Rule-Based Framework for Intelligent GUI Adaptation

Plamen Mihaylov Paskalev, Anatoliy Stefanov Antonov

ABSTRACT
Building of a rule-based framework for dynamic adaptation of a complex GUI is discussed in the article. The need of such a solution and using of approaches from Intelligent User Interfaces (IUI) paradigm are augmented. Ways of collecting the information for the user activities and algorithms for conclusions based on it are described. The problems, technical solutions and the advantages of the selected approach are overviewed.

KEY WORDS
Intelligent User Interfaces (IUI), Rule based systems, Clips, Machine Learning

1. Introduction

The user interface is an essential part of every application. The design and development of the GUI for a new application has to answer many, in some cases opposite requirements: the interface has to be clear, user-friendly, predictable and consistent, to show the right amount of information. The GUI design becomes a challenge especially in case of complex systems with large amounts of data to be shown.

The work with such applications is often a challenge for the user too. He has to become familiar with a large number of screens, sequences of actions, controls, features etc. The applications of this type are equipped with user guides and online help systems, but it remains a tremendous task for at least part of the users to understand the logic of the application. The user interface should be intuitive and should be constructed dynamically following the user experience.

There is one aspect, connected with the two, discussed above: the marketing of the newly developed application. If the application provides rough interface, one, the user is not familiar to or interface, which doesn’t help the user in performing his every day’s tasks then the application most likely will be not a financial success.

One approach to lower the pressure for the user is to give to the system the ability to learn the behavior of the individual users and to adapt itself according it.

This article describes an approach for building of adaptive interfaces, the problems encountered and the solutions, found during the development. Although not a general solution, able to be used in any domain, it was designed not as a part of a specific application, but rather as a framework, which can be used in various scenarios.

In the next chapter an overview of the adaptive interfaces as part of Intelligent User Interfaces (IUI), together with review of some related works is given. The solution’s architecture, data structure and algorithms used are discussed in chapter 3. The behavior and technical challenges in the realization are discussed in chapter 4, followed by some results from a real prototype work in chapter 5. In chapter 6 the advantages of the discussed approach are classified, and the opened points are listed.

2. Adaptive Interfaces

2.1 Using of adaptive interfaces paradigm

The main idea behind using the adaptive interfaces is to adapt the system functionality or user interface or both to each individual user [1]. Known under different names as adaptive interfaces, user modeling systems, software agents, intelligent agents or personalization [1], the approach includes collecting and using of information for the user, identified by its login name, IP address, etc. Based on a learning algorithm, the system is capable to deduce some kind of user model, which is then used as a basis for the adaptation of the behaviour of the system.

In [1] the user-adaptive system is defined as (fig.1):
“An interactive system that adapts its behaviour to individual users on the basis of processes of user model acquisition and application that involve some form of learning, inference, or decision making.”

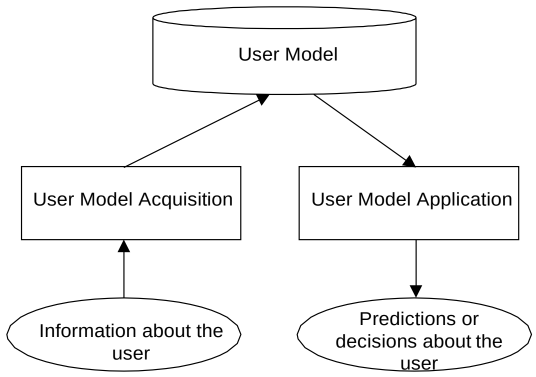
User Model
User Model Acquisition
User Model Application
Information about the user
Predictions or decisions about the user

Figure 1: user-adaptive system [1]

The following prerequisites for the implementing of such a system can be defined:

- Complexity of the applications – Today, and moreover in the future most of the people already use or will use one or more complex applications in their everyday life. Using of adaptable systems, i.e. systems, which can be modified by the user, will be not sufficient to reduce the stress of the user and to avoid potential mistakes. As it was discussed in [23], the users don't tend to adapt the interface radically. And it is not clear, whether if they use this opportunity, they make it in the most effective and useful way ([24]).
- Amount of data – Even if the application is relatively simple, the users now have access to significantly larger amounts of information. Internet access, search engines, multimedia, different types of documents are only part of the reasons for this phenomenon. The idea to delegate part of the tasks for maintaining this array of information to the software is not only attractive, but also is quite necessary.
- Different types of users – As it was discussed above, the software can be used from different people, with different skills, competency level, education, nationality, age, etc. Moreover, the skills are changing with time. It is more and more difficult to create application, suitable for all the types of users, for all contexts of usage.
- Complex interfaces, where on a single screen large amount of heterogeneous data has to be shown (health information systems, financial software, industry applications, etc.), need dynamic control, as the effort for building and testing of such control can be difficult and time-consuming task.

