Intelligent Support for End-User Web Interface Customization

José A. Macías[1] and Fabio Paternò[2]

[1] Universidad Autónoma de Madrid. Ctra. De Colmenar, Km. 15,
28049 Madrid, Spain
[2] ISTI-CNR. Via G. Moruzzi, 1,
56124 Pisa, Italy
j.macias@uam.es, fabio.paterno@isti.cnr.it

Abstract. Nowadays, while the number of users of interactive software steadily increase, new applications and systems appear and provide further complexity. An example of such systems is represented by multi-device applications, where the user can interact with the system through different platforms. However, providing end-users with real capabilities to author user interfaces is still a problematic issue, which is beyond the ability of most end-users today. In this paper, we present an approach intended to enable users to modify Web interfaces easily, considering implicit user intents inferred from example interface modifications carried out by the user. We discuss the design issues involved in the implementation of such an intelligent approach, also reporting on some experimental results obtained from a user test.

Keywords: End-User Development, Intelligent User Interfaces, Model-Based Design of User Interfaces, Programming by Example.

1 Introduction

Very often, customizing software applications implies extra knowledge and effort that some users cannot simply afford. Providing users with real authoring facilities is not yet as widespread as one would expect. Most of the existing approaches pay poor attention to end-users, and the ease of customization of commercial applications is still barely visible.

In general terms, the explicit customization of interactive applications requires considerable skill in programming and technology. Some preliminary studies indicate that these limitations in user development activities are not due to lack of interest, but rather to the difficulties inherent in interactive development [14]. Some development tools already offer support for high-level functionality, but most of these tools are not aimed at non-programmers.

Our research is aimed at addressing such problems by providing end-users with easy and automatic mechanisms to customize Web applications. Our research experience is in Model-Based User Interfaces [12] design combined with End-User

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-92698-6_37

J. Gulliksen et al. (Eds.): EIS 2007, LNCS 4940, pp. 303–320, 2008.

Development [6] techniques to help users interact with computers through intelligent WYSIWYG authoring environments [9], [10]. To this end, one of our main concerns is end-user development environments oriented to nomadic Web applications [3], which are Web applications accessible through a variety of platforms, including wireless devices supporting mobile users.

In this work, our effort is aimed at allowing users to provide modification examples of nomadic interfaces in such a way that the system be able to learn and generalize customizations automatically. From this point of view, our system is based on Programming by Example (PBE) mechanisms. Programming by Example [4], [7] is one of the most relevant efforts in EUD for obtaining a real trade-off between ease of specification and expressiveness. In our approach the user provides the system with an example of what s/he wants to modify by means of a standard authoring tool and then the system analyses the modifications at the server side for a given user and platform.

In particular, in the paper we report on some design issues addressed in our environment, combined with a further analysis of the type of reasoning that our system is able to apply. We also report on empirical system verification through an experiment carried out with real users. The paper is structured as follows. Section 2 introduces related work and discusses it. Section 3 describes our approach in further detail. Next, Section 4 reports on design and architectural issues. Section 5 provides further detail and introduces rule firing, describing an experiment carried out with real users with the aim of verifying the proposed approach. Lastly, Section 6 draws some conclusions and provides indications for future work.

2 Related Work

Intelligent rule-based systems have been traditionally used in Programming by Example research mostly due to the execution speed and simplicity they supply. Other complex machine-learning algorithms usually suffer from high error rates and low generalization in real time interaction with users.

Some early tools developed by Myers's group, such as Peridot [11] applied a rule-based approach. Peridot is more oriented to supporting user interface design and uses about fifty hand-coded Interlisp-D rules to infer the graphical layout of the objects from the examples. This type of system has the disadvantage of being subjected to rule-based heuristics that generalize from a single example, which implies that only a limited form of behaviour can be generalized since the system can only base its guess on a single example. More complex behaviours are either not treatable or must be created manually by editing the code generated by the PBE system. These types of systems are mostly focused on static knowledge and can be considered domain-dependent. In contrast, our system proposes an approach to build dynamically knowledge, in which a complete rule structure is created in order to consider different kinds of conceptual knowledge that can be updated from time to time through an evolving approach.

Recent systems such as AgentSheets [13] are examples of commercial EUD approaches for building intelligent interfaces. AgentSheets is a simulation environment that allows the user to create advanced simulation scenarios by defining intelligent agents and behaviour separately. AgentSheets combines PBE with graphical rewrite rules into an end-user programming paradigm. Like AgentSheets, our approach applies semantic rules for dealing with high-level behaviour. As pointed out by 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 creates a significant problem for building new behaviours from scratch, reducing significantly the scalability of a PBE approach as well. Additionally, in our approach we consider different levels of knowledge and behaviour, dividing rules and facts into different conceptual levels that will help achieve an in-depth analysis automatically, inferring with accuracy the user's intents in order to obtain an evolutionary approach.

Another related work is DESK [8], which uses domain knowledge for characterizing changes from a dynamically generated interface, making minimal assumptions about the final 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 [9]. DESK tracks and records information from user actions and builds a monitoring model specified in XML. This information is sent to the back-end application, which processes in turn the monitoring model and applies different heuristics by using domain knowledge. As a result of the inference process, the underlying models of PEGASUS (domain, presentation) are modified taking into account each change the user performs on the Web page. Our approach overcome the DESK's limitations by detecting user intents automatically, comparing original and modified interface logical specifications and with no need of having a specific authoring client application. In order to get maximal high-level domain independence, changes by users are obtained through processing directly a logical user interface description specified in TERESA XML [2]. By contrast, DESK is limited to deal with HTML code modifications, which are later processed to obtain meaningful information by means of fixed heuristics. We exploit the information provided by the logical interface description to obtain semantic information. The knowledge management is improved by defining different levels of knowledge that are applied to better characterize and obtain evolving knowledge for future inferences.

