Finding Iteration Patterns in Dynamic Web Page Authoring

José A. Macías and Pablo Castells

E.P.S. Universidad Autónoma de Madrid
Ctra. de Colmenar, km. 15
28049 – Madrid – Spain
+ 34 91 497 22{41, 84}
{j.macias, pablo.castells}@uam.es
http://www.ii.uam.es/~{jamacias, castells}

Abstract. Most of the current WWW is made up of dynamic pages. The development of dynamic pages is a difficult and costly endeavour, out-of-reach for most users, experts, and content producers. We have developed a set of techniques to support the edition of dynamic web pages in a WYSIWYG environment. In this paper we focus on specific techniques for inferring changes to page generation procedures from users actions on examples of the pages generated by these procedures. More specifically, we propose techniques for detecting iteration patterns in users' behavior in web page editing tasks involving page structures like lists, tables and other iterative HTML constructs. Such patterns are used in our authoring tool, DESK, where a specialized assistant, DESK-A, detects iteration patterns and generates, using Programming by Example, a programmatic representation of the user's actions. Iteration patterns help obtain a more detailed characterization of users' intent, based on user monitoring techniques, that is put in relation to application knowledge automatically extracted by our system from HTML pages. DESK-A relieves end-users from having to learn programming and specification languages for editing dynamic-generated web pages.

1 Introduction

Since its emergence in the early 90's, the WWW has become not only an information system of unprecedented size, but a universal platform for the deployment of services and applications, to which more and more activity and businesses have been shifting for more than a decade. The user interfaces of web applications are supported by a combination of server-side and client-side technologies, such as CGIs, servlets, JSP/ASP, XML/XSLT, JavaScript, Flash, or Java applets, to name a few. For most applications, client-side GUI facilities are not enough or, as in the case of applets, have unsolved portability problems. Architectural characteristics of web systems typically bring about an inherent need for not only creating web pages that contain interactive interface components, but for generating the pages dynamically on servers or intermediate web nodes. Moreover, using as simple client-side technologies (i.e. client-side requirements) as possible is usually the preferred approach for businesses

for which reaching the widest audience possible is a critical concern. As a matter of fact, dynamic pages make up the vast majority of the current web ([23] gave an estimate of 80% in year 2000).

With dynamic web pages, user interfaces can be generated whose contents, structure, and layout are made up on the fly depending on application data or state, user input, user characteristics, and any contextual condition that the system is able to represent. However the development of dynamic pages is a quite complex task that requires advanced programming skills. The proliferation of tools and technologies like the ones mentioned above require advanced technical knowledge that domain experts, content producers, graphic designers or even average programmers usually lack. Development environments have been provided for these technologies that help manage projects and provide code browsing and debugging facilities, but one still has to edit and understand the code. As a consequence, web applications are expensive to develop and often have poor quality, which is currently an important hurdle for the development of the web.

The research we present here is an effort to leverage these problems by developing Programming By Example (PBE) techniques [5, 9, 16] to allow regular users, with minimum technical skills, to edit dynamic web pages. Our work can be situated in the End-User Development (EUD) area [19], concerned with enabling a non-expert user to deal with a software artifact in order to modify it easily. Many WYSIWYG tools are available today for the construction of static HTML pages, but is it not clear how procedural constructs, like the ones needed for creating dynamic web pages, can be defined within the WYSIWYG principle. Our proposal consists of letting the user edit the product of the page generation procedures, i.e. one or more examples of the type of dynamic pages that will be generated at runtime, and build a system that is able to generalize the actions of the user on the examples, and modify the page generation procedure accordingly.

We have worked our proposal through the development of a purely WYSIWYG authoring tool, DESK [10, 11, 12], which supports the customization of page generation procedures in an editing environment that looks like an HTML editor from the author point of view. With DESK, users edit dynamic pages produced by an automatic page generation system; DESK keeps track of all user's actions on edited documents, finds a semantic meaning to the editing actions, and carries the changes to the page generation system. A differential aspect of our approach with respect to previous PBE techniques is the explicit use of an application-domain model, based on ontologies, to help characterise the user's actions in relation to system objects and interface components. Semantic relationships between application objects underlying HTML constructs are used by DESK to trace back the inverse path from generated pages up to the generation procedure.

In this paper we focus on the inference mechanisms by which DESK infers the user's intent, by means of data models and characterizations of user actions. A particularly interesting and complex problem to make sense of the user's actions is when the user manipulates complex layout structures made of tables, lists, trees, or combinations thereof. The need for these layout primitives is unavoidable in any but most trivial HTML pages and, when it comes to dynamic pages, they are often used in correspondence to application information structures. A specialized assistant, DESK-A, attempts to find out *iteration patterns* in the user behavior when s/he handles these

structures, in order to infer the user's intent and provide with assistance in addressing complex high-level tasks. An iteration pattern involves –and provides a means to correlate– a layout structure, application information structures, and a likely structure in user's actions. How to correctly identify and find the relation between these three parts of the equation is a problem addressed by the work presented here.