The main directions, in which the adaptation can help the communication between the system and the user, are summarized below:

- Personalization of the user interface – The system is able to modify the user interface in order to make it closer to the way the user is working with it. The elements of the GUI (controls, menus, forms) are being automatically modified.
- Automatic help, advice or suggestions generation – based on assumptions concerning the user goals and expertise level, the generated information is expected to help the user in performing his tasks and understanding the logic of the system.
- Content personalization – based on the previous user activities, the system is able to filter the information to be shown on the forms, in the lists, selections, etc.
- Automatic help in searching information, making choice, etc. – The system is able to help the user in finding electronic documents, preparation of the queries to the search engines, proper displaying of the information, the user might need, etc.
- Performing of sequences of routine tasks – Tasks or sequences of tasks, which are performed often and consume the user's time and attention, can be performed from the system.

2.2 Related Works

There exist various works in each of the directions, discussed in the previous chapter.

In [2] the interaction between the user and the interface of an online shop is being investigated. The interface is being adapted both between the sessions and during the sessions. Special attention is paid to the correlations between the products bought. The complex correlations lead to complex dependencies and rules in an attempt to find the most convenient way of adaptation.

Microsoft MS Office Assistant [3] provides context-dependent automatic help in using the application. Based on Lumiere prototype ([4]), it uses Bayesian models for representing the user behaviour, includes own language for definition of the events, uses together the conclusions and the user actions and inputted information. The prototype calculates the probability of whether the user needs the advices of the system.

As defined in [20], inflecting an interface means organizing it to minimize typical navigation. In some cases, 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 (Microsoft SmartMenus, etc.). Rarely used features should be removed from the common workspace. This is an example of the work the application can done in order to let the user feel more comfortable.

SwiftFile ([5]) analyzes the way the user is sorting emails in different folders and tries to predict the 3 folders, where the next email is going to be placed with higher probability. To be able to do this, the system presents the folders as vectors, weighted on the words, which appear in the mails. When a new mail arrives, the distribution of the words in it is compared to the ones, already represented in the vectors. The 3 potential folders are then visualized as options via 3 buttons on the screen. The user can use the buttons or to ignore the suggestions.

The observation of the user actions is the usual way of collecting data for the intelligent assistants. AVANTI ([6]) is an example of this approach. The system is watching the communication between the user and the AVANTI Web browser, which provides opportunities for adapting its behaviour according to the user's abilities, requirements and preferences.

In a similar way, the DESK ([7]) add-in to the system for generation of dynamic Web pages Pegasus is watching the actions of the users and is collecting data for the steps of modification of the pages. The system tracks down all the main actions of the user (inserting text, change of the style, etc.) and tries to determine the context. Finally, it builds primitives, which are used as a basis for creating of a model of the communication.

An approach for adjustments of the application in order to meet some specific requirements of people with disabilities were discussed in [8]. The problem is identified (for example repeatable double keyboard button click instead of single click) and an interval is determined, where to ignore the second click. Similar idea was discussed in [19] – the system extends automatically the acceptance area around a button, realizing that the user is repeatable clicking around it.

In [9] different variants for analysis of the user actions are examined in order to make some conclusions on his interests.

Plan recognition as an attempt for predicting the user actions, based on assumption, that the user actions are parts of his plan to reach a goal or to complete a task, are discussed in [10]. This approach, together with using of stereotypes ([11], [12]) to present the user's characteristics (competency, education etc.) requires creating of precise user models and thorough knowledge of the domain.

3. Adaptive Interface Framework

2.1 Goal of the Research

The aim of the research, discussed in the current article was to build a framework, which can be used for adaptation of user interfaces of complex applications. The following requirements were defined to the results of the investigation:

- The framework is expected to work well with screens with many controls.
- The investigation must be platform independent, i.e. not connected with a certain type of applications (Web, Windows).
- Framework should be capable of using different software solutions, not to be tied to a certain data / project.
- The last direction of the previous chapter, *performing of sequences of routine tasks*, was determined as the first goal of the investigation.

The investigation is part of a larger project ([13], [14],[15]), which examines the different aspects of Intelligent User Interfaces (IUI), applicable to medium-sized software projects.