To summarise, the work presented in this paper provides a novel solution with respect to approaches such as DESK and AgentSheets because it applies reverse engineering tools able to build automatically descriptions at different abstract levels represented using TERESA XML, which is a domain-independent modelling language. Such semantic information is then exploited in out intelligent approach to supporting user customization.

3 Our End-User Approach

Our system supports a EUD framework intended to provide the user with an easy mechanism to freely customize Web interfaces.

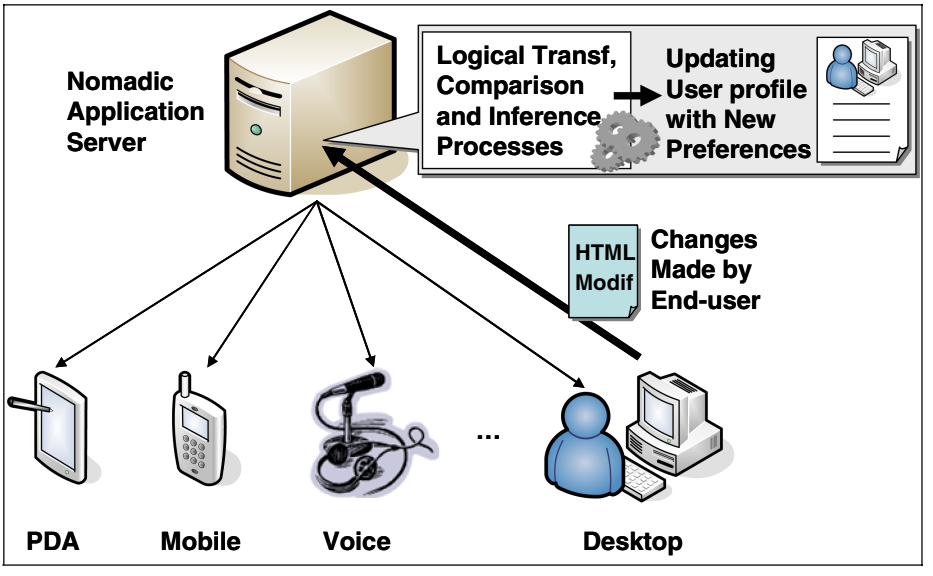

Fig. 1. Our approach can be used to customize user interfaces for different platforms. Users make changes to express their preferences and then upload the modifications onto the server, which infers customizations from the changes accomplished.

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

1. The Web server of applications generates an interface adapted to the platform accessed by the user, so s/he can access the application using a desktop computer, laptop, mobile, PDA and vocal interface.
2. The end-user navigates through the information and, at some point, s/he decides to modify something by using a standard Web authoring tool (such as Macromedia Dreamweaver) that supports modifications by direct manipulation of the interface elements.
3. Once the user has finished the changes, s/he sends the modified page to the server, by using a specific Web application in which s/he first needs to login.
4. 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 intro a logical description stored into a XML file, using the reverse mechanism developed by our group [1]. The resulting file contains the user interface description of the page 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 one.
   c) In the comparison process, the system also generates high-level information in order to find out meaningful information about the user's intents, and also to identify general user preferences.
   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 others previously generated.

5. The End-User Profile is then used to generate again the Web interface, taking into account preferences and personal customization. The system stores an End-User Profile for each user and platform, controlling which aspects of the generated interface could be significant for each one.

The most relevant information stored in the End-User Profile is the set of Interface Rules. Such rules are inferred from the logical descriptions' comparison and aim to reflect the knowledge acquired from the user's changes. The rules are used for driving the generation of the Web pages after the changes, customizing the Web presentation and navigation depending on the inferred preferences. The rules are based on knowledge acquisition algorithms and targeted at obtaining information regarding user intents in order to characterize some preferences for customization purposes. Such information can be modelled by means of both a knowledge base and a set of rules to be applied when new information about user modifications is identified.

