solution was based on GUI description, located in a database. Although the results from the tests with the application, built using that approach, were positive, it became clear, that some extensions are necessary. The problems which were identified during the tests can be grouped in the following two categories:

- Development problems – during development of the test user interfaces it became clear, that the development of the modules without graphical tool is a time and effort consuming task. The absence of validation for the GUI description was also reported as a problem during the development stage.
- Run time limitations – Keeping of the GUI definition in the database, despite the advantages, contains some drawbacks, especially in cases, when the application has more simple structure and is not using database itself.

The current article discusses the attempt of the authors to solve these problems using UsiXML and implementing a rule-based converter between the GUI descriptions, built using the two approaches. The input and the output of this module have identical structure thus it is transparent to the GUI interpreter.

The conversion between both descriptions can be realized in different ways. The rule-based approach is the natural solution because it provides flexibility; ease of integration with the existing solution and ability to cooperate with the other layers of GUI manipulation (Fig. 1) which are developed in the same way. The conversion with the rule-based engine and the usage of the same internal representation in both cases (UsiXML and database description) makes the form of the structure transparent. In this way the application logic, which uses and manipulates the GUI description, remains unchanged.

TASKS AND GOALS

Several aims are defined for the research application prototype of this article. They arose naturally as an evolution of a previous development [16, 17]. The solution should use UsiXML as a GUI description language as well as previously used database description. The application has to provide a converting mechanism which can transform UsiXML to database GUI description and vice versa. The following advantages are pursued while designing the prototype:

- The discussed solution attempts to provide a graphic environment (a UsiXML editor) for GUI description definition in order to improve the usability of the proposed interpreter. The approach should allow designers who are not familiar with the underlying technologies to describe GUI fast and easy. The tool should be used to define new or rearrange existing application views without the need to recompile the application for them to take place in the system.
- The XML schemas should provide validation of the description at an initial level to prevent propagation of problems in the system.

- The solution, discussed in [16, 17] uses GUI definition, stored in a database. In some cases, the usage of a GUI description, located in XML files on the server is the preferable solution. For some applications all the additional tasks related to the “GUI in DB” approach, including the database access, administration and user restrictions, are too complex and therefore must be avoided. The application developers will be able to fulfil this requirement using the suggested solution.
- The discussed tool should provide easy-to-use export/import functionality. UsiXML description gives the opportunity to realize the latter since the XML standards are widely used in case of server-server communication and in case of web services.

However, the proposed interpreter should combine the abovementioned advantages with the ones of a description of the GUI placed in a database. The most essential of them are:

- The usage of the database as a description container will enhance the reusability of groups of logically connected controls or sub-groups among different views in the application avoiding duplication. The groups can be used as building elements for complex structures thus improving readability.
- The database organization allows the definition of so called exclusion rules for hiding single controls or groups in a specific hierarchy of components thus implying *polymorphism*.
- The database supplies straightforward utilities of user authorization policy.

Both UsiXML and database descriptions are platform independent. As far as both realize the UIDL concept they isolate the GUI description from the domain business logic. They are flexible, robust and the parsing of a description written in any of the former is not time consuming. The conversion mechanism between the two provides the interpreter with the combination of the UsiXML and database description advantages.

SOLUTION OVERVIEW

Application structure

The approach discussed in this article consists of (Fig. 2):

- XML description – with respect to UsiXML schema of CUI description. In the current solution GrafiXML[12] editor is used to generate it;
- User interface description structure in a database;

- Converter, which translates UsiXML to database description and vice versa – an intelligence engine realized in CLIPS;