The current article discusses the ways of collecting information for the user activities, using this information for acquisition of the user model (fig. 1), and building dynamical suggestions for the user.

2.2 Application Structure

The solution, discussed in the previous articles ([13], [14]) uses semi-automatic approach for generation of GUI. It combines a presentation layer effort for dynamic changes ([13]) 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 application is based on using rules, which is common practice in dynamic GUI generation [16].

The proposed solution is based on CLIPS engine which uses rule-based language with a clausal logic. The reasoning process is data driven; rules are inserting the result facts into the fact base. Other rules are activated by inserted facts.

The GUI is described in form of CLIPS ([17]) facts. The description is platform-independent ([18]). The tasks are solved via applying two-layered set of rules ([14]) on the facts, describing the GUI. The current article discusses the rules from the second set (fig. 2), providing adaptation of the UI.

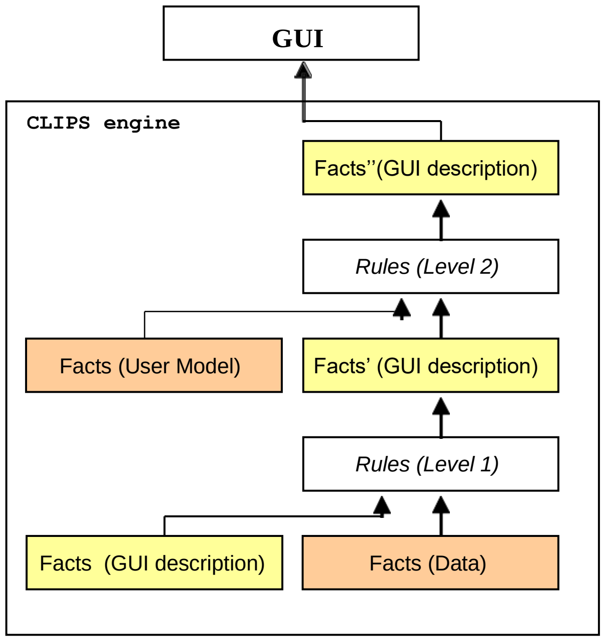

Figure 2: Application structure

The CLIPS module can be integrated into projects on different platforms and operating systems: Windows applications using standard API, .NET managed code or Windows Presentation Framework (WPF) GUI description, Web applications, Java-based server applications with Java Server Faces (JSF) GUI, etc. It can be compiled for different operating systems (Windows, Linux, Solaris, etc.).

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 ([13]):

- 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 – A control object describes a <Label, Control> pair of a single control (edit box, drop down list, radio button, etc.).

2.3 Building the GUI

The data objects, user activity and GUI definition are converted to facts and loaded into CLIPS engine. The rules from group 1 (fig. 2) modify the GUI definition facts, leading to a new GUI, which match the constraints and requirements of these rules ([14]). The rules from group 2 are the object of the current article. They analyze the data for the user activities and modify further the GUI description. The GUI description is converted back to the internal class objects and is used for building the GUI. The format of the input and output CLIPS facts is identical, so the including of the rule-based approach is transparent and doesn't need any changes in the data representation and logic of the application.

4. Adaptation of the GUI

4.1 User Traces

As defined in [21], an adaptive interface 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 for collecting information for the user's behaviour. In AVANTI [6] 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 (I) and user traces (T) are given in form of facts, the device model (D) 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 [22], 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 [21] way - as a set 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 discussed approach and the one in [21]: Due to the specific of the Web applications the order of the events of a trail can't be restored on the server side. All the modified controls are received together with the HTTP response. The trail is assumed to end either when when the user switches to another page or invokes an action (pressing button, menu item, etc.).

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

4.2 Learning Algorithms

Current version of the discussed solution is able to handle one of the adaptation directions, discussed in chapter 2: *Performing of sequences of routine tasks*. The applied rules search for the following recognizable patterns:

- Repeatable actions, performed over the content of a single control
- Sequences of actions

4.2.1 Repeatable actions

The first stage is searching for repeatable actions. The action a_i is defined as the triple

$$a_i = < C_i , As_i , V_i >$$

where C_i is the identifier of the control, where the action was identified, As_i defines the type of the action and V_i is the value of the action. For each recognized pattern there is a new fact added. The actions then are groupped using the following triples

$$Sa_i = < a_i , Tsa_i , w_i >$$

where Tsa_i denotes the trails, the action was found in and w_i denotes the weight, assigned to this action.

Some of the recognized actions are given in table 1.