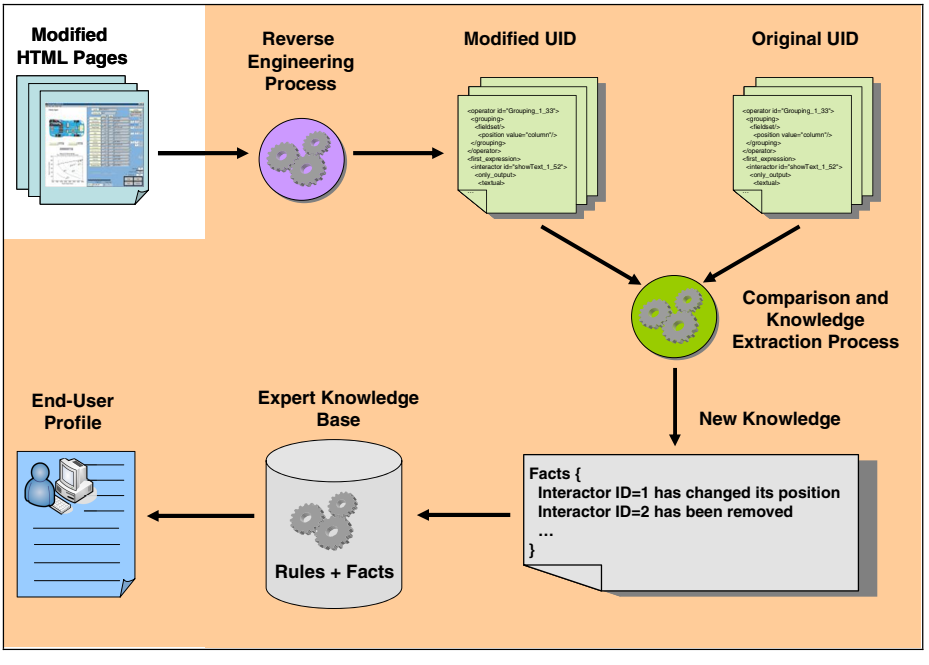

Fig. 2. Comparing both modified and original user interface descriptions, the system automatically extracts information in order to feed the expert system, generates the information to reason about and updates the user profile

The intelligent approach is implemented by using an expert system, where the knowledge can be modelled conveniently and the inference takes place more efficiently. Particularly, it supports 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 (i.e. evolutionary approach). In our approach, the facts represent the information coming from the user's modifications. This information is extracted by

comparing both modified and original logical interface descriptions (see Fig. 2). On the other hand, the rules are conditions used to get semantic information from the facts, that is, from the syntactical changes the user makes to the presentation and from other high-level information available in the expert knowledge base. The rules will reflect not only user changes but user information about the platform (Desktop, Mobile, and so on). By means of an evolutionary approach, continuous production and modification of facts helps the system refine the user's preferences and extract accurate information as interaction evolves.

In order to obtain greater precision and accuracy in the inference process, we use Jess [5], a Java framework which includes the Rete pattern matching algorithm for implementing rule-based (expert) systems. This algorithm was originally designed by Forgy at Carnegie Mellon University. It provides the basis for an efficient implementation of an expert system and is designed to sacrifice memory for increased speed.

3.1 Interface Knowledge Modelling and Construction

We base on TERESA XML language [2] for detecting changes on logical user interface descriptions. In this specification language, a user interface can be described at different abstraction levels. The concrete level is platform-dependent but implementation-language-independent, while the abstract level is also platform-independent. In both cases the user interface is composed of interactors and composition operators, indicating how to structure their composition. 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. Modifications affecting the concrete level provide syntactical knowledge, while those that effect the abstract level provide semantic knowledge as the abstract level identifies the type of basic task associated with the interface element. We consider both kinds of modifications in order to construct a knowledge structure aimed to feed the expert system with suitable facts, activate expert rules and produce user customizations efficiently.

The conceptual levels in which the knowledge is structured is crucial. Thus, we need to consider the following steps in defining that knowledge:

- **Defining base knowledge** containing basic definition about user, platform and the previous knowledge on user modifications. This is the information that always remains in the expert system and is updated from session to session.
- **Defining syntactic knowledge** that contains facts and rules triggered by syntactical modifications to presentation elements such as concrete interactors and concrete composition operators. (e.g. when a concrete interactor changes the value of its attributes). Furthermore, this level of knowledge deals with the syntactic context associated with the concrete composition operators, detecting for instance when a concrete composition operator has changed its colour, alignment, justification and so on.
- **Defining semantic knowledge** for dealing with semantic information by taking into account the syntactic information already created. The semantic level uses the abstract platform-independent elements associated with interactors and composition

operators. For instance, it identifies when the number of interactors changes in one possible composition (i.e. ordering, hierarchy relation or grouping). The semantic level also constructs presentation context, that is, contextual information extracted from the surrounding elements where a change took place in the graphical interface. Presentation context allows the creation of expert rules based on contextual information that can be applied more than once.

- **Defining expert rules** for dealing with further semantic aspects and characterizing user intents. The main goal of this level is to define both syntactic and semantic customization rules that will be deployed using the underlying knowledge available for the previous levels. Syntactic customization rules detect changes concerning concrete interactors and concrete composition operators, whereas semantic customization rules detect high-level changes affecting interactors and composition operators. For dealing with semantic customization rules, presentation context needs to be considered.

Knowledge construction is carried out progressively from the lowest levels to the highest ones. The knowledge constructed at lowest levels is basically composed of syntactic information automatically generated by the system. This information comes from the comparison of the specification of the concrete user interfaces before and after the user's changes and is related to the elements that the user implicitly manipulates when authoring a nomadic presentation. These elements are the concrete interactors and mostly indicate platform-dependent interaction techniques of different type (for instance, in a graphical desktop system concrete interactors can be Radio Button, List Box, Test Link, Button, Input Text and so on). All the changes concerning concrete interactors are added as syntactic knowledge in order to populate the expert system with detailed information about the type of concrete interactor, its implicit properties and so on. In concrete interface specifications, concrete interactors are composed through specific operators, in order to create relationships between different elements that will be presented for a platform and user. 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.