This paper is organized as follows: Section 2 describes how our system deals with iteration patterns as well as the metrology used for extracting and classifying different types of patterns. Additionally, an specific case of use will be presented and deployed throughout the paper in order to show how DESK-A works and finds out iteration patterns from user actions. Section 3 describes related work on EUD and PBE systems that mostly exploit user monitoring techniques. Finally, in Section 4, some conclusion will be provided.

2 Iteration patterns

Iteration patterns can be though of as a generalization of common user actions that can appear more than once, so that they can be used to apply similar behavior on future interaction. Iteration patterns help be able for the system to suggest the user to achieve cumbersome tasks on her behalf.

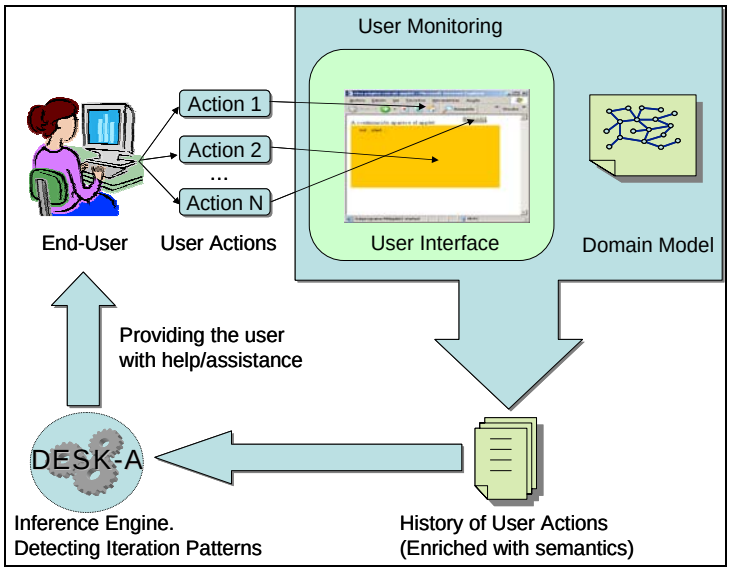

Fig. 1. Our approach. The end-user interacts with the system that extracts information from her actions. A domain model is in turn used to create a detailed history of user actions enriched with semantics from the domain model. Finally DESK processes all this information to detect high-level tasks on the monitoring model, in order to provide the end-user with assistance at the interaction.

In order to address iteration patterns, our approach needs the system to record the user's actions by building a specialized *monitoring model*. The monitoring model can be regarded as a built-in low-level task model, where all the actions the user achieves on the web interface are stored and enriched with add-on implicit information about the interface itself. This way, one of the advantages in using a monitoring model is that a semantic history of user actions can be built in real time. Therefore in our approach the system analyses and manages such history to find iteration patterns.

Fig. 1 shows how the system tracks the user's actions and then uses domain information to generate a semantic history. Such history is in turn added on with references of the interface's components as well as with internal annotations. The system also detects and models presentation structures like tables and selection lists. An inference engine (i.e. DESK-A) processes the history of user actions and detects iteration patterns than can be applied to assist the user. Finally the system provides the end-user with help and performs task as a user's surrogate.

2.1 Detecting iteration patterns

Detecting iteration patterns consists of analyzing the history of user actions (i.e. the monitoring model) to find out meaningful information about the user's high level tasks. To carry out this challenge, the system implements a set of heuristics for finding relationships between the user's actions and the interface's presentation elements (i.e. widgets) than are being manipulating by the end-user in the interaction. The system detects linear relationships between the geometry features of the widgets and, basically, divides interaction patterns into two different categories: *regular pattern* and *non-regular patterns*.

Regular patterns are meant to be iteration sequences on certain widget attributes that define linear relationships between the widget's features (such as table columns and rows, selection list items and so on), whereas non-regular patterns are meant to be iteration sequences without regular relationships (i.e. no linear relationships can be found out) between widget attributes, and they have to be tackled apart.

Regular patterns

Regular patterns are detected and processed by means of specialized heuristics called *Iteration Patterns Algorithms* (hereafter IP Algorithms). IP Algorithms are a set of algorithms specialized in studying widgets geometry and extracting specific properties about them. Such properties will help find suitable iteration masks for copying elements automatically from one widget into another, holding the same domain model properties and mappings.

