Customization of Web Applications through an Intelligent Environment Exploiting Logical Interface Descriptions

José A. Macías[1], Fabio Paternò
ISTI-CNR
Via G. Moruzzi 1
56124 Pisa, Italy
{Jose.Macias, Fabio.Paterno}@isti.cnr.it

Abstract

Customization of Web-based applications is often considered a designer skill rather than an end-user need. However, there is an ongoing shift to end-user-centred technology, and even users with poor or no skill in Web-based languages may feel the need to customize Web applications according to their preferences. Although Web authoring environments have an increasing number of features, the challenge of providing end-users with the ability to easily customize entire Web applications still remains unsolved. In this paper, we propose an intelligent approach to customizing Web-based applications. Customizations rules are automatically inferred by the system from changes that users supply as examples. They remain as long-term knowledge that can be applied to support future interactions, thus minimizing the amount of authoring that end-users need to do for this purpose. In order to better understand the implications of the user's modifications, they are analysed using the logical descriptions of the corresponding Web pages.

Keywords

Human-Computer Interaction; End-User Development; Intelligent User Interfaces; Programming by Example; Model-Based User Interfaces; Web-based Nomadic Applications.

1. Introduction

As software products grow in terms of expressivity, there is also a growing need to allow people to customize, configure and also create their own software artefacts in order to accomplish their daily tasks properly. This is important, for example, for professionals such as engineers, scientists and freelance professionals having concrete domain skills but lacking programming abilities. Further support is needed in order to provide non-programmer professionals with easy-to-use mechanisms to customize software artefacts, avoiding the need for them to learn programming and specification languages that are usually deemed to be irrelevant to their daily work. In our view, customization can be regarded as the modification of interactive software in order to better match the user preferences.

Generally, the explicit customization of software applications is considered a cumbersome task that most non-computer-skilled end-users cannot afford. The

[1] Permanent Address: EPS, Universidad Autónoma de Madrid. Ctra. de Colmenar, Km. 11. 28049 Madrid, Spain. E-mail: j.macias@uam.es

complexity of programming and specification languages discourages users even from attempting software customization. Although most applications do not provide much support for customization, some of them allow users to adapt partial aspects of the application to their own needs by selecting predefined options. However, this is clearly not enough. Desktop applications are usually complex and implemented in structured programming languages. This has traditionally made it difficult to provide easy-to-customize end-user approaches for them. Thus, the few existing approaches to this respect have been mainly focused on some domain-dependent support. Further, the traditional desktop customization process cannot be applied straightforward to Web environments, since they are based on an underlying specific mark-up language (i.e. XHTML). On the other hand, XHTML is easier to access and manipulate and enables the possibility of end-user development, in which users can customize their applications. However, customization of interactive Web applications still requires considerable skill in programming and Web technology. Some preliminary studies indicate that these limitations in end-user Web development activities are not due to lack of interest but rather to the difficulties inherent in interactive Web development (Rode et al., 2006). Commercial Web development tools already offer support for high-level functionality, but most of these tools are not aimed at non-programmers.

Providing users with real customization facilities is not yet as widespread as one would expect. Most existing approaches provide little support to end-users, and the ease of customization of commercial applications leaves much to be desired. However, some researchers have devoted considerable effort to bring design closer to users. This involves End-User Development (EUD) research (Lieberman et al., 2006), where the main concern is to enable users to easily modify and create software artefacts. Programming by Example (Cypher, 1993; Lieberman, 2001) is one of the more relevant approaches in EUD, which aims at obtaining a satisfactory trade-off between ease-of-specification and expressiveness. Programming by Example has the potential to allow users to customize their applications. Rather than writing a program in a programming language to automate a task, users simply demonstrate how to perform it.

Our research is aimed at leveraging these problems by providing automatic mechanisms to allow customization tasks easily. Therefore, the main problem we address here is how to provide intelligent automatic support for customising Web applications even for non-computer-skilled end-users. To face such a challenge, we leverage Model-Based User Interfaces Design (MBUID) approaches (Paternò, 1999) combined with customization techniques (Macías and Castells, 2004). The overall goal is natural development (Berti et al., 2006), which implies that people should be able to create or modify applications by working through familiar and immediately understandable representations to express relevant concepts. In this respect, our main contribution exploits Model-Based User Interface Design (Szekely, 1996) and End-User Development research, combining them by means of an intelligent environment that can infer meaningful information from the user's modifications. Our approach is based on an expert system where the knowledge is built up progressively, increasing in every user session (i.e. evolutionary approach). According to the MBUID paradigm, we consider the different conceptual levels in which a user interface can be specified. We obtain a logical user interface description (UID), which provides a description of the main conceptual elements associated with the user interface without considering implementation details**,** from the reverse engineering transformation of XHTML pages. Automatically, our intelligent environment compares the logical description of the modified page with the original ones, reasoning about such changes by means of rules.

Therefore, the user only has to provide the system with examples of what s/he desires to modify, and the system identifies *customization rules* automatically by analyzing the user changes. To this end, our intelligent rule-based system exploits JESS, the Rule Engine for Java Platforms (JESS, 2006). Jess includes an enhanced version of the Rete algorithm (Forgy, 1982), an efficient pattern matching algorithm for implementing production rule systems. We use the Rete algorithm together with information regarding the page context in which the user change occurred, in order to reduce the implicit ambiguity in drawing inference from similar user changes. The possible page contexts are identified by analyzing the abstract descriptions of the corresponding pages. They are used to identify different user changes in similar situations, and so obtain more general domain-independent customization rules (Macías and Castells, 2005).

This paper is structured as follows. After the related work in Section 2, we describe the general approach proposed. Section 4 provides further detail and describes how the intelligent mechanism is structured. Next, we explain the process of extracting meaningful information from user customizations. Section 6 reports on a user test that has been carried out to provide empirical feedback on the approach proposed. The test results are useful to understand the feasibility of reducing the complexity of Web application customization. Lastly, Section 7 draws some conclusions and provides indications for future work.

2. Related Work

Traditionally, Programming by Example (PBE) research has adopted rule-based systems mostly due to the execution speed and simplicity they provide, compared with other complex machine-learning algorithms that often have a high error rate and low generalization in real-time interaction with users.

Some early PBE systems used rules to infer user intentions. For instance, Eager (Cyper, 1993) uses LISP implemented rules to complete specific repetitive tasks on behalf of the user. Eager is considered a predictive interface since it detects two consecutive occurrences of a repetitive task in a sequence of user actions. Eager automatically infers patterns by observing user actions. Typically, the actions that Eager infers are mostly based on textual operations that users carry out on the Apple-Macintosh HyperCard application. By contrast, in our system users are not monitored during the interaction, and we provide them with the freedom to use any authoring environment to modify Web applications. Moreover, in addition to syntactic changes our system also deals with semantics corresponding to changes in the logical structure (i.e. relations, grouping, hierarchical, navigational and other changes) that can be generalized from session to session, as the system is responsible for distinguishing between pending and permanent customization preferences in any Web application.

Inference Bear (Frank et al., 1995) is another PBE system that infers design choices from user-generated snapshots. Inference Bear generates a custom user interface by observing the behaviour of the interface designer. Like Eager, Inference Bear is based on a specific application- i.e. a graphical design tool that makes it possible to infer geometrical relationships from user actions. In our approach, we provide more general inference by supporting both semantic and syntactic changes, independently of the authoring tool used. We mean for semantic changes those that modify the logical structure of the user interface or the tasks supported by the interactive application. User preferences are captured by our system and they can be used later on to obtain further information about customization that will be updated and refined in an evolutionary way from session to session.

Some early Myers's tools such as Peridot (Myers, 1998) supported a rule-based approach. Peridot is more oriented to supporting user interface design. It uses about fifty hand-coded Interlisp-D rules to infer the graphical layout of the objects from the examples. The system allows the interface designer to draw a picture of what the user interface should look like using a special drawing package. This way, the system infers user interface groupings, geometrical dependences and spatial relationships, taking also into account the user's input. This type of system is able to generalize only limited forms of behaviours. Additionally, it is mostly focused on static knowledge and can be considered domain-dependent. By contrast, our system proposes a dynamic knowledge approach together with an application-independent intelligent environment, in which a complete rule structure is proposed in order to consider different kinds of conceptual knowledge that can be updated from time to time through an evolutionary approach. Our system can be considered domain-independent since it is able to process any application written with XHTML. Indeed, the semantic structure is extracted and processed independently of the content type.

More recently, AgentSheets (Repenning and Ioannidou, 2006) is an example of a commercial EUD approach for building intelligent interfaces. AgentSheets is a simulation environment that allows users to create advance simulation scenarios by defining intelligent agents and behaviour separately. AgentSheets combines PBE with graphical rewrite rules into an end-user programming paradigm. Graphical rewrite rules are powerful languages to express the concept of change in a visual representation. Like AgentSheets, our approach applies semantic rules for dealing with high-level behaviour. As pointed out by the AgentSheets' authors, a first step toward creating more usable and reusable rewrite rules is to move from syntactic rewrite rules to semantic ones, including semantic meta-information. The lack of semantics not only makes reuse difficult, but also involves significant problems for building new behaviours from scratch, reducing dramatically the scalability of a PBE approach as well. In this respect, in our approach we consider different levels of knowledge and behaviour. We accomplish this by dividing rules and facts into different conceptual levels that help to automatically achieve in-depth analysis and accurately infer the user's preferences through an evolutionary approach, in which the knowledge increases proportionally with the number of user sessions considered.

The use of semantic knowledge has been a main concern in building intelligent inference systems. Lieberman's work on Common Sense Reasoning (Lieberman et al., 2004) provides some evidence by using high-level knowledge bases to carry out automatic reasoning. In that work, mostly related to natural language-based interaction, the system tries to infer meaningful user definitions by using natural language (Lieberman and Liu, 2006). The authors assume that syntactical definitions are vague and imprecise, and so they need to be disambiguated using a semantic layer in the form of an ontology to obtain high-level information. In our approach, we deal with logical user interface descriptions to infer more specific user preferences. We reduce the ambiguity in analysing user changes by using both a more accurate inference algorithm and semantic knowledge extracted from the modified Web-based interface.

Another related work is DESK (Macías et al., 2006), which uses domain knowledge for characterizing changes from a dynamically generated interface, also making minimal assumptions about the end-user's skills on programming and specification languages. DESK uses the PEGASUS specification based on domain ontologies in order to specify explicit knowledge of both presentation and domain information separately. DESK is based on a client-server architecture that comprises two different applications. The client side is a front-end what-you-see-is-what-you-get (WYSIWYG) authoring tool, whereas the server side is a back-end application that infers and carries out the changes the user makes to the Web interface. DESK's front-end tracks and records information about the user's actions by building up a XML monitoring model. This information is sent to the back-end application, which in turn processes the monitoring model and applies different heuristics by using domain knowledge. DESK deals with modifications in the HTML code that are later processed to obtain meaningful information by means of fixed heuristics. Our approach improves DESK's mechanisms by identifying customization rules automatically, comparing logical descriptions of the original and modified interfaces and with no need to have a specific authoring client application. This allows our environment to achieve domain independence. The logical user interface descriptions are specified in TERESA XML (Berti et al., 2004). We exploit the information provided by the logical interface description to obtain semantic information. Besides, the knowledge management is improved by defining different levels of knowledge that are applied to better characterize customizations and update the expert system for future inference.

More recently, Chickenfoot (Bolin et al., 2005) provides a programming environment embedded in the Firefox Web browser for automation and customization of Web pages. Our approach differs from it in many respects: it is not linked to any particular Web browser and, since we manage the customization rules at the server side, we can support customization even when different client devices (and browser) are used to access the same Web application. Additionally, in Chickenfoot the user is requested to introduce script code, whereas in our system we avoid users to program any code as changes to Web pages are made by WYSWYG tools and are automatically detected and processed by our system independently of the specific authoring tool used.