On the other hand, the system extracts presentation context that is based on the abstract specification of the interface, which is platform-independent and hence useful in order to get high-level contextual information about the presentation. This allows defining more general rules that can be applied to similar presentation contexts more than once. The abstract information of both interactor and composition interactor is managed by the semantic level of the expert system. Actually, this information can be regarded as a knowledge add-on that is based on the syntactic information already added by the syntactic level. The semantic level is responsible for detecting when an interactor is moved from a composition operator to another, or when it is deleted or removed, generating knowledge that can even affect the task model of the application. The semantic level is also responsible for extracting presentation context, and then adding it to the system as semantic knowledge. However, the first and foremost goal of the semantic level is to populate the system with information that will be deployed by the expert level, in order to carry out generalization in applying advanced customization rules.

4 The Software Architecture

Our system was originally conceived as a client-server architecture, where two principal processes run on the client and the server side and communicate one another to carry through the approach here presented.

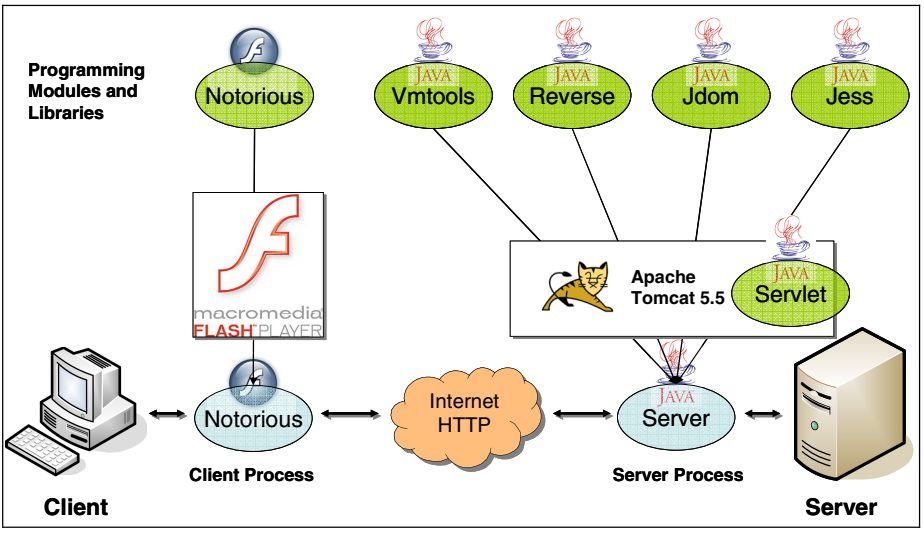

Fig. 3. The architecture of the system is mainly composed of a client and a server side, where two different processes run and communicate one another. The front-end sends to the server process the changes to be processed at the back-end of the application.

Fig. 3 depicts how the system is structured. At the client side, a Macromedia Web application called Notorious is executed. This application communicates with a server process, which is exported as a HTTP service by means of the Apache Tomcat Web Server 5.5. The process is a Java Servlet that is installed on the port 8080 of the server. The client application mainly consists of a user interface intended to identify the user when s/he connects to the server and uploads the modified Web pages. It also manages the feedback coming from the server process and visualizes the information reported (i.e. rules inferred and also the user interface descriptions). This application processes, by means of a XML connector, the user model from the server, and visualizes and stores such information properly. When the user decides to send Web pages using Notorious, this client application accesses the server. Then, the server takes up the request from the client application and in turn processes it, storing the Web page and generating the Concrete User Interface corresponding to the file that has been sent. Additionally, the Server routine compares both Concrete User Interface files (the original and modified one) and calls the expert system module to create new knowledge and obtain feedback to be sent to the client application. In doing so, the Server comprises the following modules:

- **Vmtools** is a library used to compare and obtain the differences from two XML files. The library comprises different classes and objects that can be used from a Java program. In our approach, this library was useful in order to compare the logical descriptions of the interface (the modified and the original one) and easily process the modifications by which the expert system is fed.
- **Reverse** is a Java library developed by our group which concerns the reverse-engineering routines for transforming HTML code into a logical user interface description called Concrete User Interface. Different methods are used to tidy and transform the code properly.
- **Jdom** is a Java library that comprises the routines used by Vmtools library. It is used to manipulate XML code and deal with XML-tree operations easily.
- **Jess** is the Java library used to deal with the expert system implementation. This library includes classes and objects to manipulate the inference engine called Rete algorithm, as well as the methods to activate, trace and deal with facts and rules in a nondeterministic way.

Additionally, the system uses the standard Java and Servlet libraries included in the standard edition of Java. Servlet routines are used to program different Web services in Apache Tomcat, so they can be regarded as a library as well.

5 Verification and Experimental Results

After the design of the system, one of our principal aims was to test the approach implemented. To this end, we carried out an experiment in order to check and obtain feedback on the methodology here proposed.

This experiment was mainly motivated by the need to measure the proposed rule-based approach. The test was aimed at detecting meaningful reactions of the system according to the user's modifications for a specific nomadic application. We recruited 11 participants from our institution, with heterogeneous scientific backgrounds, and asked them to freely customize a desktop Web museum application.

Based on different cases of use previously studied and analyzed, the expert system was programmed containing different kinds of expert rules, which can be divided into syntactic customization rules and semantic customization ones, as explained in Section 3.1. Furthermore, each rule has to be triggered at least three times to be considered a permanent customization, which the user can still turn on or off for future applications. In particular, a total of 14 syntactic customization rules and 10 semantic ones were created, with the intention of activating them according to the modifications performed by end-users. These included syntactic customization rules for detecting changes in text style preferences, interaction widgets and composition structures such as forms and fieldsets. On the other hand, semantic customization rules were also defined in order to deal with changes involving transformation, deletion and insertion of interactor groupings, as well as changes affecting composition operators that involve interactor repositioning. These reflect end-user preferences in navigation, ordering and hierarchical structure customization.