Fig. 2 shows two snapshots of DESK environment where a transformation of widgets takes place. This example will be used throughout the paper to put into context the algorithms for dealing with iteration patterns. That figure depicts how the user is attempting to copy elements from a selection list into a table previously created. After a couple of intents, DESK asks the user for confirmation to transform the selection list into a table, and finally the tool accomplishes the transformation. Therefore, it results in removing the list and replacing it by a table which has the same number of items and internal domain model mappings.

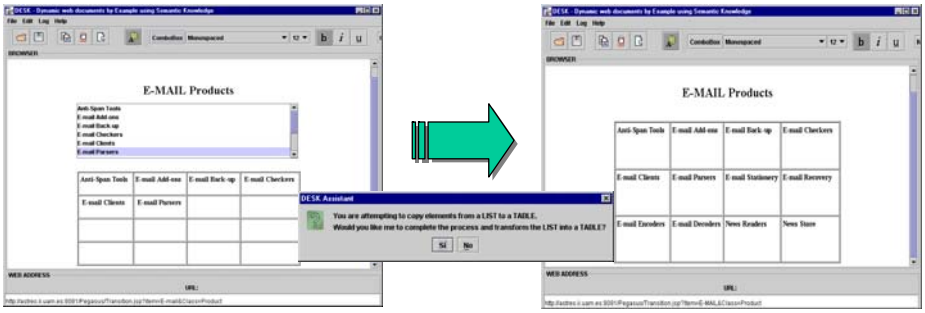

Fig. 2. Two snapshots from DESK. The scenario depicts an automatic transformation from a selection list into a table. The system detects the user's intent while s/he copies elements from a selection list into a table (left window), so the system suggests her (central message box) to convert the whole list into a table automatically (right window after the end-user has accepted the suggestion)

There are several IP Algorithms that can are applied depending on the type of the widget the system deals with. A sample code of one of these algorithms (inspired in Fig. 2) for managing transformation of tables and selection lists is as follows:

```
IP_Algorithm (Widget W1, W2, Set TG) {
  ColumnSequence      = A.getColumnSequence(W2);
  RowSequence         = A.getRowSequence(W2);
  ElemIndexSequence   = A.getElementIndexSequence(W1);
  ColJumpSet          = ColSequence.getColJumpSet();
  RowJumpSet          = RowSequence.getRowJumpSet();
  ColShiftSet         = BuildColShiftSet(ColumnSequence,
                        ColJumpSet,RowJumpSet);
  RowShiftSet         = BuildRowShiftSet(RowSequence,
                        ColJumpSet,RowJumpSet);
  Iterator            = BuildIterator(W2.getBounds(),
                        TG, ColShiftSet, RowShiftSet,
                        ElemIndexSequence);
  ...
  While (Iterator.hasNext()) {
     i = Iterator.getNexti(i);
     j = Iterator.getNextj(j);
     k = Iterator.getNextk(k);
     W2.setElementAt(i,j,W1.getElementAt(k));
  }
}
```

`W1` represents the source widget (i.e. a selection list) and `W2` is the destination one (i.e. a table). `TG` contains information about the widget's properties (i.e. number of fixed columns and rows). `A` is a set that stores information about actions that concern the

process of copying elements from one widget into another. This set is very useful in order to obtain common properties about the widget's manipulation sequence (for example, the column insertion sequence of elements into a table), as well as to obtain an abstract model about the widgets are being manipulated by the user throughout the interaction. Properties stored in `A` can be accessed by means of specialized methods:

- `A.getSize(Widget)`
- `A.getElementIndexSequence (Widget)`
- `A.getColumnSequence(Widget)`
- `A.getRowSequence (Widget)`
- `A.getElementAt(Widget,i[,j])`
- `A.getID(Widget)`
- `A.getClassName(Widget)`
- `A.getObjectName(Widget)`
- `A.getExistsRelation(Widget1,Widget2)`

The main goal of above operators is to provide the inference engine with information about the widget (and its properties), such as the size of a given widget, the insertion sequence of elements (index, column and row), the class and the object's names as they appear in the domain model, and the existing relationships between the source widget and the destination one. Therefore it is be able for the engine to build-in an iteration mask (`Iterator`) which provides with a mechanism for copying automatically elements from the source widget to the destination one, and adapting the properties of the destination widgets as the original one appears in the underlying models of the interface.

Fig. 3 depicts an example (based on Fig. 2) as the result of executing the above algorithm for copying elements from the selection list into the table. As shown in this figure, `ColumnSequence` and `RowSequence` sets store the insertion sequence achieved at each user step on the table. On the other hand, `ElemIndexSequence` stores the followed-up sequence of item selection on the selection list. Furthermore, the IP Algorithm calculates the column (`ColJumpSet`) and the row (`RowJumpSet`) jump's sets by processing `A`. The algorithm also detects whether the insertion is carrying out either on rows or columns by comparing both jump sets. This way, if `RowJumpSet` is greater (in size) than `ColJumpSet`, the insertion is achieved by iterating the rows, if not the insertion is achieved by iterating the columns. Otherwise, if both sets have the same size, special considerations has to be taken since there is a straight linear relationship between row and column on the insertion sequence. Next an increment mask is calculated for columns (`ColShiftSet`) and rows (`RowShiftSet`) by using an operator, namely $\Delta_{Average}$ defined in equation (1).

