

# Rule-Based GUI Modification and Adaptation

Plamen Paskalev

**Abstract:** *The paper describes an approach for run-time modifying and adaptation of a complex GUI using a rule-based system. Elements of Intelligent User Interfaces (IUI) paradigm, involved in this approach are discussed. A WEB-based solution, implementing the ideas, discussed in the article, is overviewed. The problems, technical solutions and the advantages of the selected approach are listed.*

**Keywords:** *Intelligent User Interfaces (IUI), Rule based systems, Clips, Real time GUI generation, Computer Systems and Technologies*

## INTRODUCTION

The development of a complex user interface meets numerous challenges: It has to have consistent and predictable design, to display the right amount of information (not too much and not too little). The internal dependencies, the relations to other screens, the changes in the GUI depending on the data to be displayed (showing or hiding controls or groups of controls, making fields read-only, moving them to different location, etc.), validating of the values, entered from the user – implementing of all these obligatory parts of the user interface can lead to program code, which is too difficult to handle. The programmer tasks were simplified with introducing programming languages like Java or C#, providing features like data binding, control properties etc., although GUI design and development can be still a challenge, especially in case of complex applications. Another aspect, which adds complexity to the user interface handling, is including of some aspects of Intelligent User Interfaces in the application's logic. The attempts to trace user behavior, 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 – it doesn't help in making the GUI solution more simple, robust or traceable.

This article describes an approach, used for realization of the GUI of a WEB based application. The GUI is defined using an internal formal specification. The both presentation logic and IUI modifications are applied over the descriptions in form of CLIPS AI system rules. The advantages of the discussed approach are classified. The transformation and technical challenges in the realization are discussed, based on some results from the real system.

## DYNAMIC MODIFICATION OF THE USER INTERFACE

The interaction between the users and the user interface should expose a lot of intelligence and dynamic behavior because of following reasons:

- Complex interfaces, where on a single screen large amount of heterogeneous data has to be shown (like health information systems, financial software, complex industry controlling software, etc.), need dynamic control, as the effort for building and testing of it can easily reach very high levels.
- Users don't read regularly on-line helps and application description. They would prefer to start working immediately with the application. The user interface should be intuitive and should be constructed dynamically following user experience.
- The different users have different skills and expertise levels. Moreover, the skills are changing with time. An adaptable interface would be able to provide detailization, which corresponds to the user knowledge and goals.

The points, listed above make the usage of intelligent kernel meaningful for analysis the user input according to context, constructing GUI fragments dynamically and controlling the overall user interface at run time.

There exist various investigations in automatic GUI generation, starting from support in design process and development of tools (see [14] for some examples), to attempts for automatic or semi-automatic building of graphical presentations [6] or interfaces (project GUITARE, [13], etc.). There were two main ideas, which became a basis for further investigations: First, since systems like AIPS [3], it was clear, that the presentation layer has to be clearly separated from the other part of the application; and second, the ability of run-time adaptation of the user interface will depend on the ability to collect information about the user activities. Several investigations have focused the extracting of personal data for the active communication between the user and application [4,15]. Using the collected data, the application can deduce the user competency, preferences and goals, the application can automatically reshape the GUI and to communicate with the user in more applicable and understandable form. Additionally, the interface can be able to adapt itself automatically to solve some common problems the user has in communication, for example to extend the acceptance area around a button, realizing that the user is repeatable clicking around it [5]. Several solutions [7] are based on grammars and definitions, describing specialized graphical language, other are defining the layout design as a constraint satisfaction problem [11].

The solution, discussed in the current article uses semi-automatic approach for generation of GUI. It combines a presentation layer effort for dynamic changes of the GUI elements (hiding controls, groups, making fields read-only, changing of the labels, changing types of controls: from edit boxes to drop down lists, etc.) with adaptation activities, based on the collected information for the user actions. The tasks are performed in two-layered set of rules, applied on the description of the GUI, represented in form of facts. Using of a knowledge base and rule technology is not new in dynamic GUI generation [16], for building multi-modal applications, etc. In an earlier article [9] an approach for building of platform-independent GUI with description, formed as CLIPS facts was presented.