Additionally, the system was programmed to detect both user-dependent and user-independent customization. The user-dependent rules concern preferences associated

with a specific user and have been described previously. As for detecting user-independent preferences, the system checks whether the same rule is triggered by more than one user. User-independent tailoring helps define general changes to presentations for all users whenever the same rule is triggered by at least more than 5 users. At this point, the rule appears in every user profile and can be individually turned off whenever one user does not accept the changes.

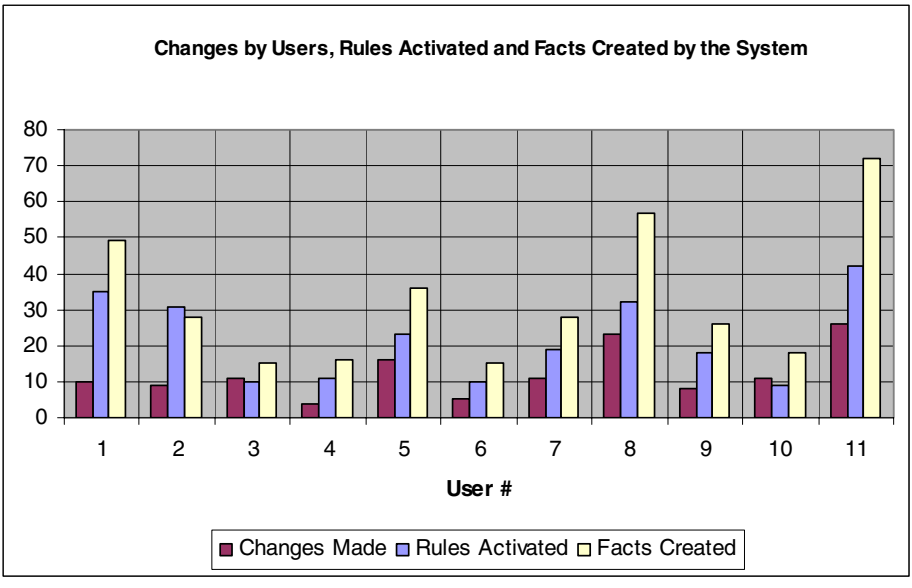

Fig. 4. The system's response to user changes, where the number of changes made and rules activated are shown, along with the facts automatically created by the system throughout the experiment with 11 users

Fig. 4 shows the relation between the number of changes, the facts generated and the rules activated for each user during the experiment. At first sight, it seems clear that the more the changes made, the more facts and rules are activated. However, this relation is not always as linear as one might expect, since it mostly depends on the complexity of the changes performed. In the case of user #2, for instance, one can see that the number of changes is lower with respect to other users, but the number of facts and rules activated is instead higher. This is due to the fact that user #2 made a total of 9 changes, but all involved complex effects. These entail 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 presentation context. In addition, the rules that had to deal with such changes were even more complex than simple syntactic ones, so that a chain of rules had to be activated to correctly detect the changes made by this user. In 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 aspects. In any case, such complexity does not at all affect the system's performance and throughput.

In addition to semantic and syntactic rules, we also considered the number of times each type of change was made by the user. A customization is applied when a rule is triggered three or more times. Otherwise, the customization is considered pending for the time being. This mechanism helped us to classify pending and permanent customizations depending on their rule-activation frequency. From the total rule activations measured during the user sessions and depicted in Fig. 4, 80% corresponded to syntactic customization rules, whereas only 20% corresponded to semantic customization ones. Regarding syntactic customization rule activations, 64% can be considered pending, whereas only 36% were permanent. With respect to semantic customization rules, only 9% of activations were permanent, whereas 91% were considered pending.

5.1 Rule Activation

In the experiment, rules were activated by following different steps. Let us examine a piece of the output extracted from the expert system for one of the user tests, illustrating how rules are activated and detected by the system.

1)Change detection and contextualization	`==> f-1 (MAIN::change (ID C1) (concrete_interactor Text Show_museum_info2) (change_type font_style_change bold)`
	`==> f-2 (MAIN::syntactic_context (ID SC1) (change C1) (from Presentation 2 FieldsetColumn 1) (above null) (below GraphicalLink 1 ) (user andrea) (platform Desktop))`
	`==> Activation: MAIN::syntactic_change : f-2 ...`

2)Syntactic customization rule activation	`FIRE 19 MAIN:syntactic_change f-2`
	`==> f-57 (syntactic_change_fact (syntactic_customization_rule6) (change C1) (syntactic_context SC1))`
	`==> Activation: MAIN:: syntactic_customization_rule6 : f-57, f-54, f-51, f-44, ...`

3)Pending and permanent rule activation	`FIRE 20 MAIN: syntactic_customization_rule6 : f-57` **`Pending Syntactic Customization (fired 1 times): Text style for Description Interactor will be bold`**

```
FIRE 21 MAIN:
syntactic_customization_rule6 : f-54
Pending Syntactic Customization (fired 2
times): Text style for Description
Interactor will be bold
FIRE 22 MAIN:
syntactic_customization_rule6 : f-51
Permanent Syntactic Customization
(triggered more than twice): Text style
for Description Interactor will be bold
```