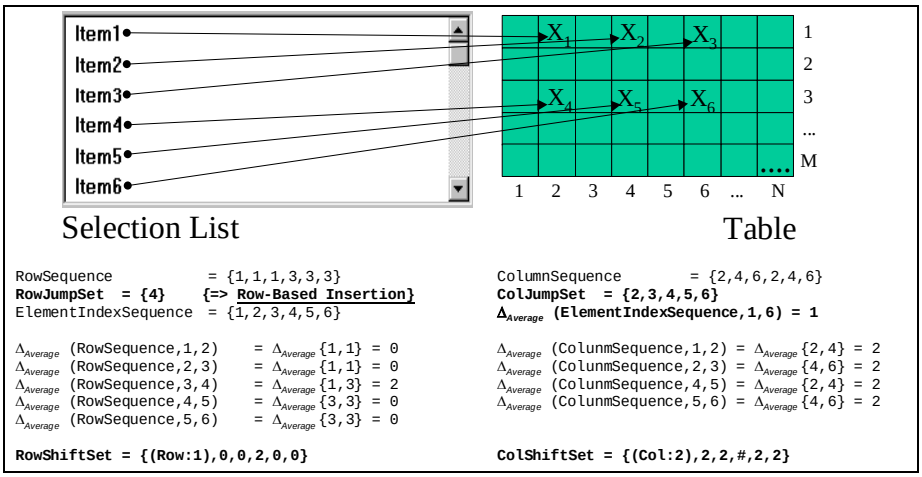

Fig. 3. Execution of an IP Algorithm for a table and a selection list. Before transforming the selection list intro a table, the system generates specific sets that store information concerning the rows and columns involved as well as the jump sequence's sets. Finally, a couple of iteration masks are calculated for both column and row, those intended to create an automatic iteration process for carrying out the transformation among widgets

$$\Delta_{Average}(x_1,x_2,x_3,\ldots,x_n)=\begin{cases}\dfrac{(x_2-x_1)+(x_3-x_2)+\ldots+(x_n-x_{n-1})}{n-1}, & n>1\\ 0, & n\le 1\end{cases}=\begin{cases}\dfrac{x_n-x_1}{n-1}, & \\ 0, & \end{cases} \quad (1)$$

Equation (1) represents an operator that calculates the average sequence of jumps. The operator is applied to obtain a couple of masks (`ColShiftSet` and `RowShiftSet` sets) which include the increments used in the loop for column and row jumps. Initial positions are also considered at loop starting (`Col:2` and `Row:1`), resulting in this case as follows: increasing 2 columns for the first time, jumping then two more rows (# in `RowShifSet` and 2 in `ColShiftSet`), next jumping 2 columns, and finally repeating the sequence all over again.

All these sets are finally used to create the iteration index to iterate though the widgets and to easily complete the iteration sequence previously calculated.

Fig. 4 shows examples of similar transformation processes, where different cases of tables with different types of insertion sequences are depicted. Those result in different values for each set depending on widget geometry. As shown, the algorithm can face correctly a great deal of cases where cut-in columns and rows are detected as a part of the iteration mask, using & symbol for row-based jumps and # one for colum-based jumps. Fig. 4 also shows a case where the iteration pattern is defined as an identity function (i.e. the same number of row jumps than column ones), finely detected by DESK-A as well.