## **SOFTWARE REALIZATION**

### **1. TASKS OF THE SELECTED APPROACH**

There are two tasks this solution was expected to simplify: control of a complex user interface and adaptation of the user interface based on collected information about the user activities. Although with different purpose these two tasks are actually steps of modification of the user interface according to some pre-defined criteria. The intention to use rule based approach for these two tasks was to provide maximum flexibility, simplicity and integration between these steps where it is necessary. One important requirement was for flawless integration of the rule based mechanism into the business layer code.

### **2. REALIZATION**

The application, implementing this approach, is a WEB server application, written in Java. This is a financial application, which has to handle different and, in some cases, quite complex financial instruments. The description of the GUI exists in a specially designed set of data tables in the database [10]. The application loads the descriptions in and converts them to a hierarchical structure of objects. The objects define the appearance of the graphical elements (position, style, type, data binding, etc.). The following types of objects these hierarchies consist of:

- Forms – correspond to the application views. The form object includes only one (root) group object.
- Groups – objects, which are able to realize the GUI hierarchy. A group contains ordered set of other sub-groups or controls.
- Controls – the leaves in the hierarchy. A control object describes a (Label, Control) pair of a single control (edit box, drop down list, radio button, etc.).

Independently, the business layer loads two other sets of data (fig. 1a):

- Instrument data: The information for the currently selected instrument. It also has hierarchical structure (instrument, legs, cash flows), which describes the instrument completely (dates, coupons, conventions, amortization schedule, etc.).
- Information for the user activities. The traces of the previous activities of the current user are being kept in a knowledge base (discussed further below).

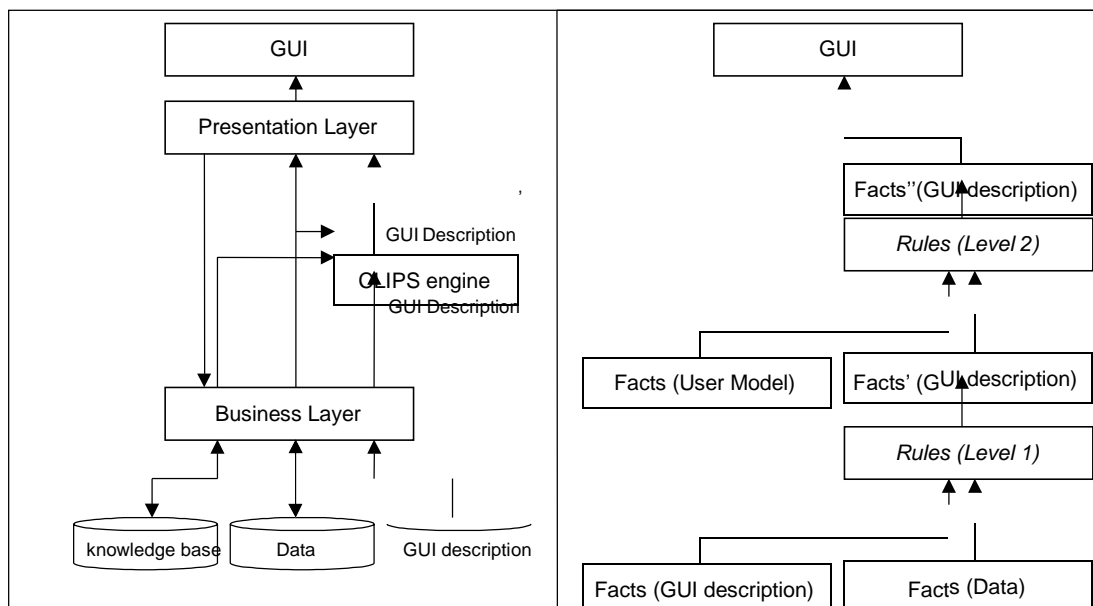


Figure 1a: General structure of the application

Figure 1b: Rules hierarchy

The descriptions of the instruments, user activity and GUI definition are converted to facts and loaded into CLIPS engine. The rules (fig. 1b), responsible for completing the two tasks, discussed above, after invoking, modify the GUI definition facts, leading to a new GUI, which match the constraints and requirements of these rules. The modified facts then are converted back to the internal Java class objects and are delivered to the renderer for building the GUI. The format of Java objects, used before and after conversion to CLIPS facts are identical, so the including of the rule-based approach don't need any additional changes in the data representation and logic on the application server.