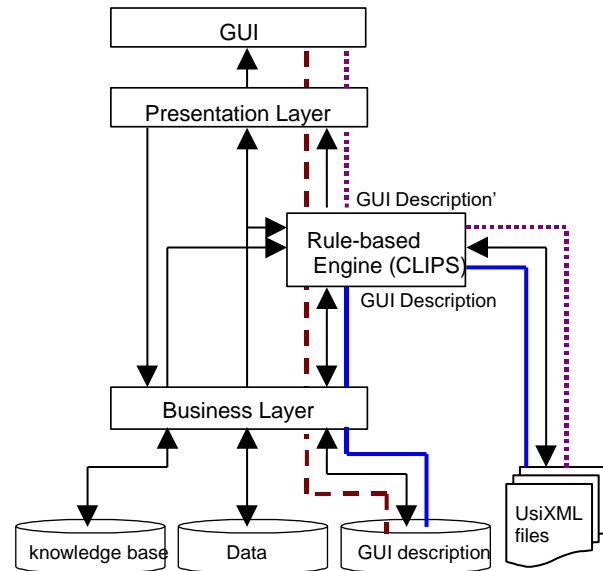


Figure 2: GUI generation process

- Intelligence module for GUI enhancement [16].
- Description interpreter realized in Java [17].

The solution is integrated in a server-side WEB application, designed to work with financial information (financial instruments, environments, etc.). The description is organized into groups of components, which facilitate their reuse in visualization of different instruments. The discussed approach is applicable in any domain where complex data is arranged in reusable logical units.

Fig. 2 presents the process of the GUI generation. First the GUI description is built. It can be extracted from the database directly using SQL statements over the description tables. On the other hand, the former can be obtained from a UsiXML description. The controls from the latter are loaded into the CLIPS engine as facts and after some rules have been applied to them the modified facts are converted back and proceeded to the business layer. On the next step additional information is extracted from the database: information about the currently selected instrument, which is also hierarchically organized (instrument, legs, cash flows, etc.), and information about the current user activities. All these three, including the description, are passed to the CLIPS engine once more for the GUI to be enhanced by the application of other rules. The format of Java objects, used before and after the conversion to CLIPS facts is identical, so the inclusion of the rule-based approach does not need any additional changes in the data representation and logic on the application server. Finally, the description is translated to a JSF component tree and displayed to the user.

The separation between the two passes in the CLIPS engine allows a background conversion between UsiXML and database formats. Thus, a UsiXML description can be imported into the database without a graphical representation, so does the export of a database description into a UsiXML file.

Business objects

The GUI description hierarchy consists of (Fig. 3):

- Controls (leaves in the hierarchy) which describe simple GUI controls like inputs and selects;
- Groups which contain controls and/or other groups (sub-groups). The groups on the highest level have special interpretation. For example, part of them represents tab control in a common tab holder of a form;
- Forms, defined with their identifiers, describing the views of the application;
- An instrument defines the financial instrument type to be shown on the GUI.

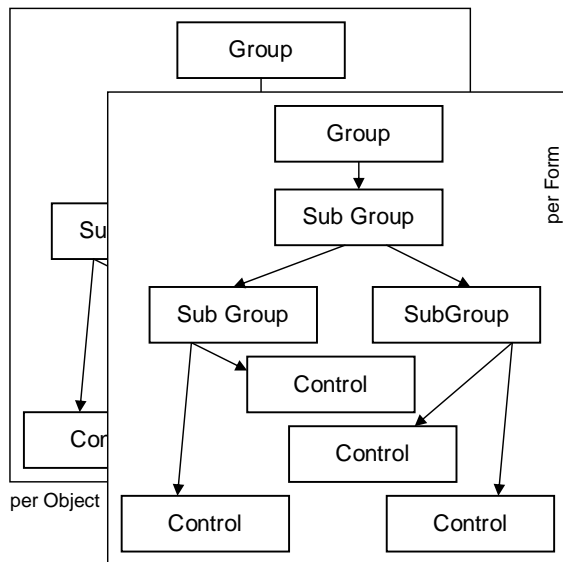


Figure 3: GUI description structure

Description element belongings such as control-group, group-group or group-instrument relations are also defined along with the (X; Y) coordinates of the element in its parent. Other components of the element description are control type, data binding, validation and presentation information, etc. The discussed solution supports complex control types like tables and charts.