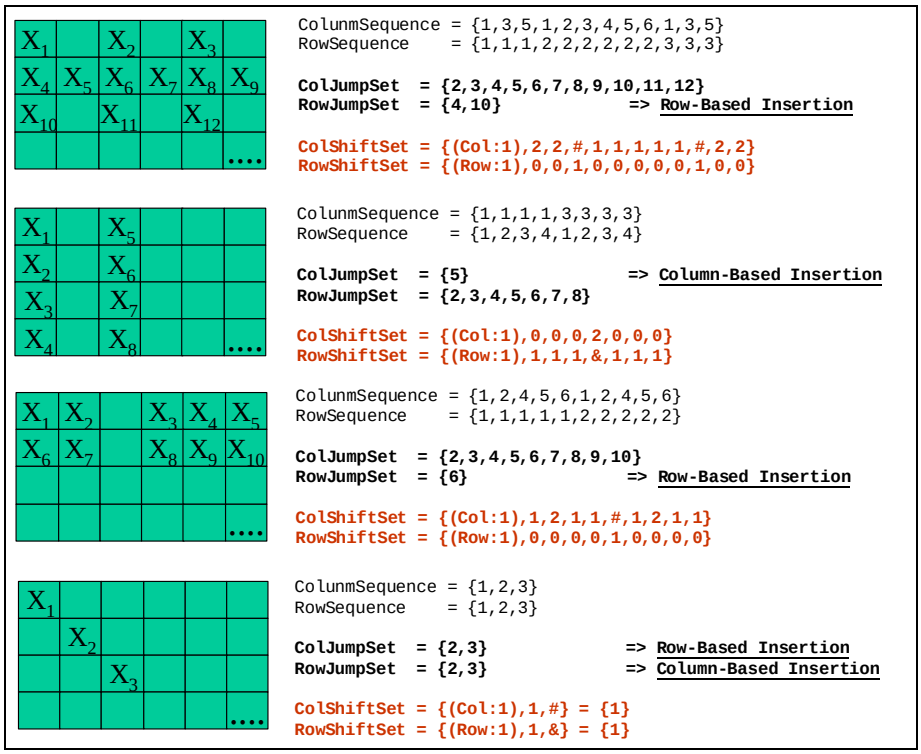

Fig. 4. Some examples of iteration patterns. These examples are generated using IP Algorithms, as it depicted in Fig. 3. So that Figure shows the iteration patterns for copying elements to the table as well as the sets generated for achieving the final transformation among the selection list and the table.

Non-regular patterns

Unfortunately it is not always able to create an iteration pattern that best fits a sequence started by the user. Actually, when the system is not able to find out linear relationships in iterative sequences on widget geometry then had-hoc or specific-purpose iteration patterns have to be considered.

The system faces the challenge of non-regular patterns by allowing the user to create a pool of pre-defined iteration patterns. Therefore s/he can customize the design and tell the system how to resolve the iteration in order to accomplish the transformation successfully. The pool of non-regular patterns can be included in the engine configuration, specifying the behavior for how the assistant (i.e. DESK-A) has to deal with each type of widget.

Fig. 5 shows an example of two iteration patterns that can be defined in the non-regular part of the DESK-A configuration file (see Section 2.2). This example reflects non-regular patterns where linear relationships are hard to find out, since there is not a

straight relationship among the widget's attributes (i.e. column and row insertion sequences), so that IP Algorithms cannot be applied directly.

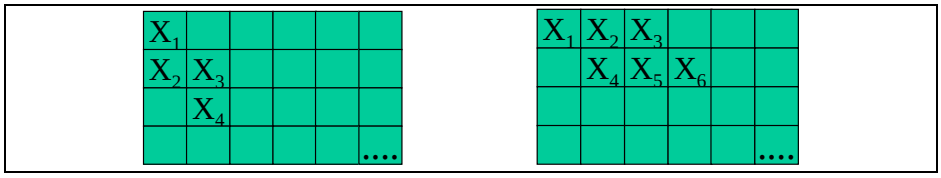

Fig. 5. Two examples of non-regular iteration patterns detected while copying elements from a selection list into a table. Here the relationship between rows and columns is not easy to find out since non linear sequences make IP Algorithms unlikely to deal with those cases. Anyway, those kinds of patterns are not usual to find in mostly common practice, so that a customized pool of predefined patterns is enough in order for the system to tackle non-regular patterns.

2.2 DESK-A

DESK-A (DESK-Agent) is a specialized inference assistant for finding out high level tasks (i.e. changes) related to the user's actions. DESK-A is based on the idea of the *Information Agent* [1] focused on *wrappers* paradigm [8, 16]. By contrast, in our approach the agent searches the monitoring model, which has an explicit semantic representation of the user's actions, rather than searching the HTML code directly. Therefore it is able for DESK-A to activate more complex heuristics [13] in order to find out transformation of presentation widgets, such as transforming a combo box into a table or transforming a table into a selection list. DESK-A can also infer more complex intents such as sorting a selection list and copying attributes from one table cell into another [13].

DESK-Agent detects and manages both regular and non-regular patterns by monitoring the user input. Basically, DESK-Agent comprises three main states:

- pre-activation: where the agent checks up the monitoring model for detecting high level tasks. This depends on the configuration set.
- activation: where the agent searches for specific widget values on the monitoring model once is pre-activated. Here, DESK-A analyzes in-depth the history of user actions and makes up different models for each widget involved in the interaction.
- execution: where the agent executes the transformations taking into account the values found at the activation step.