### 3. TRANSFORMATIONS SEQUENCE

The hierarchical structure of objects is converted to CLPIS *deftemplate* constructs. The *deftemplate* definitions for group and control facts are shown in fig.2. The hierarchy is coded in (ID, parent Id) pair. Thus, the rules in CLIPS can find the group, the current group or control belong to. A sample fact, created during the transformation step is shown below:

```
(control_template (ID 33) (groupID 18) (position 5) (style "text-align: right")
(type "RW") (controlType "Edit") (accessRule "null") (dataBinding "leg1.spread")
(validatorType "null") (label "SPREAD") (convertorType "javax.faces.Number"))
```

### 4. GUI CONTROL

The first tasks to be solved is to handle the GUI elements according to the data to be shown. The pre-defined GUI description was created in its most universal form. It, together with the data for the financial instrument, is introduced as facts and they invoke one or more rules, which actually modify the GUI. Several rules were realized up to now:

- Hiding of fields, which are not relevant for a fix or float leg of an instrument; making second currency field read-only if the instrument has one currency only.
- Hiding of group of controls, defining conventions, for simple contracts.

- Removing of the amortization table in case the amortization dates coincide with the cashflow dates.
- Hiding of amortization table in case the amortization type of the instrument is set to 'bullet'.

The set of rules can be easily extended with adding of new rules.

<pre>(deftemplate group_template "P_Group"   (slot ID      (type INTEGER))   (slot type    (type STRING))   (slot parentID (type INTEGER))   (slot positionRow (type INTEGER))   (slot positionCol (type INTEGER))   (slot labelID  (type STRING))   (slot label    (type STRING))   (slot style    (type STRING))   (slot extBinding (type STRING)) )</pre>	<pre>(deftemplate control_template "P_Control"   (slot ID      (type INTEGER))   (slot groupID (type INTEGER))   (slot position (type INTEGER))   (slot style    (type STRING))   (slot type     (type STRING))   (slot contolType (type STRING))   (slot accessRule (type STRING))   (slot dataBinding (type STRING))   (slot validatorType (type STRING))   (slot labelID  (type STRING))   (slot label    (type STRING))   (slot convertorType (type STRING)) )</pre>
--	--

Figure 2: Deftemplate definitions, describing a group (P\_Group) and control (P\_Control)

## 5. USER TRACES

As defined in [2], inflecting an interface means organizing it to minimize typical navigation. In practice, this means placing the most frequently desired functions and controls in the most convenient locations for users to access them, while the less frequently used functions will be moved deeper into the interface. Rarely used features should be removed from the common workspace. This is an example of the work; the application can have done in order to let the user feel more comfortable. As defined in [15], an adaptive interface renderer requires three inputs: an interface specification (I), a device model (D), and a user model, which can be represented in terms of user traces (T). The tracing of the user actions is a common practice in the applications, which are providing intelligent interface to the users and web applications. In AVANTI [12] the system tracks the user interaction in a web browser designed as a front-end of the AVANTI information system. In our realization, the interface specification and user traces are given in form of facts, the device model is ignored as far as the using of different devices goes beyond the goals of this research.

A user trace (T) is a set of trails where, according [8], the term trail refers to “coherent” sequences of elements manipulated by the user. The trail T is defined in a similar to the used in [15] way - as a sequence of events  $U_i$ , where  $U_i$  is a tuple  $(e_i, Vold_i, Vnew_i)$  – the interface element manipulated and its old and new value respectively.

There are two main differences between the approach from this article and [15]: a trail ends when the user switches to another frame or selects another instrument; due to the specific of the WEB applications the sequence of the controls, modified in the view can't be resored on the server side (All the modified controls are received together with the HTTP response).

When the application accumulates enough trace information, it can be used in adaption rules. The information is being kept on per user basis. The current user is determined during the login procedure.