3. Our Approach

Our environment aims to support Web interfaces for different platforms in order to allow users to access the application using one device from a set of different existing platforms (desktop computer, mobile, PDA). At any time, the user can provide the system with examples obtained by modifying the generated Web pages with any authoring environment. Reverse engineering techniques are applied in order to obtain logical descriptions of the modified Web pages, which are useful to analyse high-level information about the user's modifications.

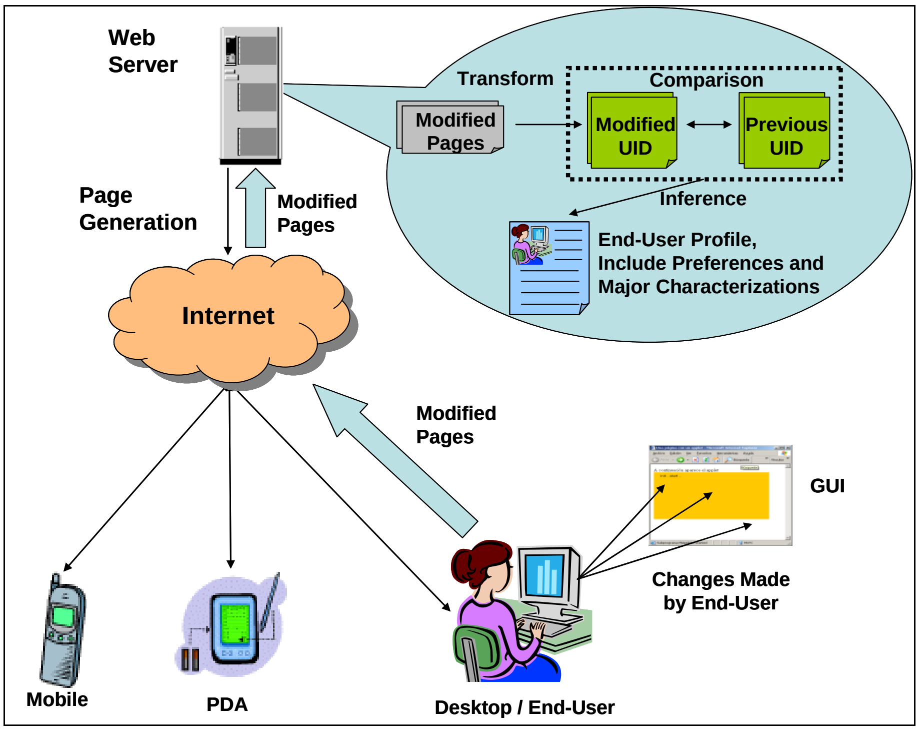


Figure 1. Framework for inferring user preferences in customizing nomadic Web-based applications.

In particular, our approach supports the following steps (see Figure 1):

1) The end-user navigates through a desktop Web application and, at some point, s/he decides to modify something by using standard WYSIWYG Web authoring tools such as Microsoft FrontPage, Macromedia Dreamweaver and so on, where users can make sample changes in order to provide indications of their preferences. Such changes comprise deletion, addition, insertion, substitution, presentation-element re-arrangement, style-effect application, property changes, and so on.

2) Once the user has finished the changes, s/he sends the modified pages to the server, by using a specific Web application in which s/he first needs to login (see Figure 8).

3) The server receives the Web page and then starts the inference process to identify the user's preferences.

    a. First, the server transforms the modified page into a logical description stored in a XML file, using the reverse mechanism developed by our group (Bandelloni et al., 2007). The file contains the user interface specification of the page (i.e. UID on top-right corner of Figure 1), in terms of language-independent elements.

    b. Then, the system compares the logical description corresponding to the modified page with the logical description of the previously generated page.

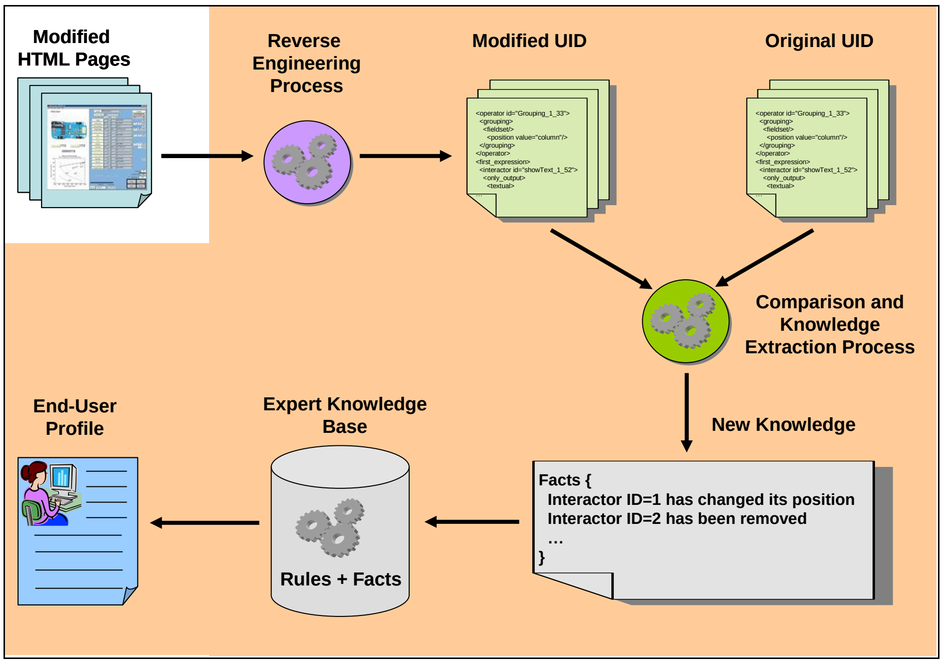
    c. During the comparison process, the system generates high-level information to feed the expert system and identify general user preferences. This intelligent mechanism will be explained in detail later on.

d. At the end of the process, the system builds an End-User Profile taking into account all this high-level information inferred, as well as other previously generated, and containing customization preferences.

4) The End-User Profile is then used to regenerate the Web pages, taking into account the user's preferences. The system stores an End-User Profile for each user and platform.

The End-User Profile must also contain the logical specification of the page modified by user. The information included in the Profile will be updated every time the user sends a modified Web page to the server. Thus, the End-User Profile is updated with new inference information and the interface logical descriptions are updated accordingly.

The most relevant information in the End-User Profile is the set of user interface customization rules. Such rules are inferred from the comparison of the logical descriptions and aim to reflect the knowledge acquired by the system from analysis of the user's changes. Later on, the rules are used in the generation of the Web pages, customizing the Web presentation and navigation depending on the inferred preferences. All this explicit knowledge can be modelled by means of a knowledge base (i.e. an expert system) containing facts and sets of rules to be applied when new information about user modifications is identified.



```
<operator id="Grouping_1_33">
  <grouping>
    <fieldset/>
      <position value="column"/>
  </grouping>
</operator>
<first_expression>
  <interactor id="showText_1_52">
    <only_output>
      <textual>
…
```

```
<operator id="Grouping_1_33">
  <grouping>
    <fieldset/>
      <position value="column"/>
  </grouping>
</operator>
<first_expression>
  <interactor id="showText_1_52">
    <only_output>
      <textual>
…
```

```
Facts {
  Interactor ID=1 has changed its position
  Interactor ID=2 has been removed
  …
}
```

Figure 2. The process of knowledge extraction from the user changes.

The intelligent approach is implemented by using an ad-hoc expert system, where the knowledge can be suitably modelled and where the inference takes place efficiently. More specifically, we are mostly interested in *production systems* to implement a *pattern recognition* approach by means of rule languages. Such languages provide a framework able to deal with facts and rules, as well as the capability to populate the knowledge base with new information from time to time (evolutionary approach).

In our approach, the facts represent the information coming from the user's modifications. This information is extracted by comparing the logical interface descriptions of the modified and the original page (see Figure 2). The rules are ad-hoc conditions used to extract information from the facts, that is, from the changes the user makes to the presentation and from other high-level information available in the expert knowledge base. Since all the user information is stored in a user model (the user profile), the rules will reflect not only the user's changes but information about the platform (Desktop, Mobile, and so on). This evolutionary approach, which continuously produces and modifies facts, aids the system to refine the user's preferences and extracts accurate information as interaction evolves.

In order to improve precision and accuracy in the inference process, we use Rete as an efficient pattern matching algorithm for implementing rule-based (expert) systems. It was originally designed by Forgy at Carnegie Mellon University. A Rete-based expert system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring on the left-hand-side of a rule. Each node has a memory of facts, which satisfy that pattern. As new facts are added or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or combination of facts satisfies all patterns of a given rule, a leaf node is reached and the corresponding rule is triggered. The Rete algorithm is designed to sacrifice memory for increased speed.

4. User Interface Modelling and Intelligent Processing

In our approach, we want to identify the user's preferences from the changes that s/he made to the Web pages. Furthermore, we want our environment to be able to take into account previous changes. Therefore, the idea was to develop an intelligent system capable of detecting changes through the analysis of the logical user interface specifications in TERESA XML (Berti et al., 2004). In this specification language, a user interface can be described at different abstraction levels. In particular, we considered the abstract and concrete levels. The abstract label is a logical description of the user interface independent of the platform. We mean for platform a group of devices that have similar interaction resources (i.e. desktop, PDA, mobile, etc.). By contrast, the concrete level is a platform-dependent description of a user interface, so it is a refinement for a specific platform. For example, at the abstract level we can have the specification of a selection object (without any indication whether the selection is performed in a graphical, vocal or gestural modality). A corresponding concrete element for the graphical platform would be a list or a radio-button or a pull-down menu, which are examples of objects that support selection in a graphical device. In both cases, the user interface is composed of interactors and composition operators indicating how to structure their composition.

While at the abstract level the various interactors are described with no reference to any particular interaction modality, at the concrete level the characterisation of each interactor depends on the type of platform and media available, with a number of attributes that define more concretely its appearance and behaviour. Examples of interactors at the abstract level are Description, Navigator, Text, Single Selection, etc., whereas examples of interactors at the concrete level are Image, Link, Text Field, Radio Button, List, etc.. There are different one to many relationships between interactors at the abstract and the concrete level (e.g. a navigator can be a text link, an image link, or a button), which indicate how an abstract interaction can be supported in a given platform at the concrete level.

The composition operators provide useful information in terms of how interactors are put together, the relation among them and the associated communication goal. At the abstract level, there are four different composition operators:

- Grouping: Indicates a set of interface elements logically connected to each other
- Ordering: Some kind of ordering among a set of elements can be highlighted
- Hierarchy: Different levels of importance can be defined among a set of elements
- Relation: Highlights a one-to-many relation among some elements, one element has some effects on a set of elements

By contrast, the concrete level indicates how each composition operator can be supported, for example a grouping can be obtained through a Fieldset (a rectangle including the grouped elements), or lining up the composed elements, and so on.