DESK-Agent searches the monitoring model for primitives that better fit the requirements defined at its configuration. The agent can be set-up by defining a configuration file at client-side. That configuration reflects the agent's behavior:

```
<TransformationHint>
  ...
  <widget type="List" changeTo="Table">
    <Condition action="Creation"
```

```
            widget="Table"  />
    <Condition action=”PasteFragment"
            from="Table" to="List" />
    <Non_Regular_Pattern_Pool>
      <Pattern  col_sequence=“1,1,2,2”
             row_sequence=“1,2,2,3”
             elem_sequence=“1,2,3,4”>
        <Resolve i=“from 1 to List.getSize(); i++1”
              next_col_sequence=“col[i],col[i]”
             next_row_sequence=“row[i],row[i+1]”
             next_elm_sequence=“elm[i]” />
      </Pattern>
      <Pattern col_sequence=“1,2,3,2,3,4”
             row_sequence=“1,1,1,2,2,2”
               elm_sequence=“1,2,3,4,5,6”>
        <Resolve
             next_col_sequence=“3,4,5,4,5,6,...”
             next_row_sequence=“3,3,3,4,4,4,...”
             next_elm_sequence=“7,8,9,10,11,...” />
      </Pattern>
      ...
    </Non_Regular_Pattern_Pool>
  </widget>
  ...
</TransformationHint>
```

The above code is a fragment of the DESK-A configuration, where <TransformationHint> elements are pre-activation directives the agent will check for arranging transformations between both widgets (<widget>), in that case a selection list (type="List") and a table (changeTo="Table"). Furthermore, DESK-A checks the creation status (action="Creation") of the table, as reflected in <Condition> elements, and analyses the copy sequence of elements (action=”PasteFragment") from the table into the selection list, making up dependences between the two widgets.

When all these prerequisites are satisfied, the agent executes transformation heuristics for detecting iteration patterns (see IP Algorithms at regular patterns Section) by selecting meaningful information from the monitoring model. Finally, the process results in transforming the widgets and keeping the same structure that holds the source widget by firstly asking the user for confirmation.

DESK-Agent also deals with non-regular patterns by allowing the user to create a pool of pre-defined iteration pattern (<Pattern> element inside <Non_Regular_Pattern_Pool>, at agent configuration code). This way DESK-A completes and resolves (<Resolve> element) the iteration sequence in order to accomplish the transformation successfully. Non-regular patterns are represented by using an indexed-construction, defining a for-like loop to iterate trough columns, rows and selection list items (<Resolve i=“from 1 to List.getSize(); i++1”). Furthermore DESK-A allows a numerical

representation of iteration sets (`<Resolve next_col_sequence = "3,4,5,4,5,6,..."`) for column, row and item indexes. This kind of specification becomes more natural and easy-to-understand for non-expert users.

3 Related work

One of the main limitations of early PBD systems that monitor actions [5] is that they are too literal. Some of these systems replay a sequence of actions at the keystroke and mouse-click level, without taking any account of context or attempting any kind of generalization. By contrast, later works are based on recording the user's actions at a more abstract level and making explicit attempts to generalize them. However, they have been demonstrated only in special, non-standard, often tailor-made software environments (see [9]).

Our approach aims at providing PBD techniques for domain-independent web-based interfaces, focused on dealing with high level tasks where different domains have been proposed in order to evaluate the level of trust of the tool. DESK-A is comparable to *Predictive Interfaces* [6] and *Learning Information Agents* [1] approaches, where the system observes the user while she interacts with the environment. These approaches assist the user by predicting and suggesting some commands to carry out tasks automatically.

Eager [5] is one of the most famous PBD attempts to bring together PBD and Predictive Interfaces. Eager is a Macintosh-based assistant which detects consecutive occurrences of a repetitive task, thus Eager proposes the user to complete the loop automatically. The loop is inferred by observing the user's actions. Eager needs the user to enter two consecutive tasks. This becomes a limitation since occurrences do not have to appear consecutive.

Familiar [22] overcomes some Eager's limitations but it also does not address the previous mentioned problem. Other works, like APE and SMARTEdit (both described in [9]) attempt to solve this difficulty by using *machine-learning* mechanisms in order to learn efficiently and rapidly when to make a suggestion and which sequence of actions to suggest to the user.

DESK-A analyses the monitoring model, regardless of the number and the sequence of user actions, and finds meaningful high-level information about the user's intents. DESK-A does not need to learn about the user's behavior and operates in-real time, without the necessity of *machine-learning* algorithms. As well as Familiar, DESK-A is domain-independent, but in DESK-A the domain information is used in order to enhance the inference process.

Some Lieberman's earlier work like Mondrian (described in [5]) was based on AppleScript to monitor the user and control applications, but it does not exploit its domain independence and high-level application knowledge. Similarly, in TELS [17] the system takes into account the user's actions, inferring iteration patters for addressing loops and conditions. TELS enables the end-user to meet the inference process, by asking for her opinion. In DESK-A, the system avoids the user from having to make assumptions of the inference mechanism, the PBE-based inference process is being as transparent as possible.