The output above has been divided into 3 different parts. The first part corresponds to the change detection process. This information is directly supplied by an algorithm that compares the logical descriptions of the interface (original and modified Concrete User Interface files) and extracts information about what has changed. Consequently, the first fact is added to the system (f-1), reflecting the change (font text style has changed to bold) as well as the concrete interactor affected (Text element called Show_museum_info2). In addition, the syntactic information about the change is also created as fact number 2 (f-2), reflecting the context of the change (in Presentation 2, in FieldsetColumn 1, where above there is nothing and below there is the GraphicalLink 1 element) and the platform and user who made the change (user Andrea on platform Desktop). This change activates an internal rule called syntactic change that deals with the previous information and tries to find a suitable match for the rule to be applied (either syntactic or semantic customization rule). For this case, the second part of the output shows that the system has detected a syntactic customization since the change made is likely to be considered syntactic (a text style has changed). Thus, a new fact has been created (f-57) that relates the change (C1), the syntactical context (SC1) and the syntactic customization rule to be activated (customizaton_rule6). The syntactic customization rule number 6 deals with text style changes, and will be activated for the current fact (f-57) as well as for others which correspond to the same change and syntactic context (f-54, f-51, f-44, …). The third part of the output depicts the activation of customization rule number 6 for each change (fact) previously specified in the expert system. In this way, fact f-57 triggers a pending syntactic customization rule for a description interactor (the Text concrete interactor). This pending rule is triggered again for a different fact (f-54). Then, at the third matching (fact f-51), the pending customization rule was turned into a permanent one. This means that the description interactor, in the context observed (in this case the first occurrence at the beginning of a page), will appear in bold style. Consequently, this preference will be included in the user profile and can be turned off later on by the user.

The detection of semantic rules implies a similar sequence of facts creation and rules activations. In contrast, semantic rules require identification of the presentation context. The following output shows an example extracted from the user test.

```
FIRE 5 MAIN:syntactic_change f-14
==> f-27 (syntactic_change_fact (presentation_context
PC1) (change C7) (syntactic_context SC7))
==> Activation: MAIN::semantic_change : f-27 ...
FIRE 22 MAIN:semantic_change : f-27
==> f-36 (Presentation_Context (ID PC1) (Change_type
Insertion) (From Grouping FieldSet Grouping FieldSet)
(Above Navigator GraphicalLink Navigator GraphicalLink)
(Below null))
==> Activation: MAIN::semantic_customization_rule1: f-36
FIRE 23 MAIN: semantic_customization_rule1 : f-36
Pending Semantic Customization (fired 1 times):
Navigational Preferences have been changed by user
(inserted Navigator)
```

The piece of output above shows how initially the system mapped (f-14) a syntactic change (C7) to the context (SC7). Later on (FIRE 22), the system realised that such change regards the insertion of a navigational element, which is an interactor, and has semantic implications for the system. To this end, a new element appears (presentation_context PC1), identifying that a presentation context is needed in order to correctly identify this change. This causes the creation of a new fact (f-27) that involves semantic changes. Next, an internal rule (semantic_change) is called in order to extract the presentation context for such change. The presentation context is created in the form of a new fact (f-36), which reflects the context of the change in terms of abstract elements (Grouping, Navigator, and so on). Lastly, semantic rule number 1 is activated by means of the creation of the previous fact (FIRE 23), and thereby activates a pending rule once the presentation context has been successfully matched. It is worth noting that this customization reflects the fact that the user decided to change the navigational structure by adding a new navigator (a link).

5.2 Comparative Example

Fig. 5 shows one of the pages of the marble museum used in the user test (window at the top), as well as three pages (at the bottom) corresponding to three different modifications made by three different users. Although there are some similarities between some of the changes, the modifications differ from one another significantly. The dotted text box near each window describes the most important changes effected by each of the three users. Let us see in detail how the system reacts to each change for each modified page in Fig. 5.

In the first modified page (#1), the main change is the addition of a grouping consisting of a new navigational set inserted on the top of the page. This action stems from the fact that the user copied and pasted a fieldset, containing the navigational links of the home page of the museum, into each page with the idea of navigating everywhere from every page without the need to go back to the home page. In this case, the system activates different pending semantic customization rules, since the change mainly affects a grouping composition operator and thus can be considered a semantic change rather than a syntactic one. The system's reaction to such change