The structure of the data tables, used to collect the information about the user activities is shown on fig. 3. The traces, trails and events are linked together using relational tables *KB\_TRACES\_REL* and *KB\_TRAILS\_REL*. On the next step the

information is loaded to the CLIPS engine in form of facts and is used for making conclusions concerning the next user activities.

## 6. RULES IMPLEMENTATION

The proposed solution is based on CLIPS engine which uses rule-based language with a clausal logic. A rule-based language is a declarative language, which binds logic with rules [1]. A rule, takes the form:

H  $\Rightarrow$  B: where H is the rule head and B is the rule body.

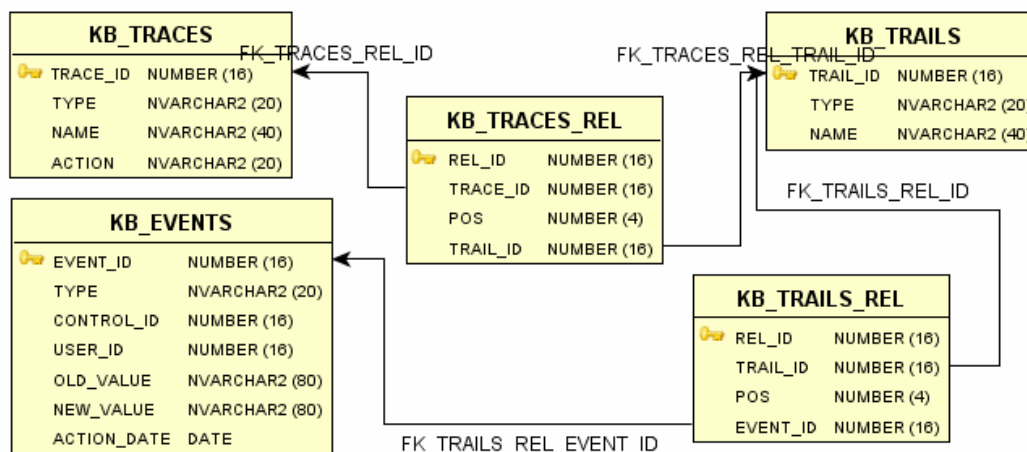


Figure 3: Structure of the KB for collecting User Traces

If the condition defined in the rule body is true, then it will trigger some actions. The rules, implemented in the discussed solution include a) rules reconfiguring the UI and b) rules, providing adaptation of the UI.

The rules are immediately executed when they are applicable. The reasoning process is data driven, rules are inserting the result facts into the fact base. Other rules are activated by inserted facts.

<pre>(defrule <b>instrument_currency</b> (instrument_template   (instrumentType "Swap")   (instrumentSubType ~"Cross Curr")) ?sec_currency_control&lt;- (control_template   (dataBinding "leg2.currency")   (type "RW")) =&gt; (modify ?sec_currency_control (type "RO")))  (defrule <b>float_leg</b> (leg_template (legType "FLOAT")) ?legflt_coupon&lt;- (control_template   (dataBinding "leg2.coupon")) =&gt; (retract ?legflt_coupon))</pre>	<pre>(defrule <b>fix_leg</b> (leg_template (legType "FIX")) ?leg_fix_currency_control&lt;- (control_template   (dataBinding "leg1.fixCurrency")) ?leg_fix_curve_control&lt;- (control_template   (dataBinding "leg1.fixCurve")) ?leg_fixing_control&lt;- (control_template (dataBinding   "leg1.fixing")) ?leg_spread_control&lt;- (control_template   (dataBinding "leg1.spread")) ?legflt_factor_control&lt;- (control_template   (dataBinding "leg1.fltFactor")) =&gt; (retract ?leg_fix_currency_control) (retract ?leg_fix_curve_control) (retract ?leg_fixing_control) (retract ?leg_spread_control) (retract ?legflt_factor_control))</pre>
---	--