4.2.2 Actions set

On the second stage the information for the isolated single actions is examined. The goal is to extract information for repeatable sequences of actions.

A set of actions can be described as

$$Ma_j = < A_j , T_j , W_j >$$

where $A_j = (a_{j1}, a_{j2}, ..., a_{jm})$ is the set of single actions, united in this set, $T_j = (t_{j1}, t_{j2}, ..., t_{jk})$ is the set of trails, where this set of actions was found, W_i is the weight, assigned during the analysis.

The weight is calculated in the following way:

$$W_g' = \sum_k w_k * q_k^g , q_k \in A_g , q_k^g = \frac{|T_g|}{|Tsa_k|}$$

Table 1
Recognized Actions

Data Type	**Example**	**Recognized Action (A)**	**Value (V)**
String	“Vienna” 7 “VIENNA”	Upper case	
String	“Sunday” 7 ”sunday”	Lower case	
String	“london” 7 “London”	First capital letter	
String	“fast food “ 7 “fast food“	Trim spaces	
String	“xpos” 7 “ndef”, “ypos” 7 “ndef”	Convert to constant	“ndef”
number	5.13 7 -5.13	Change of sign	
Number	57 10 10.5 7 15.5	Absolute change	5
Number	57 10 10 7 20	Relative change	2
Date	12.05.2010 7 12.05.2012	Adding of years	2
Date	12.05.2010 7 12.08.2010	Adding of months	3
Date	12.05.2010 7 02.06.2010	Adding of days	21
Date	15.05.2010 7 17.05.2010	Next working day	
List	March 7 January	First element of the list	
List	April 7 July May 7 August	Absolute change	3

4.2.2 Recognizing sets of actions

1. A new action is found
$\exists Ma_j =< A_j, T_j, W_j >, Sa_i =< a_i, Tsa_i, w_i >, a_i \in A_j$
=> A new set record is created
$Ma_g =< a_i, Tsa_i, w_i >$
=> Records with a subset of the trails $Ma_s =< a_i, T_s, W_s >, T_s \in Tsa_i$ are removed (if exist).

2. Coincidence of actions
$\exists Ma_j =< A_j, T_j, W_j >, Sa_i =< a_i, t_i, w_i >, A_j = (a_i)$
=> The existing record is modified:
$Ma_j =< A_j, T_j + t_i, W'_j >$

3. Coincidence of the trails, existing action
$\exists Ma_j =< A_j, T_j, W_j >, Sa_i =< a_i, T_j, w_i >, a_i \in A_j$
=> The weight is recalculated
$Ma_g =< A_g, T_g, W'_g >$

4. Coincidence of the trails, new action
$\exists Ma_j =< A_j, T_j, W_j >, Sa_i =< a_i, T_j, w_i >, a_i \notin A_j$
=> The existing record is modified
$Ma_j =< A_j \cup a_i, T_j, W'_j >$

5. Partial coincidence
$\exists Ma_j =< A_j, T_j, W_j >, Sa_i =< a_i, Tsa_i, w_i >, a_i \in A_j,$
$T_j = Tsa_i \cup Toff_j, Toff_j \neq 0$
=> A new set record is created or
$Ma_g =< A_j + a_i, Tsa_i, W'_g >$
=> The weight is recalculated (if the record exists already)
$Ma_g =< A_g, T_g, W'_g >$

6. Partial joining
$\exists Ma_j =< A_j, T_j, W_j >, Sa_i =< a_i, Tsa_i >, a_i \in A_j,$
$Tsa_i = T_j \cup Tsout_i, Tsout_i \neq 0$
=> A new set record is created (if it doesn’t exist)
$Ma_g =< A_g, Tsout_i, W'_g >$
where $A_g \in A_j, \forall a_m \in A_g, Tsa_m \in Tsa_i$
=> The weight is recalculated (if the record exists already)
$Ma_g =< A_g, T_g, W'_g >$

7. Adding of new action
$\exists Ma_j =< A_j, T_j, W_j >, Sa_i =< a_i, Tsa_i >, a_i \notin A_j,$
$Tsa_i = T_j \cup Tsout_i, Tsout_i \neq 0$
=>A new set record $Ma_g =< a_i, Tsa_i, w_i >$ is created, where
$A_g \in A_j, \forall a_m \in A_g, Tsa_m \in Tsa_i$
=> Adding the new action to the existing record
$Ma_j =< A_j + a_i, T_j, W_j >$
=> Recalculation of the weight of the both records
$Ma_j =< A_j, T_j, W'_j >, Ma_g =< A_g, T_g, W'_g >$