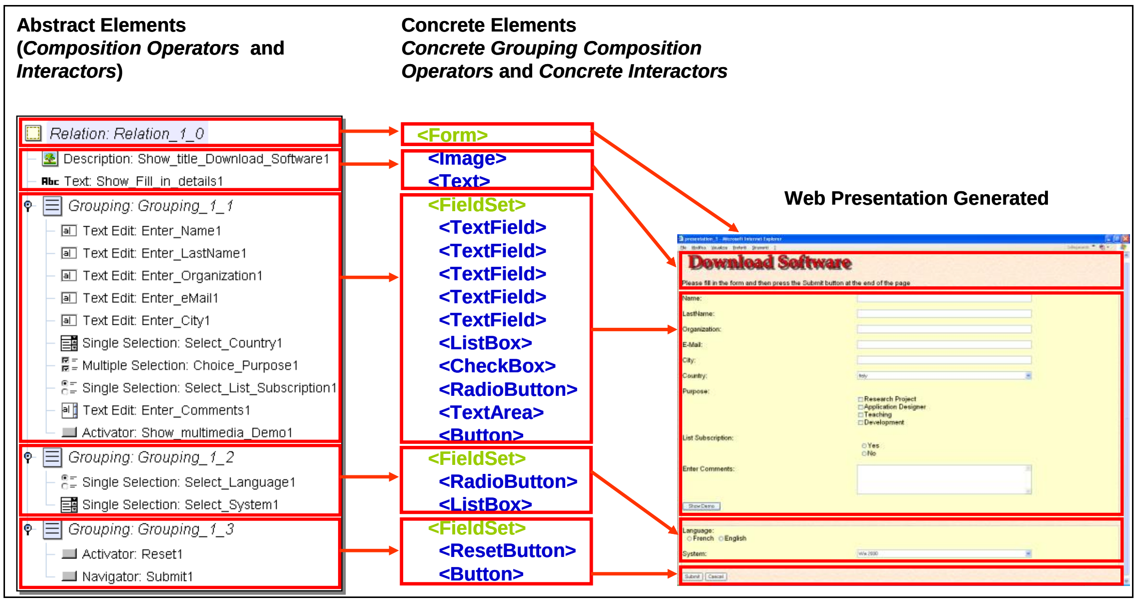

Abstract Elements
(Composition Operators and Interactors)
Concrete Elements
Concrete Grouping Composition Operators and Concrete Interactors
Web Presentation Generated
Relation: Relation_1_0
Description: Show_title_Download_Software1
Text: Show_Fill_in_details1
Grouping: Grouping_1_1
Text Edit: Enter_Name1
Text Edit: Enter_LastName1
Text Edit: Enter_Organization1
Text Edit: Enter_eMail1
Text Edit: Enter_City1
Single Selection: Select_Country1
Multiple Selection: Choice_Purpose1
Single Selection: Select_List_Subscription1
Text Edit: Enter_Comments1
Activator: Show_multimedia_Demo1
Grouping: Grouping_1_2
Single Selection: Select_Language1
Single Selection: Select_System1
Grouping: Grouping_1_3
Activator: Reset1
Navigator: Submit1
<Form>
<Image>
<Text>
<FieldSet>
<TextField>
<TextField>
<TextField>
<TextField>
<TextField>
<ListBox>
<CheckBox>
<RadioButton>
<TextArea>
<Button>
<FieldSet>
<RadioButton>
<ListBox>
<FieldSet>
<ResetButton>
<Button>
Download Software

Figure 3. Mappings between Web-page components and concrete and abstract elements that are extracted from a Web page.

Figure 3 shows an example of a generated Web form and the structure of its corresponding abstract and concrete specifications. The concrete information is represented by concrete interactors (such as Textfield, Button and so on) and concrete composition operators (such as Form and FieldSet). It is worth looking at the mappings between abstract elements (on the left) and page components. The abstract elements provide the intelligent environment with the conceptual description of the interface design. This information consists of the types of composition operators (such as *Relation*, *Grouping*,) and different kinds of interactors (such as *Description*, *Text Edit*, *Single Selection* and so on). This knowledge is useful for identifying presentation contexts when changes to a Web page are analysed.

Modifications affecting the concrete level provide syntactical knowledge, while those that have effects on the abstract level provide semantic knowledge because they imply changes to the actual purpose of the interface elements. We consider both kinds of modifications in order to construct a knowledge structure to feed the expert system with suitable facts and activate expert rules to produce user customizations efficiently. Additionally, we also want to consider both syntactic and semantic customization rules that may be fired more than once for different changes applied in the same page context. Semantic customization rules are used to deal with changes concerning interactors and composition operators at the abstract level. By contrast, syntactic customization rules are related to user preferences associated with user modifications that have an effect only on the concrete specification (e.g. changes concerning font size, colour, and other syntactical preferences). The knowledge obtained will be applied the next time that the application server generates the corresponding pages.

This conceptual separation helps to identify different kinds of rules as well as to build the knowledge progressively, according to the kind of fact that is produced and managed. In addition to rules, different sets of facts are generated at every level. In this sense, the relations among the different levels of knowledge can be represented as in Figure 4.

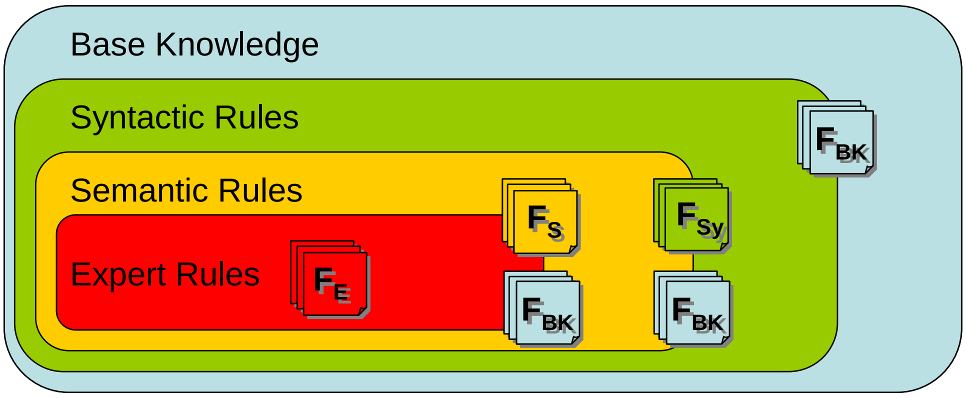


Figure 4. The four levels of our knowledge structure.

Regarding Figure 4, it is worth noting that there are dependences between the facts generated by each rule and the rules that process such facts. As we can see, base knowledge facts (F_{BK}) are processed by each kind of rule, whereas syntactic facts (F_{Sy}) are generated by the syntactic rules and processed only by the semantic ones. In contrast, the semantic facts (F_S) are generated by the semantic rules and processed by the expert ones. Moreover, expert rules generate expert facts (F_E) that reflect the meaningful information in terms of customization rules for future use. All the facts generated, as well as the rules, will remain as base-knowledge facts to be taken into account in later sessions.

The following steps are considered in defining the knowledge:

1) **Defining base knowledge** containing basic definition about user, platform and the previous knowledge inferred. This is the information that always remains in the expert system and is updated from session to session.

2) **Defining syntactic knowledge** that contains facts and rules triggered by syntactic modifications to presentation elements such as concrete interactors (for instance, in a graphical desktop system concrete interactors can be Radio Buttons, List Boxes, Textual Links, Buttons, Input Texts and so on) and concrete composition operators. The concrete composition operators implement the abstract operators (grouping, hierarchy, ordering and relation) through constructs such as Fieldset, Unordered List, Ordered List, Table, Form, and so on. An example of syntactical modification is when the value of an attribute of a

concrete interactor or a concrete composition operator is changed (e.g. when its colour, alignment, justification and so on are changed).

3) **Defining semantic knowledge** for dealing with semantic information by taking into account the syntactic information already created. The semantic level considers changes in the abstract (platform-independent) elements called interactors and composition operators. For instance, it identifies when an interactor is moved from a composition operator to another, or when it is deleted or removed, or also when the number of interactors in one possible composition (ordering, hierarchy relation or grouping) is changed. The semantic level also concerns presentation contexts, that is, high-level descriptions of the elements (and their relations) surrounding a change made in the graphical interface. Presentation contexts allow the creation of expert rules based on contextual information that can be applied more than once.

4) **Defining expert rules** for dealing with further semantic aspects and detecting user preferences. Expert rules utilize all the underlying information available in the expert system, and they can be regarded as the top (semantic) layer by which high-level preferences can be finally inferred. These rules can be hand-coded by experts to define both syntactic and semantic customization rules that will be deployed using the underlying knowledge available in order to further characterize user changes (e.g. when a same user has changed the navigational structure or the layout, or several users have made similar modifications). This can be inferred in a domain-independent way by analyzing the knowledge available in the expert system from previous sessions.

Knowledge is built up progressively from the lowest levels to the highest ones. The knowledge constructed at the lowest levels is basically composed of syntactic information automatically generated by the system. This information is inferred by comparing the concrete user interface specifications before and after the user's changes, and is related to the elements the user implicitly manipulates when modifying a presentation. The system provides mechanisms to analyze user modifications and identify to what knowledge level they belong. For instance, when the user attempts to modify a form, s/he might decide to replace a Radio Button with a Selection List. This change will be considered as a syntactic one, since both interactors are under the same abstract category: *single selection interactor,* which identifies its main semantic effect. However, if the user decides to replace a Radio Button with a CheckBox, then a semantic change occurs, since Radio Button belongs to the *single selection interactor* abstract category and CheckBox to *multiple selection interactor*. In this case, even the task supported by the application changes because of such a user change.

In addition, for each user change the system extracts the corresponding presentation context, which is based on the abstract specification of the interface. This allows the definition of more general rules that can be associated with similar, though not exact, user modifications.

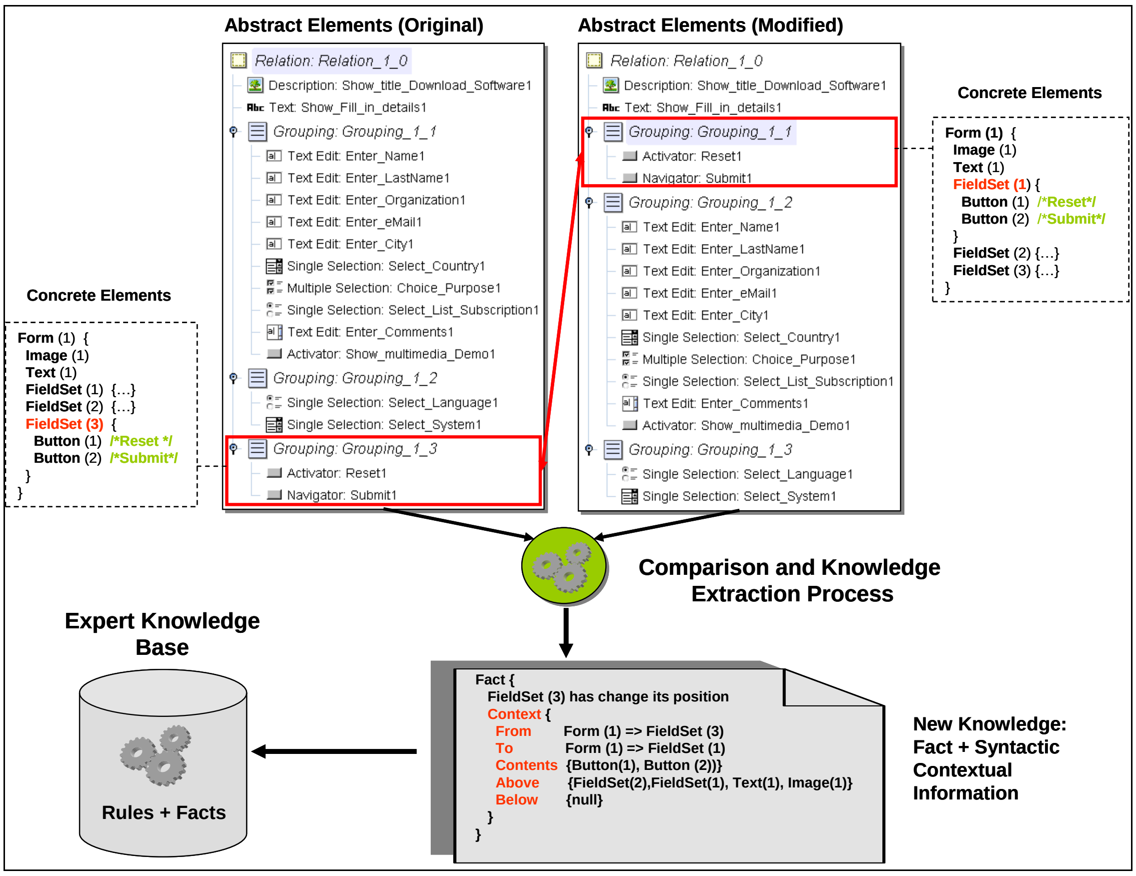
5. Inferring Meaningful Information from User Modifications

Expert Rules deal with information about user preferences from one session to another, taking into account the existing facts generated by the previous expert-knowledge layers. If these rules are often fired, then the customization rules corresponding to the user's design preferences can be detected. Therefore, we can distinguish two different kinds of rule activations in our system: those pending and the permanent ones. Pending customization rules are identified whenever the system detects a probable user attempt to customize an element or a group of elements in the presentation, whereas permanent customization rules correspond to pending rules that have been identified more than twice in the same context. This mechanism allows us to differentiate occasional user modifications from explicit and repetitive preferences. Permanent customizations will drive the customization of the application in future accesses, and they can be turned on and off by end-users using the NOTORIOUS environment (see Section 6.2).