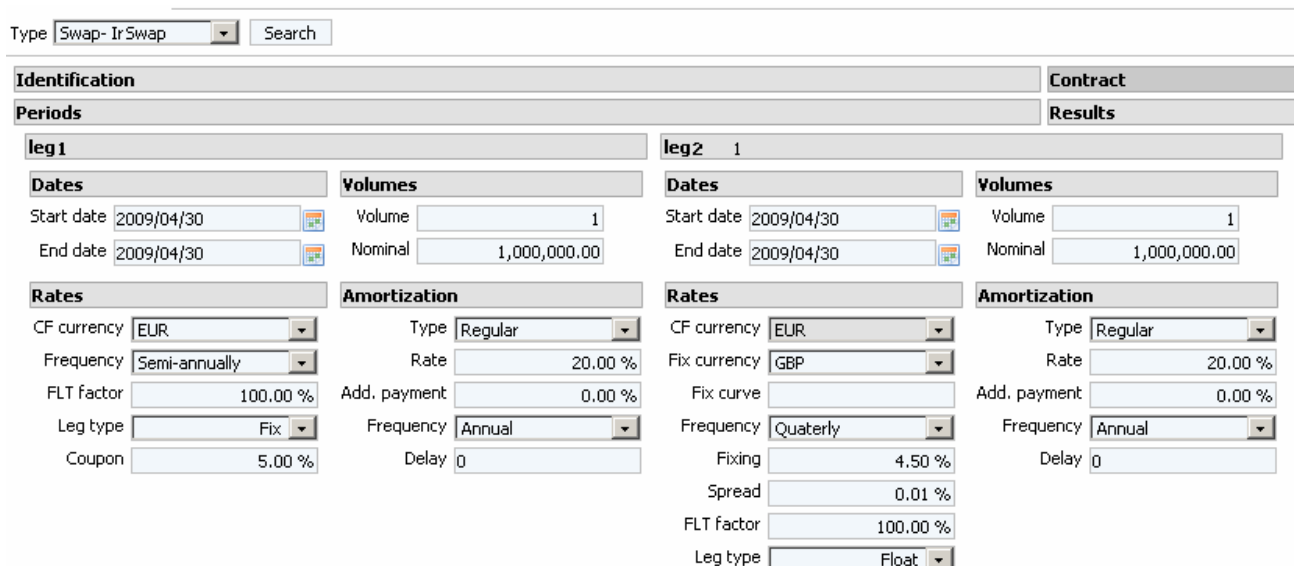
Figure 4: Rules, used in the reasoning process

## EXPERIMENTAL RESULTS

The discussed approach was implemented in a WEB server application, realized in Java. The discussed solution was developed and tested using sample data. The approach was compared to a standard-approach version of the application. The CLIPS engine was compiled as a dynamic library and accessed via JNI. The rules were placed in a text file, which is being loaded during the initialization phase. The multi-user capabilities are realized using different CLIPS environments for every user. The users are identified using their credentials from login-in process.

On fig. 4 three rules are shown: *instrument\_currency* rule makes the second drop-down for currency read-only if the instrument is not using two currencies. *float\_leg* rule removes the coupon from the leg definition in case float leg is to be displayed (this field has no meaning for float legs). In a similar way *fix\_leg* rule removes the controls, which are not used for fixed legs.

The result from applying these rules on the common leg GUI definitions can be seen on fig. 5. The first leg (leg1) is a fix leg, so it doesn't need fields for *Fix Curve*, *Fixing*, *Spread*, *Flt Factor*. The second leg (leg2) is a float one, which has no need for *Coupon* field. These fields are removed using the *fix\_leg* and *float\_leg* rules, discussed above from *Rules* group from both legs. In addition, the *CF currency* field for the second leg is made read-only according to the rule *instrument\_currency*.