4.3 Data Representation in CLIPS

The discussed approach was realized as a set of rules in CLIPS. The templates for definition of the control ([14]), trail, recognized action and set of actions are given below:

```
;;----------- defines data for a control -------------------------------
(deftemplate  control_template
"P_Control"
(slot ID                (type INTEGER)     (default ?NONE))
(slot groupID           (type INTEGER)     (default ?NONE))
(slot position          (type INTEGER)     (default ?DERIVE))
(slot style             (type STRING)      (default ?NONE))
(slot type              (type STRING)      (default ?NONE))
(slot contolType        (type STRING)      (default ?NONE))
(slot accessRule        (type STRING)      (default ?DERIVE))
(slot dataBinding       (type STRING)      (default ?DERIVE))
(slot validatorType     (type STRING)      (default ?DERIVE))
(slot labelID           (type STRING)      (default ?DERIVE))
(slot label             (type STRING)      (default ?DERIVE))
(slot convertorType     (type STRING)      (default ?DERIVE))
(slot labelStyle        (type STRING)      (default ?NONE))
(slot addPosition       (type INTEGER)     (default ?DERIVE)) )
;;----------- defines data for a single action (user trails) ----------
(deftemplate  trail_template
"P_Trail"
(slot trailID           (type INTEGER)     (default ?NONE))
 (slot eventID          (type INTEGER)     (default ?DERIVE))
 (slot controlID        (type INTEGER)     (default ?DERIVE))
(slot controlName       (type STRING)      (default ?NONE))
(slot controlType       (type STRING)      (default ?DERIVE))
(slot oldValue          (type STRING)      (default ?DERIVE))
(slot newValue          (type STRING)      (default ?DERIVE)) )
;;----------- describes one action found among the trail_template facts --
(deftemplate action_r
"P_Action"
(slot actionID          (type STRING)      (default ""))
(slot controlID         (type INTEGER)     (default ?DERIVE))
(slot controlType       (type STRING)      (default ?DERIVE))
(slot type  (type STRING)          (default  "STRING"))
 (slot changeType       (type STRING)      (default  "UNDEFINED"))
(slot changeValue       (type STRING)      (default " "))
(slot weight            (type FLOAT)       (default 0.000000))
(multislot trailsIn     (type INTEGER)     (default ?DERIVE)))
;;----------- describes an action, consisting of several actions -----
(deftemplate  action_set
"P_MultiAction"
(slot actionID          (type STRING)      (default ""))
 (slot weight           (type FLOAT)       (default 0.000000))
(multislot actionsList (type INTEGER)      (default ?DERIVE))
(multislot trailsList   (type INTEGER)     (default ?DERIVE)))
```

4.4 Dynamic Modification

Once the module has information concerning actions and action sets discovered, the GUI changes can take place. A CLIPS rule determines the action sets with the highest weight. Then it automatically creates buttons for the 3 action sets with the highest weight. The buttons are displayed in a special group below the main frame. The buttons are defined as facts and they are added to the GUI description before rendering. For each button a corresponding rule is automatically created, which is responsible for performing the data modification accordingly the recognized pattern.

If the user decides to press one of the buttons, the function loads the data objects back to CLIPS. The new rules take the data according the data binding information, kept in the control_template facts (fig. 3) and performs the set of changes instead the user, saves the modified data back to the data objects.

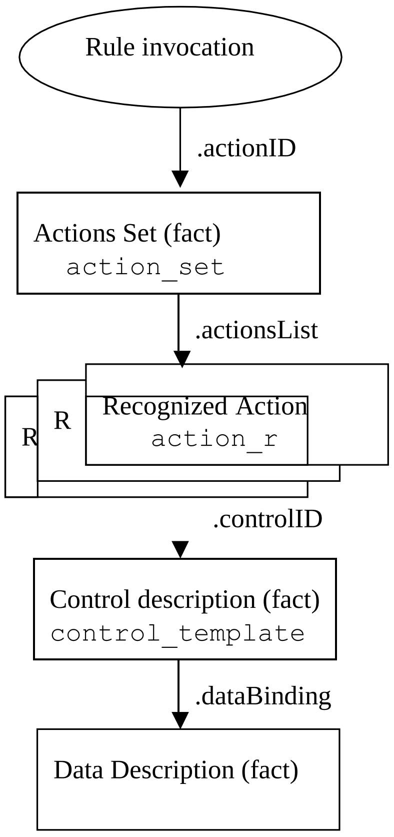

Figure 3: Data flow in dynamic data modification

5. Experimental Results

A Web application, built in Java, was developed as a test environment for the research [14]. It is working with financial instruments information (contracts, conventions, cashflow, etc.).