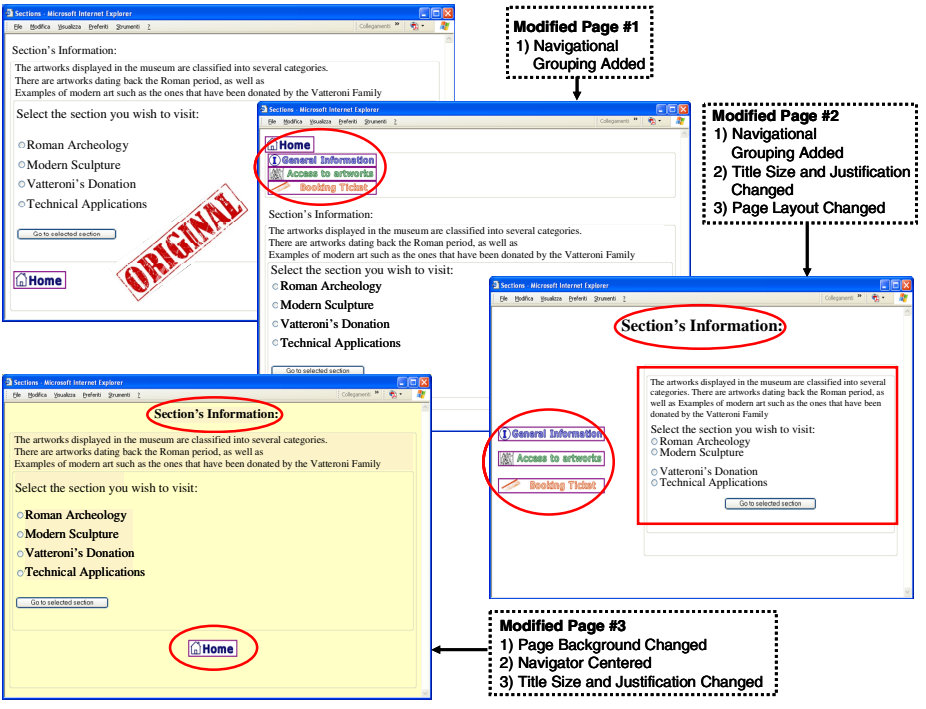

Fig. 5. Screenshots of 3 different pages modified by users during the test. The original page is at the top, whereas the other three windows depict the diversity of modifications made by users.

appears automatically specified by the system as rule firing numbers 9, 13 and 17. These can be summarized as follows:

```
FIRE 9 MAIN:semantic_customization_rule6 : f-22
Pending Semantic Customization (fired 1 times): New
Interactors have been added to an existing Grouping by
user (Grouping Add-on) ...
FIRE 13 MAIN:semantic_customization_rule6 : f-24
Pending Semantic Customization (fired 2 times): New
Interactors have been added to an existing Grouping by
user (Grouping Add-on) ...
FIRE 17 MAIN:semantic_customization_rule6 : f-26
Permanent Semantic Customization (triggered more than
twice): New Interactors have been added to an existing
Grouping by user (Grouping Add-on)
```

The above output describes how the system detected a semantic customization rule related to a grouping change (customization_rule6). This process is carried out after analysing the change in the grouping composition operators and obtaining the presentation context involved in each change. Lastly, the system converted the pending rule into a permanent one. This is due to the fact that the user decided to

make the same change three times to more than one Web page, as shown in firing numbers 9, 13 and 17.

In the second modified page (#2), the user made different changes, some involving semantic changes and others only syntactic ones. The semantic changes were related, once again, to the movement of elements as well as changes in grouping composition operators. In this case, one can see how the user decided to copy and paste the navigational set from the home page into the modified one, and then made changes to the page layout as well. Three different semantic customization rules were activated. These rules were applied to changes associated with modification, movement and distribution of interactor groupings. Additionally for this user and presentation, some syntactic changes were detected, meaning that the user also decided to change the text size and justification for the description element. The following rules were eventually activated:

```
FIRE 16 MAIN::semantic_customization_rule4 : f-43
Pending Semantic Customization (fired 1 times): Grouping
movement into another by user (Grouping Movement) ...
FIRE 18 MAIN::semantic_customization_rule5 : f-44
Pending Semantic Customization (fired 1 times): Grouping
layout has been set to horizontal by user (Grouping
Distribution) ...
FIRE 22 MAIN::semantic_customization_rule6 : f-46
Pending Semantic Customization (fired 1 times): New
Interactors have been added to an existing Grouping by
user (Grouping Add-on)
```

In this case, three different semantic customization rules were activated (4, 5 and 6). These rules deal with detecting changes in, and movement and distribution of, groupings. Like in the first modified page, the system firstly detected the change, obtained the syntactic and presentation context and then detected a matching in the presentation context that triggered this pending rule multiple times. This time, no pending rule was turned into a permanent one since the user only decided to make the change more than twice on different contexts, hence the system did not consider it to be the same change.

Additionally for this user, some syntactic changes were also performed, leading to the following output from the system:

```
FIRE 31 MAIN:syntactic_customization_rule6 : f-36
Pending Syntactic Customization (fired 1 times): Text
Font justification for Description Interactor will be
centred
FIRE 33 MAIN: syntactic_customization_rule1 : f-35
Pending Syntactic Customization (fired 1 times): Text
Size for Description Interactor will be 14
```

The output above reflects that the user also decided to change the text size (to 14 points) and justification for the description interactor (Text) on the top of the page. In this case, two syntactic pending customization rules were activated (6 and 1) that deal

with text justification and size, respectively. As before, no permanent execution was considered for such changes either.

The last page (#3) modified by the user contained mostly syntactic changes: only one navigator that the user centred, the description element at the page top, which the user also centred and enlarged in size, and a change to the page background colour. For these, the reaction of the system was to activate syntactic customization rules as follows:

```
FIRE 10 MAIN:syntactic_customization_rule5 : f-27
Pending Syntactic Customization (fired 1 times): Page
Background will be #FCF4CD ...
FIRE 11 MAIN:syntactic_customization_rule5 : f-26
Pending Syntactic Customization (fired 2 times): Page
Background will be #FCF4CD ...
FIRE 12 MAIN:syntactic_customization_rule5 : f-22
Permanent Syntactic Customization (triggered more than
twice): Page Background will be #FCF4CD ...
FIRE 13 MAIN:syntactic_customization_rule1 : f-25
Pending Syntactic Customization (fired 1 times): Back
Graphical-Link Navigator alignment will be centred ...
FIRE 14 MAIN:syntactic_customization_rule1 : f-24
Pending Syntactic Customization (fired 1 times): Text
size for Description Interactor will be 18 ...
FIRE 16 MAIN:syntactic_customization_rule4 : f-23
Pending Syntactic Customization (fired 1 times): Text
font justification for Description Interactor will be
centred
```