Each customization rule (both pending and permanent) is associated with a reference context, which is useful to identify whether user changes are substantially similar. When a change is made to a page, the system analyses the page structure and the elements surrounding the change to define its context. Then, it checks whether there is a similarity with any of the presentation contexts of the identified customization rules**.**

The contextual presentation information is processed in two stages. The first stage extracts syntactic context from the comparison of the concrete interface specifications of the original and the modified page. The second stage operates at the semantic level, processing syntactic context and relating it to knowledge concerning abstract elements, with the aim of obtaining meaningful information that could be deployed to apply more general rules in similar contexts. This way, when differences are found through comparison of the concrete interface specifications, this new knowledge is added to the expert system in terms of facts (as explained in Figure 2), together with the syntactic context in which such changes take place. The location of the modifications is extracted from the concrete specification, which provides sufficient detail to accurately identify the objects surrounding a user's change.

Figure 5 depicts how syntactic context is identified from a user's change to the Web form in Figure 3. Such change consists of moving both the Reset and Submit buttons to the upper part of the form, between the page title and the personal data input fields. This change could be due to the fact that the user prefers such buttons to appear above the other form elements.


Abstract Elements (Original)
Relation: Relation_1_0
Description: Show_title_Download_Software1
Text: Show_Fill_in_details1
Grouping: Grouping_1_1
Text Edit: Enter_Name1
Text Edit: Enter_LastName1
Text Edit: Enter_Organization1
Text Edit: Enter_eMail1
Text Edit: Enter_City1
Single Selection: Select_Country1
Multiple Selection: Choice_Purpose1
Single Selection: Select_List_Subscription1
Text Edit: Enter_Comments1
Activator: Show_multimedia_Demo1
Grouping: Grouping_1_2
Single Selection: Select_Language1
Single Selection: Select_System1
Grouping: Grouping_1_3
Activator: Reset1
Navigator: Submit1
Abstract Elements (Modified)
Relation: Relation_1_0
Description: Show_title_Download_Software1
Text: Show_Fill_in_details1
Grouping: Grouping_1_1
Activator: Reset1
Navigator: Submit1
Grouping: Grouping_1_2
Text Edit: Enter_Name1
Text Edit: Enter_LastName1
Text Edit: Enter_Organization1
Text Edit: Enter_eMail1
Text Edit: Enter_City1
Single Selection: Select_Country1
Multiple Selection: Choice_Purpose1
Single Selection: Select_List_Subscription1
Text Edit: Enter_Comments1
Activator: Show_multimedia_Demo1
Grouping: Grouping_1_3
Single Selection: Select_Language1
Single Selection: Select_System1
Concrete Elements
Concrete Elements
Comparison and Knowledge Extraction Process
Expert Knowledge Base
Rules + Facts
New Knowledge: Fact + Syntactic Contextual Information

```
Form (1) {
  Image (1)
  Text (1)
  FieldSet (1) {…}
  FieldSet (2) {…}
  FieldSet (3) {
    Button (1) /*Reset */
    Button (2) /*Submit*/
  }
}
```

```
Form (1) {
  Image (1)
  Text (1)
  FieldSet (1) {
    Button (1) /*Reset*/
    Button (2) /*Submit*/
  }
  FieldSet (2) {…}
  FieldSet (3) {…}
}
```

```
Fact {
  FieldSet (3) has change its position
  Context {
    From      Form (1) => FieldSet (3)
    To        Form (1) => FieldSet (1)
    Contents  {Button(1), Button (2)}}
    Above     {FieldSet(2),FieldSet(1), Text(1), Image(1)}
    Below     {null}
  }
}
```

Figure 5. The process of building knowledge from detected user changes. The relations among the abstract and concrete elements of the interface involved in the user modifications are shown.

Figure 5 shows the structure of the logical specification of the original and the modified XHTML pages obtained by the reverse engineering tool developed in our group. The rectangles denote the modifications to the Reset and Submit buttons, involving moving the buttons originally contained in *FieldSet3* into *FieldSet1*. This change can be regarded as a single one, since it involves a grouping of concrete interactors (in this case composed through a Fieldset element). Then, the system extracts information about the change and also about the syntactic context from the concrete presentation, identifying where the change has been performed.

In our system, all rules and facts are coded in LISP. In the paper, for the sake of readability, we provide a pseudo-code representation. In the example, the syntactic knowledge is updated by a syntactic rule fired whenever a Fieldset changes, associating the change to a specific user and platform and updating the existing information stored in our expert system:

```
Rule {
      If   change(FieldSet)
      Then update(FieldSet, to, from, contains, above, below, user,
                  platform)
}
```

While facts are automatically generated by the system, rules have to be defined by the expert system designer. However, the rules notation is flexible enough to allow for general specification based on logical interface descriptions, so they may fire for a great deal of different elements and situations by defining them only once. In this example, the information that the system updates by means of the previous rule is:

```
Assert {
   FieldSet(3) has changed its position
   Context {
     From      Form(1) => FieldSet(3)                              1
     To        Form(1) => FieldSet(1)                              2
     Contents  {Button(1), Button(2)}                              3
     Above     {FieldSet(2),   FieldSet(1), Text(1), Image(1)}     4
     Below     {null}                                              5
   }
   (user (user_name))                                              6
   (platform (platform_name))                                      7
}
```

This information refers mainly to the fact that a Fieldset has changed its position for a given user and platform, reflecting (*1*) which concrete composition operator is involved in the change (a FieldSet in a Form), (*2*) what concrete composition operators it has been changed to. Likewise, the fact indicates also the elements (*3*) inside the Fieldset: `Button(1)` which is the Reset button and `Button(2)` which is the Submit button, (*4*) the list of the concrete interactors positioned above before the change, (5) the list of the concrete interactors positioned below before the change (in this case there was nothing below), and finally the user (*6*) and the platform (*7*) involved in this change.

Semantic knowledge is also generated to reflect the presentation context of the change. This information will be generated by a specific rule, fired whenever a syntactic change occurs and the modification is updated in the expert system:

```
Rule {
      If   update(FieldSet)
      Then generate_presentation_context(FieldSet)
)
```

This rule calls a function (`generate_presentation_context`) that extracts contextual information for the previously modelled syntactic change. One of the principal concerns here is to transform syntactic information already inferred into semantic information. To this end, the corresponding abstract elements are taken into account. This way, by analysing the previous information about the syntactical change on a Fieldset, the information that is finally generated by the previous function at the semantic level can be represented by the following presentation context:

```
Presentation_Context {
     (Change_type (Movement))
     (from        (Relation(FORM),     Grouping(FIELDSET))
     (to          (Relation(FORM),     Grouping(FIELDSET))
     (Contents    (Activator(SUBBUT),  Navigator(RESBUT))
     (above       (Grouping(FIELDSET), Grouping(FIELDSET), Text(TEXT),
                   Description(IMAGE))
     (below       (null))
}
```

The presentation context is defined in terms of abstract elements that are taken from the abstract description of the page considered. To this end, a key concern is to identify interactors and composition interactors surrounding the change, as well as the relation with the concrete elements for further disambiguation. The function `generate_presentation_context` also detects (by using syntactic context) what kind of change is performed (movement, deletion, insertion). In this case, the system has inferred a change that involves a movement from one grouping (the last one) to another one (the first one). The abstract elements moved are an activator and a navigator interactors, and the context (see above) is composed of different composition operators and interactors. This elicited high-level information is useful for the expert level to carry out generalizations that can be applied as customization rules more than once. In general, the semantic level attempts to construct the suitable knowledge for dealing with complex reasoning. In the example “the user seems to prefer the form buttons appearing on the top” can be carried out. This knowledge can be constructed by associating the contextual presentation information with the user change. In order to detect whether a user change corresponds to a certain context, a similarity percentage is calculated in the matching process. Thus, the semantic customization rules are associated with the corresponding presentation contexts.

We have a general function called `ContextMatching`, which identifies when two different presentation contexts are similar. Let *PC*1 and *PC*2 be two different presentation context sets (a set is composed of various interface elements that define the presentation context). *ContextMatching*(*PC*1,*PC*2) can be defined as:

$$ContextMatching(PC1, PC2) = \begin{cases} true \quad if \quad \dfrac{|PC1 \cap PC2| * 100}{|PC1 \cup PC2|} & \geq 70\% \\ false \quad if \quad \dfrac{|PC1 \cap PC2| * 100}{|PC1 \cup PC2|} & < 70\% \end{cases}$$

In a nutshell, this heuristic calculates the number of presentation context elements (such as interactors, their positions and attributes) that show differences between one set (*PC*2) and a reference set (*PC*1). To this end, a percentage ratio is calculated and afterwards used as a comparative numerical value. We have determined that a 70% of similarity is enough to consider that the current presentation context matches the reference presentation one, which is associated with a customization rule. This is an empirical threshold that we have estimated by analyzing different examples and cases of use, and it has shown to work well in most of the cases that we have addressed. Initially, we considered a 50% threshold, which was quite a risky percentage for a great deal of the experiments we made.

Though it cannot be considered exhaustive, the matching operator provides a good heuristic in order to obtain a useful numerical result for comparing different contexts and identifying possible similarities.