The use of data models was already present in PBE systems like Peridot [16] and HandsOn [3]. In a very simple form, Peridot enables the user to create a list of sample data to construct lists of user interface widgets. The data model in Peridot consists of lists of primitive data types. In HandsOn, the interface designer can manipulate explicit examples of application data at design-time to build custom dynamic displays that depend on application data at run-time. Our view in this regard is that it is interesting to lift these restrictions and support richer information structures. To this end, DESK-A uses ontology-based domain information for user intent characterization.

Concerning EUD related work, there has been interesting approaches during last two years. WebRevenge [20] makes the reverse path of a web page. WebRevenge generates a CCTT (ConCurTaskTrees, see [21]) based task model by analyzing the interaction as well as the web interface elements: tags and links. WebRevenge works together with TERESA [15], an abstract authoring tool for modeling applications from CCTT based task models. TERESA makes the straight engineering and WebRevenge the reserve one, in order to carry through an approach that allows for migration to different platforms. By contrast DESK is intended to assist the user while s/he interacts with the system rather than using it as a multi-modal generation system. DESK also takes into account user interaction and, in addition, an ontological data model as well as information extracted from the interaction. DESK uses a low-level task model rather than a CCTT based task model, where interface objects, domain information and user actions are embedded to enrich the semantic of the monitoring model.

Another interesting work also closely tied to EUD paradigm is LAPIS [14]. LAPIS is a web scraper that allows for rendering high conceptual level information by means of a pattern library using a simple web browser. LAPIS parsers the HTML and transforms tag and link level elements into conceptual representations that help end-user understand web information easily. As well as LAPIS, DESK parsers HTML and characterizes information from the page by using a data model. By contrast DESK enables the user to authoring the web page, so the user's actions are taking into account and analyzed as an important step of the process.

Personal Wizards [2] is also a great contribution to EUD as a PBE-based system. This approach tracks user actions and records interaction from an expert. The system generates a wizard in order to guide a non-expert user throughout the application. Personal Wizards are intended to help users configure Windows based applications easily.

4 Conclusions

We have presented an approach for inferring the user's intents in a WYSIWYG web-based authoring environment. Our approach is based on PBE strategies such as monitoring the user during the interaction. In addition our system features data models for enriching the user's actions with semantics. We have also reported on a model-based representation of user actions for detecting and processing iteration patterns.

Our authoring environment, DESK, features a specialized assistant, namely DESK-Agent detects the user's high level tasks throughout the interaction and executes heuristics to achieve transformations on presentation widgets for automating iterative tasks. DESK-A checks up on pre-activation condition and searches the monitoring model for obtaining meaningful information about widget characteristics. Therefore IP Algorithms exploit widget models to build an iterator for moving elements from one widget to another. This automates a great deal of transformation processes and provides the user with assistance to complete iterative tasks on her behalf. Furthermore, DESK-A can deal with non-regular patterns by defining a pool. This information is part of the agent configuration and can be set-up by the user. This allows to build more sophisticated patterns for automatically DESK-A to address.

The main idea of DESK-A is to provide with an assistant to help end-user carry out different, somehow hard to achieve, kind of actions in editing web pages. However, this mechanism can be extended for increasing productivity in user interaction by means of providing non-expert user with continuous assistance in her daily solving activities with computer applications as well as generating programming code without the necessity of learning programming or specification languages. This challenge can be carried through by exploiting the monitoring and semantic detection strategies. The main goal is to assist the user in a great deal of different scenarios, such as classical interface builders and toolkits, authoring tools for generating model-based user interfaces and, in general terms, programming environments. To this purpose, the abstract mechanism of pattern detection can be extended and new IP Algorithms can be created, in order for other kind of user intents to be detected by the system regardless of the domain and the interface used.

In general terms, DESK works according to EUD paradigm. The authoring tool helps end-user modify a web page generated by a previous application. This way the system generates a programmatic model of user actions as a high-level knowledge representation in order to finally modify the generation procedure of the web page. The end-user is continuously assisted while s/he interacts with the authoring tool. DESK ensures the *Gentle Slope of Complexity* [8] where expressiveness and complexity of use are balanced by the means of the WYSIWYG environment; low abstract representation imply low rate of expressiveness but also easy of use.

As DESK-A is based on an ontology-driven domain model [4], it works regardless of the domain applied. Several scenarios such as educational, travel and e-shopping have been used in order to evaluate the efficiency of the system. In [13] there is an experience carried out with end-users in order to evaluate the usability of DESK as an authoring tool. Although the comments of the results are out of the scope of this paper, the main outcomes of the experience pointed out the high satisfaction rate of the user with respect to the tool. This is due to the similarity that the users perceive with respect to ordinary web editing and browsing tools, but by contrasts with some add-on mechanisms that allow for editing dynamic web pages and assisting the user in accomplishing cumbersome tasks.