The application loads the descriptions in and converts them to a hierarchical structure of objects. The descriptions of the user activity and GUI definition are converted to facts and loaded into CLIPS engine. The discussed approach was tested with sample data. The performed actions were observed and saved under user ID.

Below a part of the facts, created from the collected data for the user activity, is shown.

```
f-1(trail_template (trailID 112)(eventID 10)(controlID 445)
(controlName "EvalDate")(controlType "DATE_TIME") (oldValue
"20090424") (newValue "20091024"))
f-2(trail_template (trailID 112)(eventID 11)(controlID 446)
```

(controlName "Action")(controlType "EDIT")(oldValue "Control")(newValue "CONTROL"))
f-3(trail_template (trailID 112)(eventID 12) (controlID 447)(controlName "Type")(controlType "EDIT")(oldValue "grid")(newValue "Grid"))
f-4(trail_template (trailID 112)(eventID 13) (controlID 448)(controlName "Amount") (controlType "EDIT")(oldValue "0.89")(newValue "2.89"))
f-5(trail_template (trailID 113)(eventID 10) (controlID 445)(controlName "EvalDate") (controlType "DATE_TIME")(oldValue "20100121")(newValue "20100721"))
f-6(trail_template (trailID 113)(eventID 11) (controlID 446)(controlName "Action")(controlType "EDIT")(oldValue "Moon")(newValue "MOON"))
f-7(trail_template (trailID 113)(eventID 12) (controlID 448)(controlName "Amount")(controlType "EDIT")(oldValue "0.70")(newValue "2.70"))
f-8(trail_template (trailID 114)(eventID 10) (controlID 445)(controlName "EvalDate") (controlType "DATE_TIME")(oldValue "20071111") (newValue "20080511"))
f-9(trail_template (trailID 114)(eventID 11) (controlID 446)(controlName "Action")(controlType "EDIT")(oldValue "wheel")(newValue "WHEEL"))
f-10(trail_template (trailID 114)(eventID 12) (controlID 448)(controlName "Amount")(controlType "EDIT")(oldValue "2.70")(newValue "4.70"))

A set of rules searches for the patterns, shown in table 1. The following repeatable actions were found:

f-67(action_r (actionID "**S_ACT_ID_2**")(controlID 448)(controlType "EDIT")(type "NUMBER")(changeType "RELATIVE_CHANGE")(changeValue 0.574)(weight 1.0)(trailsIn 114))
f-69(action_r (actionID "**S_ACT_ID_1**")(controlID 448)(controlType "EDIT")(type "STRING")(changeType "CONSTANT")(changeValue "4.70")(weight 1.0)(trailsIn 114))
f-75(action_r (actionID "**S_ACT_ID_5**")(controlID 446)(controlType "EDIT")(type "STRING")(changeType "CONSTANT")(changeValue "WHEEL")(weight 1.0)(trailsIn 114))
f-77(action_r (actionID "**S_ACT_ID_9**")(controlID 448)(controlType "EDIT")(type "NUMBER")(changeType "RELATIVE_CHANGE")(changeValue 0.259)(weight 1.0)(trailsIn 113))
f-79(action_r (actionID "**S_ACT_ID_8**")(controlID 448)(controlType "EDIT")(type "STRING")(changeType "CONSTANT")(changeValue "2.70")(weight 1.0)(trailsIn 113))
f-85 (action_r (actionID "**S_ACT_ID_12**")(controlID 446)(controlType "EDIT")(type "STRING")(changeType "CONSTANT")(changeValue "MOON")(weight 1.0)(trailsIn 113))
f-138(action_r (actionID "**S_ACT_ID_3**")(controlID 448)(controlType "EDIT")(type "NUMBER")(changeType "ABSOLUTE_CHANGE")(changeValue -2.0)(weight 3.0)(trailsIn 114 113 112))
f-143(action_r (actionID "**S_ACT_ID_16**")(controlID 448)(controlType "EDIT")(type "NUMBER")(changeType "RELATIVE_CHANGE")(changeValue 0.307)(weight 1.0)(trailsIn 112))
f-149(action_r (actionID "**S_ACT_ID_15**")(controlID 448)(controlType "EDIT")(type "STRING")(changeType "CONSTANT")(changeValue "2.89")(weight 1.0)(trailsIn 112))
f-159(action_r (actionID "**S_ACT_ID_19**")(controlID 447)(controlType "EDIT")(type "STRING")(changeType "CONSTANT")(changeValue "Grid")(weight 1.0)(trailsIn 112))
f-169(action_r (actionID "**S_ACT_ID_18**")(controlID 447)(controlType "EDIT")(type "STRING")(changeType "FIRST_CAP")(changeValue "")(weight 1.0)(trailsIn 112))
f-187(action_r (actionID "**S_ACT_ID_21**")(controlID 446)(controlType "EDIT")(type "STRING")(changeType "CONSTANT")(changeValue "CONTROL")(weight 1.0)(trailsIn 112))
f-236(action_r (actionID "**S_ACT_ID_4**")(controlID 446)(controlType "EDIT")(type "STRING")(changeType "UPPER_CASE")(changeValue "")(weight 3.0)(trailsIn 114 113 112))
f-100(action_r (actionID "**S_ACT_ID_7**")(controlID 445)(controlType "DATE_TIME")(type "DATE")(changeType "PLUS_DAYS")(changeValue 181)(weight 2.0)(trailsIn 114 113))
f-285(action_r (actionID "**S_ACT_ID_23**")(controlID 445)(controlType "DATE_TIME")(type "DATE")(changeType "PLUS_DAYS")(changeValue 183)(weight 1.0)(trailsIn 112))
f-328(action_r (actionID "**S_ACT_ID_6**")(controlID 445)(controlType "DATE_TIME")(type "DATE")(changeType "PLUS_MONTHS")(changeValue 6)(weight 3.0)(trailsIn 114 113 112))