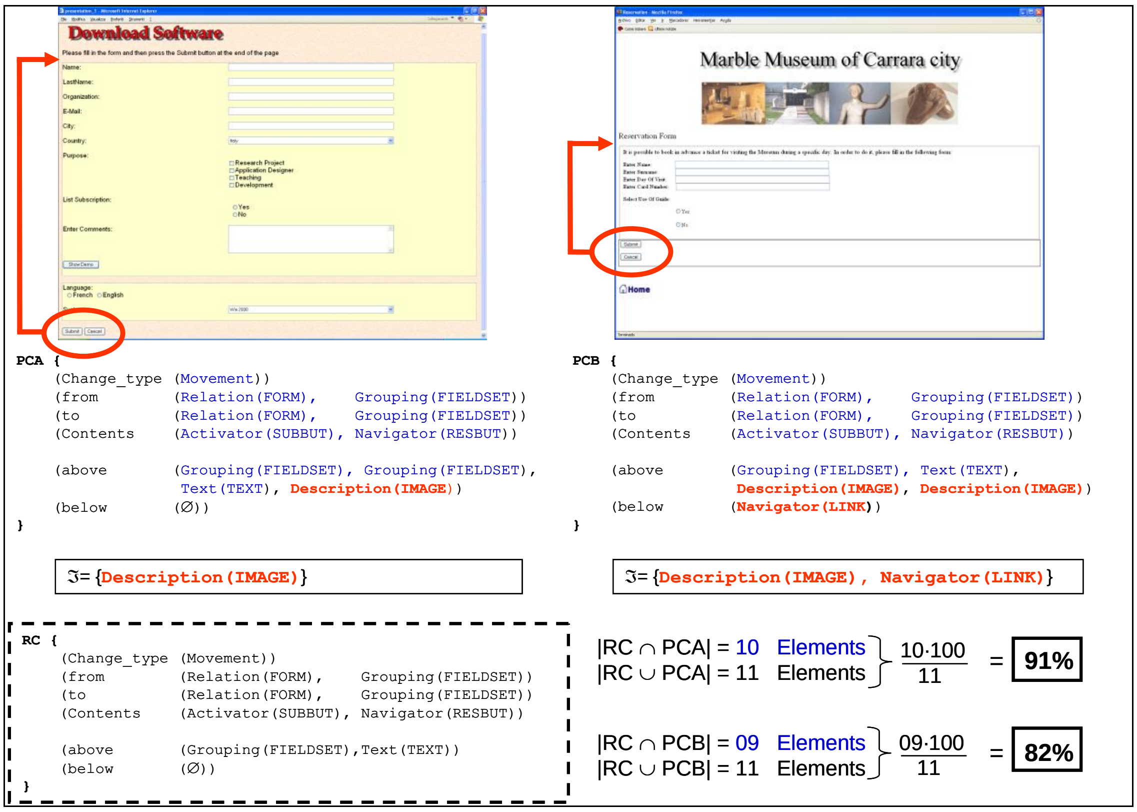
Figure 6 shows an example, where two different presentation contexts (*PCA* and *PCB*) are compared with a reference context (*RC*) which may be extracted from the knowledge base; presentation contexts can be stored in the knowledge base as any other knowledge and thus they can even be created by designers. The results from the comparison are 91% coincidence for the first case and 82% for the second, which is sufficient to state a similarity of contexts. The first example (on the left) is the example of a previously presented form. The second example (on the right) is another presentation containing a Web form with different object distribution. In this case, the reference for comparison is a form-like context that can be represented by the logical descriptions appearing inside the dotted-square (denoted by *RC*) in Figure 6. In comparing both presentation contexts, a set $\Im$ is calculated. It represents the following information:

$$\Im = \{x \mid x \in PC2 \vee x \notin PC1\}$$

This set contains the elements of *PC*2 that are not in *PC*1. In the example in Figure 6, only one element differs significantly in the left Web form (`Description(IMAGE)`) and two in the right one (`Description(IMAGE), Navigator(LINK)`). This way, the results of the matching process for *PCA* and *PCB* with respect to the reference context *RC* is:

```
ContextMatching(RC,PCA) = true (91%)
ContextMatching(RC,PCB) = true (82%)
```

The results obtained for these presentation contexts (both above 80% of similarity) imply that changes in moving buttons on the top of the form can be generalized and applied to both contexts by the same expert rule.



```
PCA {
     (Change_type  (Movement))
     (from         (Relation(FORM),     Grouping(FIELDSET))
     (to           (Relation(FORM),     Grouping(FIELDSET))
     (Contents     (Activator(SUBBUT),  Navigator(RESBUT))

     (above        (Grouping(FIELDSET), Grouping(FIELDSET),
                    Text(TEXT), Description(IMAGE))
     (below        (∅))
}
```

ℑ= {Description(IMAGE)}

```
PCB {
     (Change_type  (Movement))
     (from         (Relation(FORM),     Grouping(FIELDSET))
     (to           (Relation(FORM),     Grouping(FIELDSET))
     (Contents     (Activator(SUBBUT),  Navigator(RESBUT))

     (above        (Grouping(FIELDSET), Text(TEXT),
                    Description(IMAGE), Description(IMAGE))
     (below        (Navigator(LINK))
}
```

ℑ= {Description(IMAGE), Navigator(LINK)}

```
RC {
     (Change_type  (Movement))
     (from         (Relation(FORM),     Grouping(FIELDSET))
     (to           (Relation(FORM),     Grouping(FIELDSET))
     (Contents     (Activator(SUBBUT),  Navigator(RESBUT))

     (above        (Grouping(FIELDSET),Text(TEXT))
     (below        (∅))
}
```

$$\left.\begin{array}{l}|RC \cap PCA| = 10 \text{ Elements} \\ |RC \cup PCA| = 11 \text{ Elements}\end{array}\right\} \frac{10 \cdot 100}{11} = \boxed{91\%}$$

$$\left.\begin{array}{l}|RC \cap PCB| = 09 \text{ Elements} \\ |RC \cup PCB| = 11 \text{ Elements}\end{array}\right\} \frac{09 \cdot 100}{11} = \boxed{82\%}$$

Figure 6. Matching process for detecting presentation context similarities in two different presentations. The contexts to compare (PCA and PCB) are showed, as well as the process of identifying similarity.

All this knowledge can be used afterwards in order to define future semantic customization rules. This way, if a semantic customization rule is fired by the expert system a certain number of times (3 or more, in this case), this rule will be active whenever a form is generated for this user and for the platform s/he is using for navigating through the application. In order to apply the rule correctly, the contextual presentation information will be considered.

This way the semantic customization rule can be specified as:

```
Semantic_Customization_Rule(element, current_context) {
      If  Change(element)
      And
```

```
            ContextMatching(reference_presentat_context,current_context)
        And
            This rule has been activated more than 2 times
        Then
            Render_Form (element) /* element will be rendered first */
}
```

This customization rule drives the modifications of the Web pages for future access. It is specified in the user profile and can be activated or deactivated by using the NOTORIOUS application.

The advantage of using presentation context becomes evident when expert rules, as the previously discussed, need to be defined. Since presentation contexts are constructed from the first levels of the knowledge structure, semantic customization rules can be defined using abstract and domain-independent concepts, mostly focusing on what the rule is expected to do considering a concrete context:

```
IF    ChangeN Occurs In ContextM
THEN  Act like this: ...
```

Not all the expert rules need presentation context to be considered. Some syntactic customization rules can be constructed using only syntactical information, dealing with concrete interactors and concrete composition operators. They concern changes to attributes such as font colour, font size, background colour and so on. These changes will be modelled in the system as described above, but in this case, presentation context is no longer needed. This way we can define flexible syntactic customization rules, taking into account concrete user-interface information and syntactic context previously generated by the system:

```
Syntactic_Customization_Rule(element) {
        If    Change(element)
              This rule has been activated more than 2 times
        Then
              element will be considered for customizing future nomadic
                      applications
}
```

6. User Evaluation and Discussion

In order to receive some empirical feedback for the method proposed, we have carried out a user evaluation. Concretely, the principal objectives of the evaluation were:

- Test the system with real users.
- Evaluate the rules programmed and how they reacted.
- Analyse user interactions and detect meaningful customizations and expert rule activations.
- Populate the expert system with new knowledge to be considered in future user sessions.

In the test, we used an existing Web application generated by the TERESA tool (Mori et al., 2004). The application consisted of several pages about The Marble Museum of Carrara. We asked users to modify the museum application in order to express their preferences, and then analyse the response from our intelligent environment.

6.1 The example used

The Marble Museum of Carrara is a nomadic application. Different versions have been generated for different platforms (mobile, PDA, voice, desktop) through a model-based environment. Although our approach supports different platforms, we based our user test on evaluating the desktop version of the application as the most common platform used by end-users. On the other hand, the desktop platform allows end-users to carry out more expressive modifications, which can provide useful information for customization. The structure of the test Web site includes the typical navigation and presentation structure of many Web pages. The museum site is a large application with hundreds of pages, but during the test users needed to access only a part of it.

Figure 7 shows some screenshots of the museum application for a desktop platform. These pages include a great variety of interface components that can be used to infer syntactic and semantic changes and to automatically activate expert rules. Presentation elements and their corresponding types are described in Table 1.

	Abstract	Concrete
Interactors	Descriptions	Images
	Navigators	Links and Buttons
	Texts	Files
	Text and Numerical Edits	Text Fields
	Single Selections	Radio Buttons and Lists
	Activators	Reset Buttons
Composition Operators	Groupings	FieldSets and Columns
	Relations	Forms

Table 1. Abstract and concrete elements included in the desktop museum application that are grouped into interactors and composition operators.

As indicated in previous sections, abstract and concrete elements are used to setup the knowledge base and create facts and rules that react to user modifications. Such different semantic levels are useful to distinguish between semantic and syntactical changes.

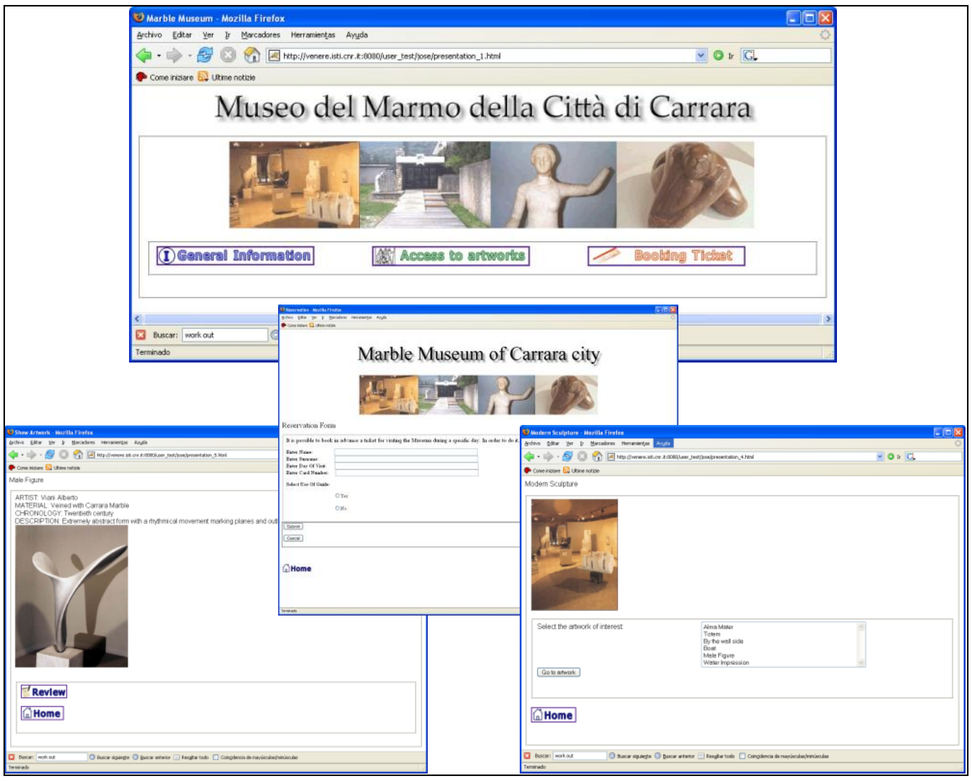


Figure 7. Some screenshots of the museum application used for the user evaluation.

Considering the application and, more concretely, the conceptual levels of the interface elements depicted in Table 1, the expert system contained different kinds of expert rules that can be divided into syntactic and semantic customization rules. In particular, the expert rules specified for the user evaluation can be summarized as follows:

- Syntactic customization rules
  - Concrete interactors
    - Detecting text font colour and size preferences
    - Detecting image attribute preferences
    - Detecting background colour preferences
    - Detecting button attribute preferences
    - Detecting attribute preferences on links and graphical links
  - Concrete composition operators
    - Detecting form attribute preferences
    - Detecting Fieldset attribute preferences
    - Detecting column attribute preferences
- Semantic customization rules
  - Interactors
    - Detecting transformations of different kinds of interactors
    - Detecting deletions of interactors inside a grouping
    - Detecting insertion of interactors inside a grouping
  - Composition operators
    - Detecting transformation of different kinds of composition operators
    - Detecting movement of interactors from one composition operator into another
      - Detecting hierarchy preferences
      - Detecting ordering preferences

- Detecting navigational preferences

The expert system included a total of 14 syntactic customization rules and 10 semantic ones that were activated according to the modifications achieved by end-users.