Acknowledgements

The work reported in this paper is being supported by the Spanish Ministry of Science and Technology (MCyT), project number TIC2002-1948.

References

1. Bauer, M., Dengler, D. and Paul, G. Instructible Information Agents for Web Mining. In Proceedings of the International Conference on Intelligent User Interfaces (January 9-12, pp. 21-28, New Orleans, USA, 2000).
2. Bergman, L., Lau, T., Castelli, V. and Oblinger, D.: Personal Wizards: collaborative end-user programming. *In Proceedings of the End User Development Workshop at CHI'2003 Conference* (Ft. Lauderdale, Florida, USA. April 5-10).
3. Castells, P. and Szekely, P. Presentation Models by Example. En: Duke, D.J., Puerta A. (eds.). Design, Specification and Verification of Interactive Systems. Springer-Verlag, pp. 100-116, 1999.
4. Castells, P. and Macías, J.A.: Context-Sensitive User Interface Support for Ontology-Based Web Applications. Poster Session of the 1st. International Semantic Web Conference (ISWC'02), Sardinia, Italia; June 9-12th, 2002.
5. Cypher A. (ed.).: Watch What I Do: Programming by Demonstration. The MIT Press, 1993.
6. Darragh, J. J. and Written, I.H.: Adaptive predictive text generation and the reactive keyboard. Interacting with Computers 3, no. 1:27-50, 1991.
7. Hurst, Matthew Francis: The Interpretation of Tables in Texts. *PhD. Thesis*. University of Edinburgh, 2000.
8. Klann, M.: End-User Development Roadmap. *In Proceedings of the End User Development Workshop at CHI'2003 Conference* (Ft. Lauderdale, Florida, USA. April 5-10).
9. Lieberman, H. (ed): Your Wish is my Command. Programming By Example. Morgan Kaufmann Publishers. Academic Press, USA. 2001.
10. Macías, J.A. and Castells, P. Dynamic Web Page Authoring by Example Using Ontology-Based Domain Knowledge. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI'03)* (Miami, Florida, USA. January 12-15).
11. Macias, J.A. and Castells, P. Using Domain Models for Data Characterization in PBE. *In Proceedings of the End User Development Workshop at CHI'2003 Conference* (Ft. Lauderdale, Florida, USA. April 5-10).
12. Macías, J.A.; Castells, P.: DESK-H: building meaningful histories in an editor of dynamic web pages. In Proceedings of the 11th Internacional Conference on Human-Computer Interaction (HCII). Creta, Grece, June 23-27, 2003.
13. Macías, J.A.: Authoring Dynamic Web Pages by Ontologies and Programming by Demonstration Techniques. PhD. Thesis. Departamento de Ingeniería Informática. Escuela Politécnica Superior. Universidad Autónoma de Madrid. September, 2003. http://www.ii.uam.es/~jamacias/tesis/thesis.html.
14. Miller, Rober C.: End User Programming for Web Users. *In Proceedings of the End User Development Workshop at CHI'2003 Conference* (Ft. Lauderdale, Florida, USA. April 5-10).
15. Mori, G., Paternò, F. and Santoro, C.: CTTE: Support for Developing and Analysing Task Models for Interactive System Design. IEEE Transactions in Sotware Engineering. IEEE Press. Vol. 28, No.8, pp. 797-813, August 2002.

16. Myers, B. A. Creating User Interfaces by Demonstration. Academic Press, San Diego, 1988.
17. Mo, D.H.; Witten, I.H.: Learning text editing tasks from examples: A Procedural approach. Behaviour & Information Technology, Vol. 11, No. 1, pp. 32-45, 1992.
18. Muslea, I. Extraction Patterns for Information Extraction Tasks: A Survey. In *Proceedings of AAAI Workshop on Machine Learning for Information Extraction* (Orlando, Florida, July, 1999).
19. Network of Excellence on End-User Development. http://giove.cnuce.cnr.it/EUD-NET.
20. Paganelli, L., Paternò, F.: Automatic Reconstruction of the Underlying Interaction Design of Web Applications. Proceedings of the SEKE Conference, pp. 439-445. ACM Press, Ischia, 2002.
21. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer Verlag, 2001.
22. Paynter, G.W.; Witten, I.H.: Automating Iteration with Programming by Demonstration: Learning the User's Task. Proccedings of the IJCAIWorkshop on Learning about Users, 16th International Joint Conference on Artificial Intelligence. Stockholm, Sweden, 1999.
23. Sahuguet, A.; Azavant, F.: building Intelligent Web Applications Using Lightweight Wrappers. Data and Knowledge Engineering, 2000.
24. Shneiderman, B.: Leonardo's Laptop. The MIT Press, 2003.