In the second stage the rules, which are applied over the repeatable action facts, create facts for action sets determined according the algorithm, discussed in p. 4.2.2. The rule action-rule-generation, shown below,

```
;;- finds the multi action facts with highest weight and generates rules ---
(defrule  action-rule-generation
        ?trails-control<-(trail_counter  (ruleGeneration  TRUE))
=>
        ; step 1: to find the highest rank multi-action facts
        (bind  ?highest-rank-action-sets  (find-actions-with-max-weight))
        ; step 2: to delete previous rules (if there are any)
        (bind ?all-rules (get-defrule-list))
        (loop-for-count (?i 1 ?*art-ai-rules*) do
                (bind ?curr-rule (create$ (string-to-field (create-rule-id ?i))))
                (if (= 2 (compare-two-lists ?curr-rule ?all-rules)) then
                        (undefrule (nth$ 1 ?curr-rule))
                )
        )
        ; step 3: to create the controls
        (create-controls  ?highest-rank-action-sets)
        ; step 4: to generate new rules
        (create-rules  ?highest-rank-action-sets)
        ; step 4: to reset the flag
        (modify ?trails-control (ruleGeneration FALSE))
)
```

determines the following action sets with highest weight:

f-331 (action_set (actionID "M_ACT_ID_1051")(**weight 9.0)** **(**actionsList "S_ACT_ID_3" "S_ACT_ID_4" "S_ACT_ID_6")(trailsList 112 113 114))
f-246 (action_set (actionID "M_ACT_ID_1006")(**weight 6.0)** **(**actionsList "S_ACT_ID_7" "S_ACT_ID_3" "S_ACT_ID_4")(trailsList 113 114))
f-323 (action_set (actionID "M_ACT_ID_1052")(**weight 6.0)** **(**actionsList "S_ACT_ID_4" "S_ACT_ID_6")(trailsList 114 113 112))

The 3 new facts, describing the suggested actions were created. The list of the final 3 facts (M_ACT_ID_1051, M_ACT_ID_1006, M_ACT_ID_1052) is used for generation of the rules, responsible for performing identical modifications on the data from the data objects if the user decides to press one of the new buttons.

6. Conclusion

The tests of the approach were encouraging. The rules were able to locate repeatable actions, the transfer of the data from the application to the CLIPS engine and vice versa are working properly.

The rule-based approach provides an easy-to-extend framework. New logic (also project-dependent one) can be added by adding new rules.

With increasing of the amount of the data for the user activity there some performance issues could be expected. To solve this problem the following steps are planned:

scheduling the analysis of the user activities as an overnight task, saving the collected information for the discovered actions and using it on later stage, thus eliminating the need for complete recalculation.

Currently the information, collected for the same user is used for deducing concerning his activities. The isolated actions from other users could be used as well.

The automatically created buttons must have reasonable names, created from the rules.