As the previous cases, the system firstly processed the changes and then triggered the syntactic customization rules for this case (5, 11, 1 and 4). The first syntactic customization rule concerned the change in the background, as the user decided to set another colour. As one can see, this pending rule became permanent since the user carried out this same change to more than two pages. This means this customization was stored in the user profile. Additionally, the user decided to centre the back navigational link at the bottom, which triggered the syntactic customization rule 11. Some other temporary customization activations were carried out as well: these affected text style and justification and concerned the description interactor at the page top. These last changes were not considered permanent, since the user decided to perform them less than three times.

6 Conclusions and Future Work

Customization of software artefacts is commonly considered as an activity that requires specialized knowledge that most end-users do not have. This is mainly due to the fact that authoring environments require manipulating programming languages and abstract specifications. Although much progress has been made by commercially

available development tools, most of them lack not functionality, but rather ease-of-use [15].

Our approach overcomes such limitations and provides easy and efficient mechanisms based on Programming by Example techniques, where the user provides the system with example changes and the system generates customizations that will be applied automatically in future interaction. More concretely, the user carries out changes to applications generated by a server for a specific platform using any commercial authoring tool, and then s/he sends the modified pages to the server. Lastly, the system processes all the pages and tries to infer meaningful customizations to be applied in the future. Instead of forcing end-users to learn 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 report on a detailed example of activations extracted from a user test, which has been introduced and further commented. Although only permanent activations were taken into account for a specific user and platform, more general information can be extracted. Collective knowledge can be deployed to detect general preferences by simply matching coincidences from more than one user. In the previous examples some changes can be understood to be general semantic customizations when the same rule is activated consistently by different users. For instance, as depicted in Fig. 5, modifications #2 and #3 reflect that both users made changes affecting the description element located at the page top, specifically changes concerning font size and justification. Independent of the user and platform, this information can be used to activate more general rules that can be triggered when the same modifications occur for more than one user. Moreover, general rules could be defined, for example "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 queries can be executed in order to mine the desired information from the knowledge stored. Additionally, other high-level rules can be defined to detect problems concerning page design. We are carefully studying and analysing such issues in order to further improve our system.

Acknowledgments. The work reported in this paper ha been supported by the European Training Network ADVISES, project EU HPRN-CT-2002-00288, and by the Spanish Ministry of Science and Technology (MCyT), projects TIN2005-06885 and TSI2005-08225-C07-06.

References

1. Bandelloni, R., Mori, G., Paternò, F.: Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments. In: Proceedings of Engineering Interactive Systems, Salamanca (March 2007)
2. Berti, S., Correani, F., Paternò, F., Santoro, C.: The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Leveles. In: Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, pp. 103–110 (May 2004)

3. Berti, S., Paternò, F., Santoro, C.: Natural Development of Nomadic Interfaces Based on Conceptual Descriptions. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End-User Development. Human Computer Interaction Series, pp. 143–160. Springer, Heidelberg (2006)
4. Cypher, A.: Watch What I Do: Programming by Demonstration. MIT Press, Cambridge (1993)
5. Jess. The Rule Engine for the Java™ Platform, http://herzberg.ca.sandia.gov/jess/
6. Lieberman, H., Paternò, F., Wulf, V. (eds.): End-User Development. Human Computer Interaction Series. Springer, Heidelberg (2006)
7. Lieberman, H. (ed.): Your Wish is my Command. Programming By Example. Morgan Kaufmann Publishers, Academic Press, USA (2001)
8. Macías, J.A., Puerta, A., Castells, P.: Model-Based User Interface Reengineering. In: Lorés, J., Navarro, R. (eds.) HCI Related Papers of Interacción 2004, pp. 155–162. Springer, Heidelberg (2006)
9. Macías, J.A., Castells, P.: Finding Iteraction Patterns in Dynamic Web Page Authoring. In: Bastide, R., Palanque, P., Roth, J. (eds.) DSV-IS 2004 and EHCI 2004. LNCS, vol. 3425, pp. 164–178. Springer, Heidelberg (2005)
10. Macías, J.A., Castells, P.: An EUD Approach for Making MBUI Practical. In: Proceedings of the First International Workshop on Making model-based user interface design practical CADUI, Funchal, Madeira, Portugal, January 13 (2004)
11. Myers, B.A.: Creating User Interfaces by Demonstration. Academic Press, San Diego (1998)
12. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer, Heidelberg (1999)
13. Repenning, A., Ioannidou, A.: What Makes End-User Development tick? 13 Design Guidelines. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End-User Development. Human Computer Interaction Series, pp. 51–85. Springer, Heidelberg (2006)
14. Rode, J., Rosson, M.B., Pérez, M.A.: End-User Development of Web Applications. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End-User Development. Human Computer Interaction Series. Springer, Heidelberg (2006)
15. Rode, J., Rosson, M.B.: Programming at Runtime: Requeriments & Paradigms for nonprogrammer Web Application Development. In: IEEE 2003 Symposium on Human-Centric computing Languages and Environments, New York, pp. 23–30 (2003)