SOLUTION KERNEL

XML Description

The XML schema used in the discussed solution is developed by the UsiXML team [13]. GrafiXML was used to generate the example for this article (Fig. 4).

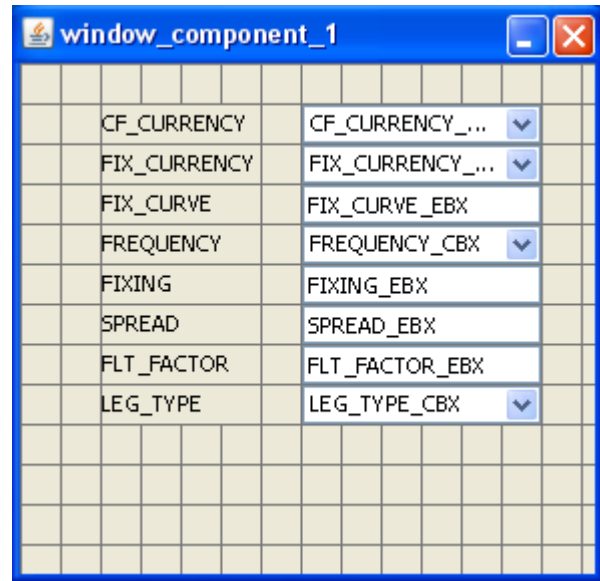


Figure 4: GrafiXML editor

Using the above-mentioned editor, a simple CUI description was generated. It was edited by adding custom attributes to its elements supplying specific information for the JSF description like validator, converter, etc. A fragment of the result .usi file is shown in Fig. 5.

```

<cuiModel id="p-test-v6-cui_10" name="p-test-v6-cui">
  <window id="window_component_1" name="window_component_1"
    width="297" height="290">
    <gridBagBox id="RATES__018" name="RATES"
      gridHeight="14" gridWidth="14">
      <constraint gridx="2" gridy="5" gridwidth="4"
        gridheight="1" weightx="1.0" weighty="1.0"
        fill="both" insets="0,0,0,0">
        <outputText id="FIXING_LBL_000"
          name="FIXING" isVisible="true"
          isEnabled="true" isBold="true" textColor="#000000"/>
      </constraint>
      <constraint gridx="7" gridy="5" gridwidth="6"
        gridheight="1" weightx="1.0" weighty="1.0"
        fill="both" insets="0,0,0,0">
        <inputText id="FIXING_EBX__032"
          name="FIXING_EBX" isVisible="true"
          isEnabled="true" textColor="#000000"
          maxLength="50" numberOfColumns="15" isEditable="true"
          istyle="text-align: right" itype="RW"
          iAccessRule="" iDataBind=".fixing" iValidatorID=""
          iConverterID="javax.faces.Number;pattern=# ##0.00%"/>
      </constraint>
      ...
    </gridBagBox>
  </window>
</cuiModel>

```

Figure 5: .usi file example fragment

GUIDescription Converter

The main module of the discussed solution is the converter that translates a description in one format to another format. If a description is defined in a UsiXML graphic editor it is read from the.usi file to a Java tree and the tree is processed to the intelligence CLIPS engine as facts. After that, some rules are applied to the latter altering the knowledge base (Fig. 6).

```
(defrule read_only_type
  ?ro_control<-(control_template
  (type ?itype&"false" | "null"))
=>
  (modify ?ro_control (type "RO"))
)

(defrule read_write_type
  ?rw_control<-(control_template
  (type "true"))
=>
  (modify ?rw_control (type "RW"))
)

(defrule map_controls
  ?lbl_control<-(control_template
  (controlType "Label")(type "RO")
  (label ?n1)(addPosition ?ap1) (labelStyle ?ls1))
  ?obj_control<-(control_template
  (controlType ~"Label")(addPosition ?ap2))
=>
  (if (= ?ap1 ?ap2) then
    (modify ?obj_control
    (label ?n1)
    (labelStyle ?ls1)
    (addPosition 0))
    (retract ?lbl_control)
  )
)
```