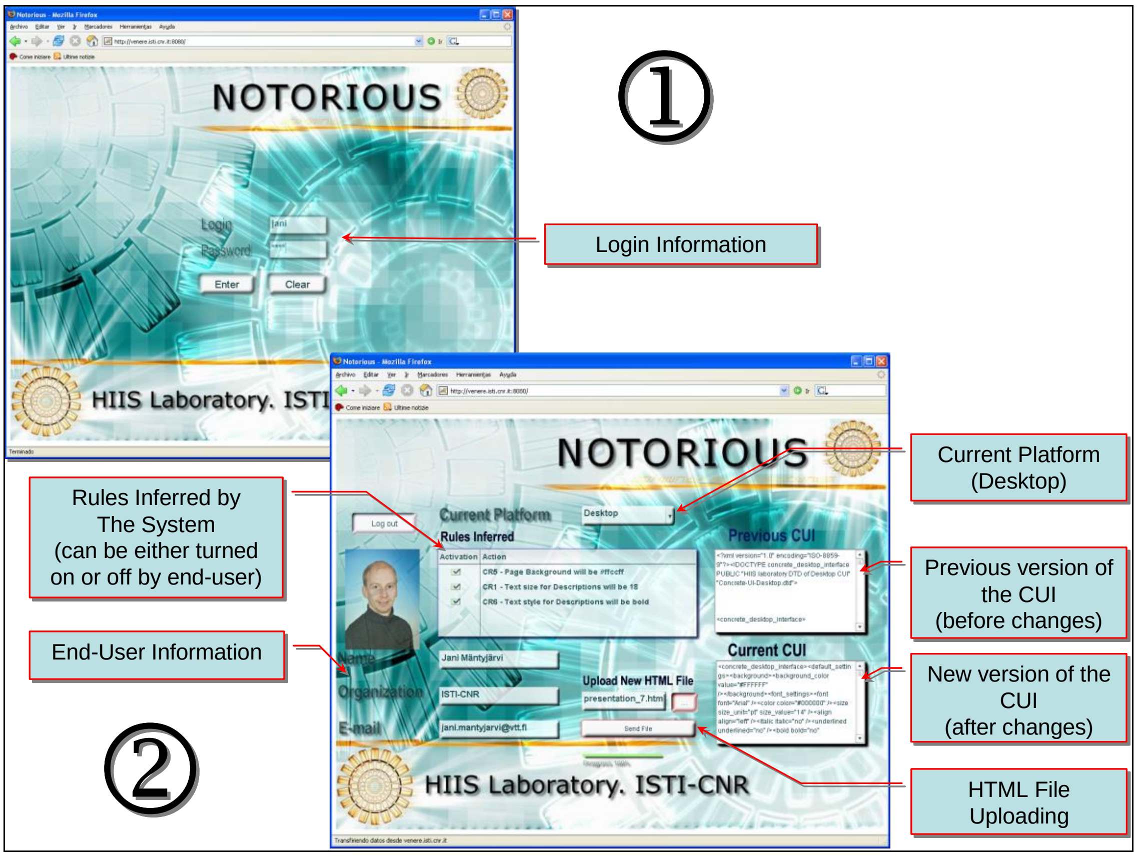
6.2 Experimental procedure

For this user test, we recruited 11 participants from our institute, all of them having heterogeneous scientific backgrounds. Post-study interviews revealed that only 3 participants had strong Web programming experience. The rest of the participants' experience was limited to navigating, creating and modifying simple Web pages by using diverse Web authoring tools. Participants were 4 females and 7 males with ages between 25 and 40.

The test was individually performed by each participant in her/his office. It consisted of the following steps:

1) Initially, for about 5 minutes, each user received basic explanations on the system, test goals and the task to accomplish - i.e. providing sample modifications of the desktop version of the museum application in order to indicate their preferences.
2) Then, each user was provided with the URL of the museum application. All participants had unlimited time to navigate through the application and, using the preferred authoring tool available on their computers, made modifications to anything they thought it could be improved.
3) Next, each user uploaded the modified application pages using the NOTORIOUS (Nomadic TailORIng On an end-User Server) environment. A user profile for each user was previously created with the aim of recording the activity and the customization rules inferred. NOTORIOUS is a specialized Web environment through which the user can send modified (X)HTML pages and also access his personal profile to see the changes and manipulate high-level rules inferred by the system. To this end, the user must log in (see Figure 8, screenshot 1). Next, the system presents the personal information and the rules inferred (see Figure 8, screenshot 2). Internally, the back-end of NOTORIOUS generates the UIDs of the original and the modified page, compares them, provides the user with comprehensive feedback and populates the expert system with suitable information about the user's modifications.
4) Lastly, each user was requested to fill in a questionnaire based on the Perceived Ease of Use (Davis, 1989).

After the user session, the system's behaviour was analysed using the NOTORIOUS activity logs. In addition, the results from the questionnaire were also processed to compare the outcome of every user session with the user's perception about the system.



①
NOTORIOUS
Login
Password
Enter
Clear
HIIS Laboratory. ISTI
Login Information
NOTORIOUS
Log out
Current Platform
Desktop
Rules Inferred
Activation
Action
CR5 - Page Background will be #ffccff
CR1 - Text size for Descriptions will be 18
CR6 - Text style for Descriptions will be bold
Previous CUI
Current CUI
Name
Jani Mäntyjärvi
Organization
ISTI-CNR
E-mail
jani.mantyjarvi@vtt.fi
Upload New HTML File
presentation_7.html
Send File
HIIS Laboratory. ISTI-CNR
Current Platform (Desktop)
Rules Inferred by The System (can be either turned on or off by end-user)
Previous version of the CUI (before changes)
End-User Information
New version of the CUI (after changes)
HTML File Uploading
②

Figure 8. The NOTORIOUS user interface, which allow users to control rule activation, upload new Web-page modifications and see the changes made.

6.3 Results and discussion

Since the test was carried out directly at the computer of each user, one interesting aspect was to see what kind of navigation and Web authoring tools each user used to navigate throughout the Web application and make modifications. Figure 10 shows the percentage rate of (a) navigation and (b) authoring tools used during the test.

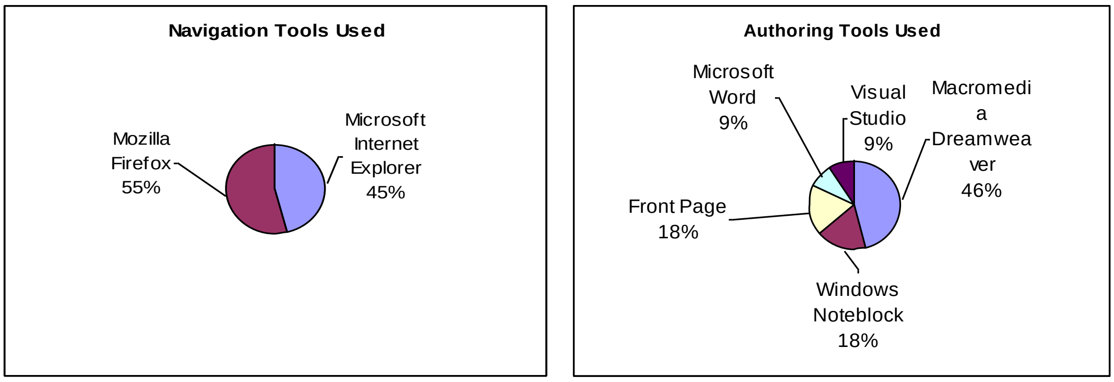


Figure 9. Navigation (a) and authoring (b) tools used by every user to achieve the test. During the test, users had freedom enough to utilize the tools they preferred.

Some measurements of the duration of the user sessions are presented in Table 2. As shown, the average time was about 25 minutes. We initially estimated the individual total time needed to carry out the modifications in about 30 minutes. In general, the time taken by each user depended on the changes accomplished. While some users decided to make some in-depth changes affecting the navigational Web structure or moving interactors from one page into another, others considered only some easy changes concerning syntactic modifications to style, colours and so forth.

	Time in Minutes
Max	38
Min	16
Mean	25
Deviation	8

Table 2. Time spent by users in carrying out the test. Max, Min, Mean and Deviation values are provided in order to have an idea of the time consumed.

During the test, different system outputs were also observed, and a detailed report was additionally obtained from each user session. It was interesting to measure the number of changes made by each user, as well as the response of the expert system in terms of number of facts generated and the number of rules activated in response to the user's changes. Figure 10 presents the correlation between the number of changes, the facts generated and the rules activated for each user. At first sight, it seems clear that the more changes made the more facts and rules activated. However, this correlation is not always as linear as one could expect, since it mostly depends on the complexity of the changes performed. In the case of user #2, one can see that the number of changes is lower with respect to other users, but the number of facts and rules activated is higher. This is due to the fact that user #2 made a total of 9 changes but all involving complex tasks, that is, moving interactors, changing the navigational structure of the page, transforming composition operators and so on. This produced a high number of facts that had to be specified in terms of syntactic information and semantic presentation context. In addition, the rules that had to deal with such changes were even more complex than trivial syntactic ones, so that a chain of rules was activated. By contrast, users #8 and #11 carried out a high number of changes (23 and 26, respectively) that generated a higher number of facts (57 and 72, respectively) created by the system, as well as a high rate of rule activations (32 and 42, respectively). In these cases, most changes were syntactical, so the response of the system was quite proportional to the type and number of changes carried out by these users. In conclusion, it is possible to affirm that the response of the system is linear as long as the user's changes do not involve complex structural changes. Anyway, this aspect does not affect our system's performance and throughput.

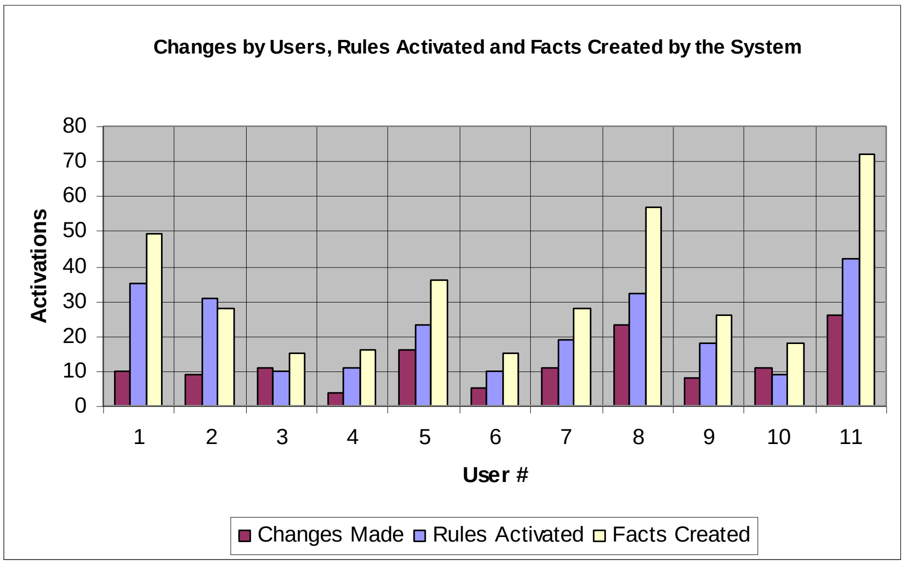


Figure 10. The system's response to user changes. The number of changes, rules activated and facts created is shown for each user.