The screenshot displays a web application interface for configuring financial instruments. At the top, there is a 'Type' dropdown menu set to 'Swap-IrSwap' and a 'Search' button. Below this, the interface is organized into several sections: 'Identification', 'Contract', 'Periods', and 'Results'. The main content area is divided into two columns for 'leg1' and 'leg2'. Each leg has its own set of controls for 'Dates', 'Volumes', 'Rates', and 'Amortization'.  
 For leg1 (a fixed leg):  
 - Dates: Start date 2009/04/30, End date 2009/04/30.  
 - Volumes: Volume 1, Nominal 1,000,000.00.  
 - Rates: CF currency EUR, Frequency Semi-annually, FLT factor 100.00%, Leg type Fix, Coupon 5.00%.  
 - Amortization: Type Regular, Rate 20.00%, Add. payment 0.00%, Frequency Annual, Delay 0.  
 For leg2 (a float leg):  
 - Dates: Start date 2009/04/30, End date 2009/04/30.  
 - Volumes: Volume 1, Nominal 1,000,000.00.  
 - Rates: CF currency EUR (read-only), Fix currency GBP, Fix curve (empty), Frequency Quarterly, Leg type Float.  
 - Amortization: Type Regular, Rate 20.00%, Add. payment 0.00%, Frequency Annual, Delay 0, Fixing 4.50%, Spread 0.01%, FLT factor 100.00%.

Figure 5: Screenshot from the test application using the discussed approach

## CONCLUSIONS AND FUTURE WORK

The following conclusions can be 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 rule-based approach provides an easy-to-extend framework, where other cases can be handled by adding new rules.
- Although the prototype is fast enough, some steps for reducing of conversions (Java ↔ CLIPS, CLIPS ↔ Java) should be considered.
  - 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);
  - The testing of the discussed approach is much easier and less time-consuming than the standard solution.

- The rule-based user interface module extends the handling of the user responses, adding intelligent user interface features.
- The adaptation has to be done very carefully because frequent changes to the interface can be distracting.
- The application has to be able to work properly also with empty knowledge base. The idea, suggested in [15], that the UI designer can provide few typical user traces as samples is planned to be used.

## REFERENCES

- [1] Abiteboul, S., Grumbach, S., A rule-based language with functions and sets. ACMTrans. Database Systems. 16, 1–30.1991
- [2] Cooper, A., Reimann, R., Cronin, D., About Face. The essentials of Interaction Design, 3 ed., Wiley Publishing 2007
- [3] Zdybel, F., Greenfeld, N., Yonke, M., Gibbons, J., An information presentation system, In 7<sup>th</sup> International Joint Conference on Artificial Intelligence, Vancouver, Canada, 1998.
- [4] Price, B., Greiner R., Häubl, G., Flatt, A., Automatic Construction of Personalized Customer Interfaces, Proceedings of the 11th international conference on Intelligent user interfaces, 2006.
- [5] Koelle, D., Intelligent User Interfaces, <http://web.cs.wpi.edu/Research/airg/IntInt/intint-outline.html>, 1996.
- [6] Roth, S., Kolojejchick, J., Mattis, J., Goldstein J., Interactive graphic design using automatic presentation knowledge, Readings in Intelligent User Interfaces, Morgan Kaufmann Publ, 1998
- [7] Mackinlay J., Automating the Design of Graphical Presentations of Relational Information, Readings in Intelligent User Interfaces, Morgan Kaufmann Publishers, 1998
- [8] Wexelblat, A., Maes, P., . Footprints: History-rich tools for information foraging. CHI Proc.,1999.
- [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] Paskalev, P., Serafimova, I., Runtime Generation of an User Interface, Described in a Database, In proceedings of International Conference CompSysTech'09, 2009
- [11] Graf W., Constraint-based graphical layout of multimodal presentations, Readings in Intelligent User Interfaces, Morgan Kaufmann Publishers,1998
- [12] Paramythis, A., Savidis, A., Stephanidis C. AVANTI: a universally accessible web browser. Proceedings of the HCI, New Orleans, USA, 2001
- [13] GUITARE Esprit 29056 Project, <http://giove.cnuce.cnr.it/Guitare/index.html>, 2001
- [14] Clerckx, T., Winters, F., Coninx, K., Tool Support for Designing Context Sensitive User Interfaces using a Model Based Approach, In proceedings of TAMODIA 2005
- [15] Gajos, K., Weld, D., SUPPLE: Automatically Generating User Interfaces , Proceedings of the 9th international conference on Intelligent user interfaces, 2004
- [16] Arens, Y., Miller, L., Sondheimer, N., Presentation Design using an Integrated Knowledge Base, Readings in Intelligent User Interfaces, Morgan Kaufmann Publishers, 1998

## ABOUT THE AUTHOR

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