Figure 6: Description converter example rule

ID	GROUP_ID	STYLE	CONTROL	DATA_BIND	VALIDATOR_TYPE	LABEL_ID	CONVERTOR_TYPE
24	15 (null)		DropDown	position	com.ers.pm1j.ex.web....	POSITION	com.ers.pm1j.ex.web....
25	19 text-align:...		Edit	volume	(null)	VOLUME	javax.faces.Number;p...
26	19 text-align:...		Edit	nominal	(null)	NOMINAL	javax.faces.Number;p...

Figure 7: Description in the database (fragment)

Next a new tree with the same structure as the input one is formed based on the final facts. The resultant Java tree can then be stored as a database description to the aforementioned database tables or it can be directly interpreted to a JSF component and then displayed in the user browser.

The mechanism can work in the opposite direction. A description might be read from the database, then processed to the CLIPS engine and finally stored as a usi file or interpreted as HTML.

The rules applied to the CLIPS facts adjust the structure of the tree nodes. For example, in the database descriptions a control element contains information (Fig. 7) about the control and its adjacent label (e.g. label "Market price" with the value of 100.00) but in the UsiXML description the label and the control are specified as separate XML elements.

The presentations in all the levels (JSF component tree, Java internal representation, CLIPS template definitions) are the same, regardless of the type of the used source. After applying the rules on the facts, defining the UsiXML components, they are converted to facts, fully compatible with the ones, created based on the database source.

GUI DESCRIPTION INTERPRETER

After being loaded the GUI description is represented as a Java tree in spite of the source (database or XML). Next, it is converted to a JSF component tree. Java Server Faces (JSF) is a Java based WEB technology which was chosen for a variety of reason like support from any Java container (e.g. Tomcat, Apache, etc.); capacity; easy extensibility [4], etc. The interpreter is realized as a custom component which simulates a tab control. The latter renders only the necessary data according to the selected tab.

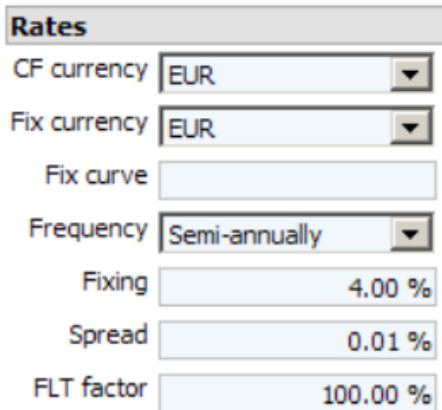
The generated tree has the following structure:

- root - attached to the standard JSF view tree;
- `- 1st level - tab views;
- `- 2nd level - layout tables;
- `- ... nested layout tables;
- `- leaves - simple or complex controls.

The JSF component tree consists of:

- layout tables which arrange elements on the page;
- pairs of label and input control for scalar values;
- data tables presenting collections;
- action components which invoke actions from the model.

This GUI generation approach creates only the necessary controls during the rendering phase. Thus, the discussed solution improves the readability of the view [17]. The visualization of the group from Fig. 4, Fig.5 can be seen on Fig. 8.