The activations showed in Figure 10 represent the total rule activations during the user sessions, which means that they include internal and permanent rule activations. Internal rules are those automatically fired by the system as a consequence of fact and rule chain activations and depend exclusively on the expert system. To have a clear idea about what kind of rules have been activated, it can be useful to compare for each user the number of both pending and permanent rule activations. Figure 11 shows such information, where internal rule activations have been omitted and only the ratios of pending and permanent expert rule activations have been considered. As explained in previous sections, the permanent rules are those that have been triggered more than twice, whereas the pending rules are those that have been identified at least once but less than three times. As explained, expert rules involve both syntactic and semantic customization rules. It is worth noting in Figure 10 that most expert rule activations correspond to pending syntactic customization rules associated with syntactical changes. These activations trigger permanent customization rules only when they are fired more than twice. Additionally, there were activations of pending semantic customization rules (for users #1 and #2, basically) that later were turned into permanent preferences.

All pending rules remain in the system as a basic knowledge that will be taken into account for future customization. That is a key point for the system in order to infer similar changes on similar context, which can be useful to detect the same changes on different presentations. In essence, Figure 11 reflects that, although some semantic customization rules have been activated as response to user actions, most user changes concern syntactic modifications that in some cases are transformed into permanent syntactic customizations later on.

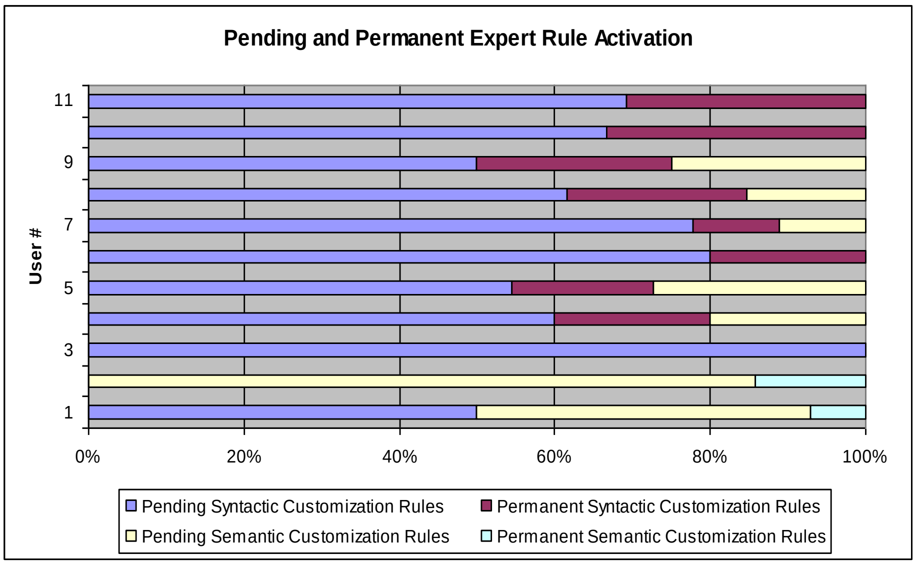


Figure 11. Expert rule activation ratio for each user in terms of pending and permanent activations for both syntactic and semantic customization rules.

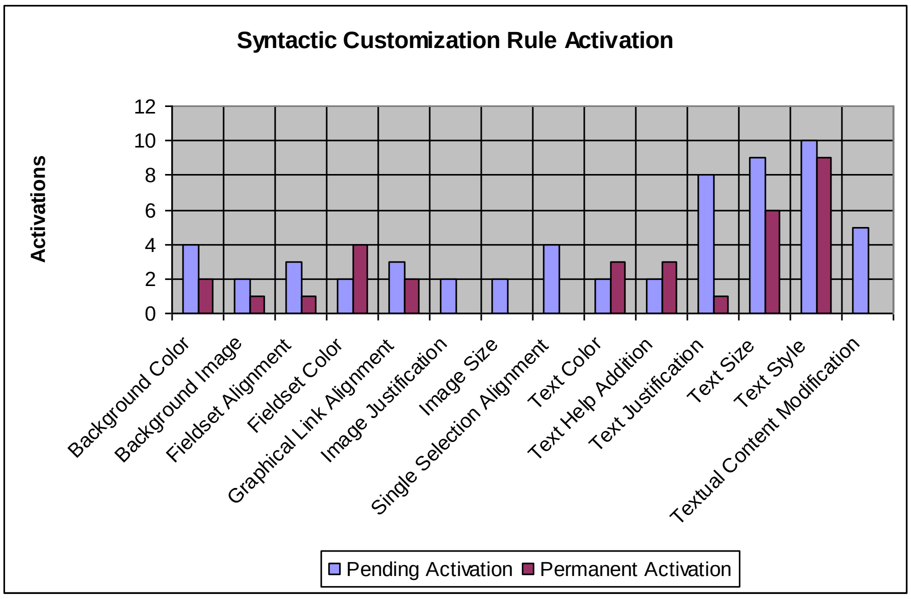


Figure 12. Number of occurrences of each syntactic customization rule and the kind of change carried out by users during the test for both pending and permanent rules.

Besides the rate of expert rule activations, we also measured the kinds of rules inferred for each user. Figure 12 shows the number of instances (pending and permanent ones) for each of the 14 syntactic customization rules. It is worth noting in Figure 12 that the rules most often activated were those concerning “text size” and “text style”. These two rules had a high rate of both pending and permanent instances. The number of permanent activations for some syntactic customization rules such as “Fieldset colour”, “text colour” and “text help addition” was higher than for the pending ones.

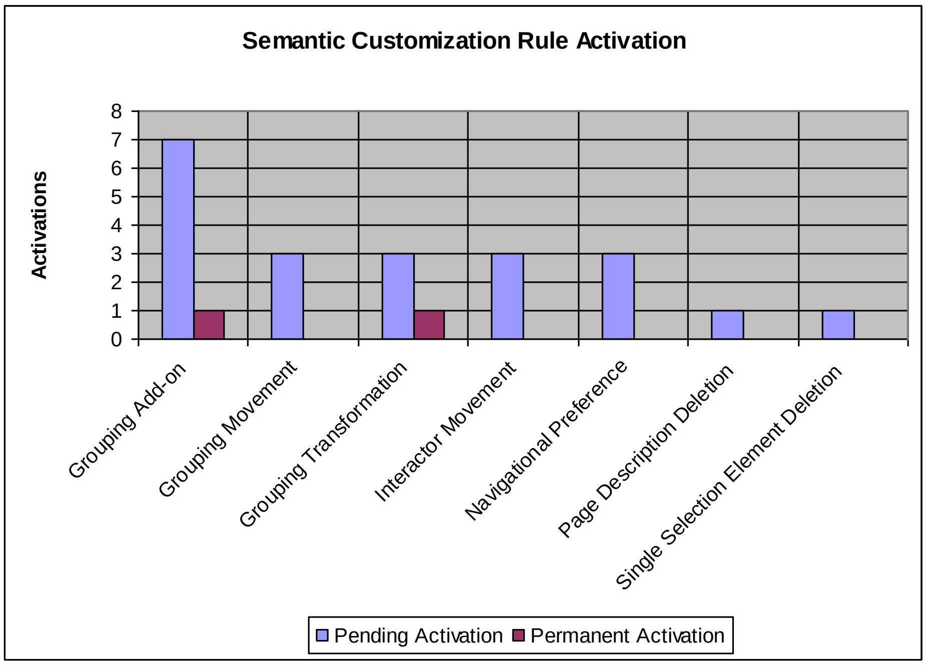
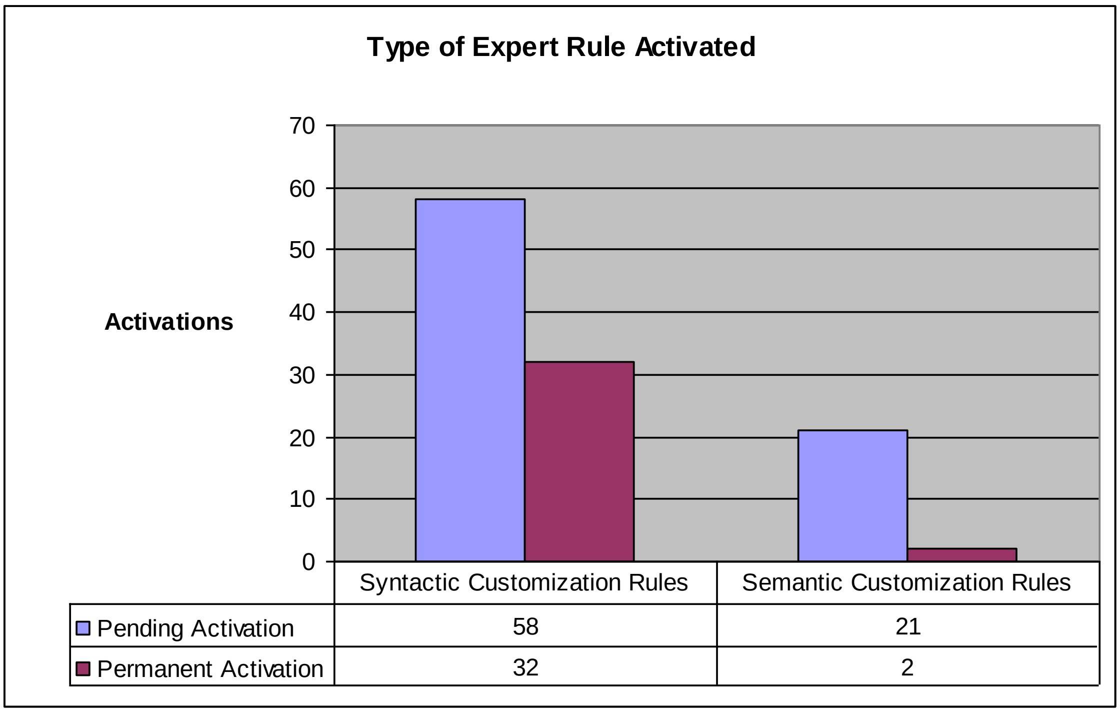


Figure 13. Number of activations of each semantic customization rule and the kind of change carried out by users during the test for both pending and permanent rule activation.

On the other hand, Figure 13 shows the activations for different semantic customization rules, which are less than the syntactic ones. In this case, the higher rate corresponds to the semantic customization rules concerning "grouping add-on", used to detect when the user decides to insert o create new interactors in a grouping of interactors. Grouping transformations have also been detected by our system, in a lower rate. As shown in Figure 13, only a couple of pending semantic customization rules has been turned into permanent ones. As for the rest of the semantic customization rules, only pending activations have been detected.

Generally speaking, the number of syntactic customization rules greatly overcomes the number of semantic ones. This is due to the fact that syntactical aspects are easier to modify and have an immediate impact on the user's perception**.** Concretely, this fact reflects that most changes made by users were related to syntactic modifications such as changing font style, size, colour, text justification, and so on. Figure 14 shows that 80% of activations corresponded to syntactic customization rules and only 20% to semantic ones. As for the syntactic customization rule activations, 64% were considered pending whereas only 36% were permanent. With respect to semantic customization rules, only 9% of activations were permanent and by contrast 91% were pending.



	Syntactic Customization Rules	Semantic Customization Rules
▢ Pending Activation	58	21
■ Permanent Activation	32	2

Figure 14. Comparison of expert rules activated during the user test, showing both permanent and pending activations over the total of rules activated.