Comprehensive tests are still to be performed.

The application has to be able to work properly also with empty knowledge base. The idea, suggested in [21], that the UI designer can provide few typical user traces as samples is planned to be used ([14]).

The other directions of implementation, discussed in p. 2.1., are to be investigated as well. The *Content personalization* direction is determined as the next area of interest.

References

[1] A. Jameson, Adaptive interfaces and agents, *Human-Computer Interaction Handbook* (Erlbaum, 2003)

[2] B. Price, R. Greiner, G. Häubl & A. Flatt, Automatic construction of personalized customer interfaces, *Proc. 11th international conference on Intelligent user interfaces*, 2006, 250-257.

[3] H. Schaumburg, Computers as tools or as social actors? The users' perspective on anthropomorphic agents, *International Journal on Intelligent Cooperative Information Systems*, *10*(1-2),2001, 217-234

[4] E. Horvitz, J. Breese, D. Heckerman, D. Hovel & K. Rommelse, The Lumiere Project: Bayesian User Modelling for Inferring the Goals and Needs of Software Users, *Proc. 14th Conference on Uncertainty in Artificial Intelligence*, 1990, 256 –265.

[5] R. Segal, J. Kephart, Incremental learning in SwiftFile, *Proc. 17th International Conference on Machine Learning*, 2000, 863-870.

[6] A. Paramythis, A. Savidis & C. Stephanidis, AVANTI: a universally accessible web browser. *Proc. Human-Computer Interaction (HCI'2001)*, 2001, 91-95.

[7] J. Macías, P. Castells, Dynamic web page authoring by example using ontology-based domain knowledge, *Proc. 8th international conference on Intelligent user interfaces IUI'03*,2003, 133-140.

[8] S. Trewin, H. Pain, A model of keyboard configuration requirements. *Proc. 3d International ACM Conference on Assistive Technologies*, 1998, 173-181

[9] J. Goecks, J. Shavlik, Learning users' interests by unobtrusively observing their normal behaviour, *Proc. 5th International Conference on Intelligent User Interfaces IUI 2000*, 2000, 129-132.

[10] S. Carberry, Techniques for plan recognition, *User Modelling and User-Adapted Interaction, 11*(1-2), 2001, 31-48.

[11] E. Rich, User modelling via Stereotypes, *Readings in Intelligent User Interfaces* (San Francisco, Morgan Kaufmann, 1998), 329-341.

[12] J. Fink, A. Kobsa, A review and analysis of commercial user modelling servers for personalization on the world wide web. *User Modelling and User-Adapted Interaction, vol 10*, 2000, 209 – 249

[13] P. Paskalev, I. Serafimova, Runtime generation of a user interface, described in a database, *Proc. International Conference CompSysTech'09*, 2009, VI.2.

[14] P. Paskalev, Rule based GUI modification and adaptation, *Proc. International Conference CompSysTech'09*, 2009, VI.3.

[15] P. Paskalev, A. Antonov, Intelligent application for duplication detection, *Proc. International Conference CompSysTech 2006*, 2006, IIIA.27.

[16] Y. Arens, L. Miller & N. Sondheimer, Presentation design using an integrated knowledge base, *Readings in Intelligent User Interfaces* (San Francisco, Morgan Kaufmann, 1998), 131-139.

[17] J. Giarratano, *CLIPS User's Guide Version 6.20*, 2002

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

[19] D. Koelle, Intelligent user interfaces, *http://web.cs.wpi.edu/Research/airg/IntInt/intint-outline.html*, 1996.

[20] A. Cooper, R. Reimann & D. Cronin, *About Face. The essentials of Interaction Design* (3 ed., Wiley Publishing, 2007)

[21] K. Gajos, D. Weld, SUPPLE: Automatically Generating User Interfaces, *Proc. 9th international conference on Intelligent User Interfaces*, 2004, 93-100.

[22] S. Roth, J. Kolojejchick, J. Mattis & J. Goldstein, Interactive graphic design using automatic presentation knowledge, , *Readings in Intelligent User Interfaces* (San Francisco, Morgan Kaufmann, 1998), 237-242.

[23] W. Mackay, Triggers and barriers to customizing software, *Proc. SIGCHI conference on Human factors in computing systems (ACM CHI'91)*, 1991, 153-160.

[24] A. Bunt, C. Conati & J. McGrenere, What Role Can Adaptive Support Play in an Adaptable System? *Proc. 9th international conference on Intelligent User Interfaces IUI2004*, 2004, 117-124.