The screenshot shows a form titled "Rates" with the following fields:

- CF currency: EUR (dropdown)
- Fix currency: EUR (dropdown)
- Fix curve: (text input)
- Frequency: Semi-annually (dropdown)
- Fixing: 4.00 % (text input)
- Spread: 0.01 % (text input)
- FLT factor: 100.00 % (text input)

Figure 8: Interpreted description result

DATABASE GUI DESCRIPTION

Fig. 9 shows the relationships between the GUI description tables. The latter are as follows:

- *INSTRUMENT_TYPE* (nomenclature table) – contains the definition of all instrument types, the GUI is capable to handle. They have a type-subtype pair and a unique identifier.
- *FORM_TYPE* (nomenclature table) – contains the definition of the views, the application consists of. Every form has a unique identifier.
- *GROUP_DATA* –contains groups definition: identification information (unique identifier, group type) and visualization information (relative positions in both directions, group style, label, etc.).
- *CONTROL_DATA* – contains controls definition: control 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 table) – defines the hierarchy of the groups.
- *INSTRUMENT_GROUP_REL*, *INSTRUMENT_CONTROL_REL_OPP* (relational tables) – give additional constraints and rules for displaying or hiding groups and controls per instrument respectively.

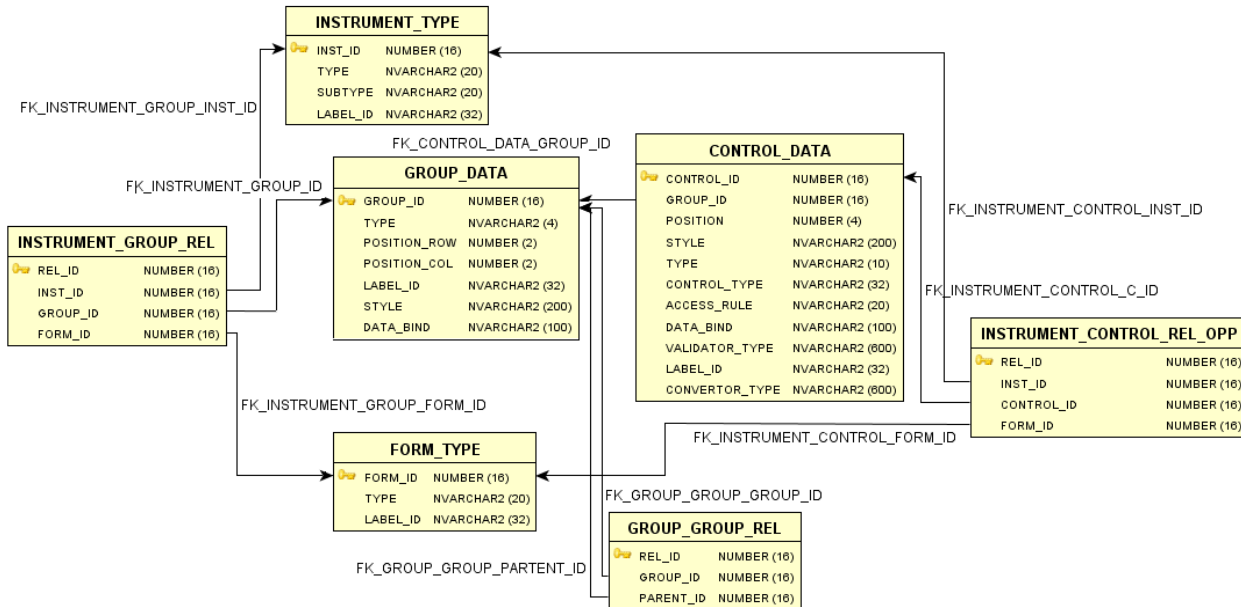


Figure 9: Database description tables

CONCLUSIONS AND FUTURE WORK

The discussed solution was developed and tested using sample data. Some results can be seen on fig. 8. 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.
- No delays were encountered when processing a description in either of the formats.
- Both UIDLs are more flexible and simple than using the standard rendering checks.
- The database-oriented approach benefits from the easier maintenance and higher availability of the data centre.
- 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);
- Using XML for description definition allows easy and intuitive input validation.
- Graphic editors are much more intuitive than any 3rd party database tool or text editor.
- Tree structured XML files are easier to understand and update in a text editor unlike the database records.
- The latter however provide recursive references to specific definition unlike XML tree structure.
- The rule-based approach provides an easy-to-extend framework, where other cases can be handled by adding new rules.