The results obtained from the system for each user helped us evaluate what changes were considered relevant as well as to detect meaningful preferences to populate the expert system with domain-independent customizations. In addition to these outcomes, the perception of the user was also considered in order to have feedback regarding the ease of use of the environment. To this end, we requested each user to fill in a questionnaire to evaluate the perceived ease of use of our system. The questionnaire consisted of 6 questions targeted at evaluating the ease of use. The range of the answers were from 1 to 7, that is, 1) Absolutely Disagree, 2) Disagree, 3) Not Very Agree, 4) Indifferent, 5) Agree, 6) Very Agree, 7) Absolutely Agree and NA (No Answer). Additionally, the questionnaire comprises a free-answer part where the user can freely express other issues related to the system and the test.

Table 3 summarizes the result obtained for the evaluation of the perceived ease of use of the system. In general terms, users found the explicit mechanisms simple in comparison to the support that the system provides. Opinions extracted from the questionnaire denoted how users perceived the implicit expressiveness in modifying a great deal of Web pages using any authoring tool available and then easily uploading them into a system that produces customizations automatically. Additionally, diverse opinions collected from the free-answer part of the questionnaire revealed useful areas of applications for the approach, suggesting the idea of applying the system to tedious daily user tasks such as automatically modifying Web sites and blogs just making minimal changes to a couple of pages. In this regard, end-users found useful the feature by which the system obtains meaningful preferences that will be applied automatically later on in the design of other similar applications.

Answer / Question	1	2	3	4	5	6	7	NA
Learning to operate the system would be easy for me	0%	0%	0%	9%	27%	9%	55%	0%
I would find it easy to get the system to do what I want it to do	0%	0%	0%	27%	0%	55%	18%	0%
My interaction with the system would be clear and understandable	0%	0%	0%	10%	20%	30%	40%	0%
I would find the system to be flexible to interact with	0%	0%	9%	0%	27%	27%	37%	0%
It would be easy for me to become skilful at using the system	0%	0%	0%	9%	18%	36%	37%	0%
I would find the system easy to use	0%	0%	0%	9%	27%	27%	37%	0%

Table 3. The users' perceived ease of use was analysed by taking into account the questionnaire filled in after the experiment. Answers ranged from 1 (absolutely disagree) to 7 (absolutely agree) and NA (no answer).

In essence, the results obtained from this questionnaire fulfil our expectations. They provided us with positive empirical feedback indicating that it is possible to reduce the complexity of customization and reach a trade-off between expressiveness and easy of use in end-user development mechanisms.

7. Conclusions

Customization of software artefacts is everyday more a common practice carried out by end-users in their daily activities (Klann, 2003). However, such practices require the accomplishment of tasks that are too difficult complex for most end-users. This is mainly due to the fact that authoring environments require manipulating programming languages and abstract specifications, as it occurs when customizing interactive applications.

An interesting study by Rode and Rosson (2003) revealed that although much progress has been made by commercial development tools, most of the tools that they reviewed did not lack functionality but rather ease of use. Rode and Rosson explored many different paths, including extensions to development tools, finding the inflexibility in controlling the users' workflow as the main hindrance to adopting these approaches. Currently, none of the commercial tools that they reviewed would work without major problems for the non-professional Web developer.

Commercial applications generally lack support to carry out customization of Web applications. Several researchers have sought to reduce the learning burden by creating design environments that do not require users to program per se. Rather, they design by instructing the machine to learn from examples (Lieberman, 2001) or by interacting with graphical micro worlds representing real domains. Our approach follows such guidelines and supports an easy mechanism based on Programming by Example techniques, where the user provides the system with changes (example of what s/he want to change) and the system generates customizations that will be applied automatically to the pages available for future access, thus minimizing the amount of authoring needed. Instead of enforcing end-users to make use of programming languages and complex specifications, our system carries out Web customization automatically by extracting meaningful information from the user's changes that will be stored in a profile and used to support future sessions.

We populate the knowledge base using logical user interface descriptions that provide domain-independent information, which can be applied to other applications. Often, expert systems are traditionally used to work on concrete problems, since knowledge is created considering information of a specific domain. We overcome such limitation using different levels of knowledge, creating mappings between them and the conceptual levels associated with the user interface (Puerta and Eisenstein, 1999). This allows our intelligent environment to carry out inference at different levels of abstraction (syntactic, semantic), activating rules and populating the expert system with different knowledge depending on the changes that the user accomplishes.

We have carried out a first user test, which has provided us with useful information to analyse the real behaviour of the intelligent system. For example, it was interesting to observe that most of the users' changes were syntactic, rather than semantic. However, semantic rules imply deeper modifications of the application, in particular to the tasks supported and how to accomplish them. Even if they may be less frequent than syntactic modifications, they can be very important for end users whenever they do not feel that the semantics of the application is satisfactory. Additionally, pending and permanent rule activations helped us check the suitability of the knowledge structure proposed for our intelligent approach, taking intro account the user's changes and analysing the way the system reacted to them. After the experiment, we informally presented the rules to the users with the aim of corroborating whether the inferred knowledge corresponds to their customization preferences or not. However, this knowledge was not applied to other Web applications, although the information reported, together with the questionnaire filled in by users, provided positive feedback regarding users' expectation and ease of use.

Although the intelligent mechanism here proposed is general enough for any kind of platform, for this first user study only desktop applications were considered. Since desktop computers are likely to be available at the user's commonplace work places, most end-users prefer desktop platforms to work and carry out authoring tasks. On the other hand, desktop authoring allows users to carry out far more expressive modifications that can provide further information regarding the authoring process. Nevertheless, modifying a mobile or PDA Web application by our system is certainly possible as long as there is an existing authoring tool to achieve such a task. The procedure is quite the same, since the user only has to make the changes and then send them to the server in order to be processed by our system. Modifying a mobile or PDA Web application from a desktop navigator is also possible, but probably this is such a less common task.

In our system, inferred information can be used to activate more general rules that can be triggered when the same modifications occur for more than one user. For example, it is possible to define general rules such as "If activation X is converted from pending into permanent for at least N users, then this rule can be included in every user profile as a general preference". This information is easy to obtain by our approach, since the expert system can be regarded as a database where new knowledge can be added and queries can be executed in order to mine the desired information from the knowledge stored. Additionally, other expert rules can be defined to detect problems concerning page design. For instance, it is possible to specify rules for detecting whether a change to a concrete element is carried out many times by different users, which could imply that some design problem may exist.

With regard to future work, we expect to improve the front-end part of our tool (i.e. our user interface) further. So far, NOTORIOUS enables end-users to see changes and switch on and off rule activation with the aim of being applied (or not) in future sessions. However, pending rules are turned into permanent only by the system whenever they have been detected at least three times depending on user modifications. Users cannot control the inference directly. Instead, the system carries out the best inference possible. A new improvement could consist of making rather interactive the inference process with respect to the possibility of allowing users to explicitly control rule transformations. Additionally, we plan to improve the system to detect more sophisticated customization cases, using the information already available in our expert system. This way, we expect to create general rules able to identify complex design problems that can be fixed automatically. In addition, an interesting add-on would consist in providing a specific tool for allowing users to easily author rules in our knowledge base. We also plan to carry out more in-depth tests on other Web applications by exploiting the previous inferred knowledge and including users interacting with other platforms (PDA, mobile, etc.).

Acknowledgements

The work reported in this paper is supported by the European Training Network ADVISES (Analysis Design and Validation of Interactive Safety-critical and Error-tolerant Systems), funded through the European Commission. Project number EU HPRN-CT-2002-00288.

References

R.Bandelloni, F. Paternò, C. Santoro, Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments, Proceedings EIS'07, Salamanca, LNCS Springer Verlag, March 2007.

Berti, S., Correani F., Paternò, F. and Santoro, C., (2004). The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels, Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, May 2004, pp.103-110.

Berti, S., Paternò, F. and Santoro, C. (2006). Natural Development of Nomadic Interfaces Based on Conceptual Descriptions. Lieberman, H., Paternò, F., and Wulf, V. (eds): End-User Development. Human Computer Interaction Series. Springer Verlag, pp 143-160.

Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R.C. "Automation and Customization of Rendered Web Pages." ACM Conference on User Interface Software and Technology (UIST), 2005, pp 163-172.

Cypher A. (1993). Watch What I Do: Programming by Demonstration. The MIT Press.

Davis, F.D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Quarterly, Vol 13, No. 3 (Sep. 1989), pp. 319-340.

Forgy, C. (1982). Rete. A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligent, 19, pp. 17-37.

Frank, M., Sukariviya, P., and Foley, J. (1995). Inference Bear: Designing Interactive Interfaces Through Before and After Snapshots. In Proc of the ACM Symposium on Designing Interactive systems. Ann Arbor, Michigan, August 23-25, pages 167-175.

JESS (2006). The Rule Engine for the Java^{TM} Platform. Http://herzberg.ca.sandia.gov/jess/.

Klann, M. (2003). End-User Development Roadmap. In Proceedings of the End User Development Workshop at CHI Conference. Ft. Lauderdale, Florida, USA. April 5-10.

Lieberman, H. and Liu, H. (2006). Feasibility studies for Programming in natural Language. Lieberman, H., Paternò, F., and Wulf, V. (eds): End-User Development. Human Computer Interaction Series. Springer Verlag, pp 459-474.

Lieberman, H., Liu, H., Singh, P. And Barry Barbara (2004). Beating Common Sense into Interactive Applications. Artificial Intelligence Magazine. Volume 25(4), pp. 63-76. Winter.

Lieberman, H., Paternò, F., and Wulf, V. (eds) (2006). End-User Development. Human Computer Interaction Series. Springer Verlag.

Lieberman, H. (2001). Your Wish is my Command. Programming By Example. Morgan Kaufmann Publishers. Academic Press, USA.

Macías, J.A., Puerta, A. and Castells, P. (2006). Model-Based User Interface Reengineering. HCI Related Papers of Interacción 2004. Jesús Lorés y Raquel Navarro (eds.). Springer-Verlag Volume, pp 155-162.

Macías, J.A., and Castells, P. (2005). Finding Iteration Patterns in Dynamic Web Page Authoring. Proceedings of the EHCI-DSVIS. Tremsbüttle Castle, Hamburg, Germany. July 11-13. Rémi Bastide, Philippe Palanque and Jörg Roth (Eds.). Lecture Notes in Computer Science, Volume 3425, pp 164 – 178. Springer-Verlag.

Macías, J.A. and Castells P. (2004). An EUD Approach for Making MBUI Practical. Proceedings of the First International Workshop on Making model-based user interface design practical. CADUI. Funchal, Madeira, Portugal. January 13.

Mori, G., Paternò, F. and Santoro, C. (2004). Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, August 2004, Vol.30, N.8, pp.507-520, IEEE Press.

Myers, B.A. (1998). Creating User Interfaces by Demonstration. Academic Press, San Diego.

Paternò, F. (2001). Model-Based Design and Evaluation of Interactive Applications. Springer Verlag.

Puerta, A.R.; Eisenstein, J. (1999). Towards a General Computational Framework for Model-Based Development Systems. Proceedings of the International Conference on Intelligent User Interfaces (IUI). ACM Press, New York.

Repenning, A. and Ioannidou, A. (2006). What Makes End-User Development tick? 13 Design Guidelines. Lieberman, H., Paternò, F., and Wulf, V. (eds): End-User Development. Human Computer Interaction Series. Springer Verlag, pp. 51-85.

Rode, J., Rosson, M.B. and Pérez, M.A. (2006). End-User Development of Web Applications. Lieberman, H., Paternò, F., and Wulf, V. (eds): End-User Development. Human Computer Interaction Series. Springer Verlag.

Rode, J. and Rosson, M.B. (2003). Programming at Runtime: Requirements & Paradigms for nonprogrammer Web Application Development. IEEE 2003 Symposium on Human-Centric computing Languages and Environments New York, pp. 23-30.

Szekely P. (1996) Retrospective and Challenges for Model-Based Interface Development. DSV-IS 1996: pp.1-27