Advantages and Disadvantages of the Approach

The implemented solution combines the advantages of both XML and database approaches. They are both platform independent and cross-technology (they can be processed by Java as well as .NET). Changes in any of the formats take place immediately in the application without the need of recompilation and/or redeployment. Also, access to the GUI description is fast and robust supported by numerous libraries providing easy treatment. Using a database description improves the reusability of the group and/or control definitions. If used in a desktop application with a centralized database changes are applied simultaneously to all the database users. However, this type of description does not allow a built-in validation of the input structure as opposed to XML schemas. Moreover, XML description, unlike database one, can be changed in any simple text editor. Furthermore, UsiXML provides ready to use graphic editors which eases the GUI definition even more.

On the other hand, being separately developed languages the database description UIDL and the UsiXML specification are not entirely overlapping. This fact limits the possible elements used to define a GUI description. The database UIDL is JSF compliant which is too specific for the UsiXML concept but at the same time increases variety of components used for GUI definition which is very important for the needs of the application using the discussed solution.

REFERENCES

1. Ali, M.F., Perez-Quinones, M.A. and Abrams, M., Building Multi-Platform User Interfaces with UIML. in Multiple User Interfaces, John Wiley and Sons, UK, 2004.
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonck, J., A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15(3), 2003, 289–308.
3. Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., Vanderdonck, J., Sketchi-XML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML. in Proc. of Tamodia'2004
4. Geary D., Horstmann C. S, Core Javasever Faces [JsF], *Prentice Hall PTR*, June 2004
5. Giarratano J. Expert Systems: Principles And Programming, 4Ed, ISBN: 053438447, CourseTec Ltd. 2004
6. Hoiem, D., Lovell, M., Method and apparatus for a unified user interface", US patent 7,512,899, 31 March 2009. , Microsoft Corporation, <http://www.freepatentsonline.com/7512899.html>
7. J van de Vegte, J., Som de Cerff, W.J., A Dynamic GUI for accessing the Netherlands SCIAMACHY Data Center, Montreal, Canada, May 2000
8. Limbourg, Q., Vanderdonck, J. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence, *In Proceedings of Workshop on Device Independent Web Engineering DIWE'04*, Munich, July 2004
9. Limbourg, Q., Vanderdonck, J., Michotte, B., Bouillon, L., Florins, M., Trevisan, D., UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces, *In Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"*, Gallipoli, May 2004
10. Limbourg, Q., Vanderdonck, 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.

11. Luyten, K., Abrams, M., Vanderdonck, J., Limbourg, Q. Developing User Interfaces with XML: Advances on User Interface Description Languages, Advanced Visual Interfaces, Italy, 2004.
12. Michotte B., Limbourg Q., Vanderdonck J. GrafiXML, A User Interface Builder Based on UsiXML, IAG, Louvain-la-Neuve, July 2004
13. Michotte B., others UsiXML CUI schema specification team,
<http://www.usixml.org/index.php?mod=pages&id=5>, 2006
14. 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.
15. Myers B., Hudson S. E., Pausch R., Past, Present, and Future of User Interface Software Tools, *ACM Transactions on Computer-Human Interaction*, Vol. 7, No. 1, March 2000.
16. Paskalev, P., Rule based GUI modification and adaptation, *In proceedings of International Conference CompSysTech'09*, Bulgaria, June 2009
17. Paskalev, P., Serafimova, I., Runtime Generation of an User Interface, Described in a Database, *In proceedings of International Conference CompSysTech'09*, June 2009
18. Puerta, A. and Eisenstein, J. XIML: A Common Representation for Interaction Data, Sixth International Conference on Intelligent User Interfaces, 2002.
19. Shaer O., Green M., Jacob R., Luyten K., User Interface Description Languages for Next Generation User Interfaces Workshop, *CHI 2008 Proceedings*, Florence, Italy, 2008