Dynamic Web Page Authoring by Example Using Ontology-Based Domain Knowledge

José A. Macías and Pablo Castells
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Ctra. de Colmenar Viejo km. 15
Madrid, 28049 Spain
+34-91-34822{41, 84}
{j.macias, pablo.castells}@uam.es

ABSTRACT

Authoring dynamic web pages is an inherently difficult task. We present DESK, an interactive authoring tool that allows the customization of dynamic page generation procedures with no a-priori tool-specific skill requirements from authors. Our approach consists of combining Programming By Example (PBE) techniques with an ontology-based representation of knowledge displayed in web pages. DESK acts as a client-side complement of a dynamic web page generation system, PEGASUS, which generates HTML pages from a formally structured domain model and an abstract presentation model. Authorized users can modify the internal presentation model by editing the generated HTML pages with DESK in a WYSIWYG environment. DESK keeps track of all user's actions and exploits the explicitly represented domain semantics to enhance the power of PBE techniques.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces And Presentation**]: User Interfaces – *Graphical user interfaces (GUI), Screen Design, Interaction styles*; H.5.4 [**Information Interfaces And Presentation**]: Hypertext / Hypermedia – *Navigation, User Issues.*

General Terms

Human Factors, Design, Algorithms, Languages.

Keywords

Programming By Example, Ontology, Model-based Paradigm, Knowledge-based UI Design

1. INTRODUCTION

An increasing proportion of web contents and functionality are accessed today through dynamically generated web pages, yet the development of dynamic pages remains a complex task that requires advanced programming skills. Languages and frameworks like XSL and JSP/ASP can greatly simplify the development and maintenance of dynamic web pages, but still require advanced technical knowledge that domain experts, graphic designers or even average programmers may 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 the code. It is difficult to provide WYSIWYG tools for the development of dynamic pages because it is hard to describe procedural behavior visually.

DESK [9] (Dynamic web documents by Example using Semantic Knowledge) takes up the challenge of supporting the customization of page generation procedures in an editing environment that looks like an HTML editor from the author point of view. DESK acts as a client-side complement of a dynamic web page generation system, PEGASUS [5], [11], [14], which generates HTML pages from a formally structured domain model and an abstract presentation model. The PEGASUS presentation model specifies which pieces of knowledge should be presented and how when a certain unit of information from the domain model is output to the user. Instead of using the PEGASUS modeling language, authorized users can modify the internal presentation model by editing in DESK the HTML pages generated by PEGASUS. DESK follows the Programming By Example (PBE) approach [6], [7], to infer changes that affect whole classes of knowledge from user's actions on the presentation of a specific unit.

DESK keeps track of all user's actions on edited documents, builds-up a model of the (HTML) document display structure (syntax), identifies domain model data fragments (semantics) in the document, and uses this information to give a semantic meaning to editing actions. DESK includes an inference agent used to recognize high-level transformations. The agent is constantly listening to the user's actions to infer complex modifications. The agent's behavior is configured using pre-activation hints that activate different heuristics for recognizing global widget changes.

A tool like DESK allows widening the spectrum of authors that can participate in otherwise abstract and complex model-based environments like PEGASUS. Inversely, our work shows that PBE techniques can benefit from a knowledge-based approach, which provides knowledgeable models of the user interface and explicit domain semantics for the PBE component to reason about.

IUI 03, January 12–15, 2003, Miami, Florida, USA.

2. PEGASUS

PEGASUS (Presentation modeling Environment for the Generation of ontology-Aware context-Sensitive web User interfaceS) is a system that helps create a dynamic front-end for ontology-driven knowledge-based applications on the web [5], [11], [14]. PEGASUS supports the definition of made-to-measure ontologies for the description of domain knowledge. The system generates web pages on the fly by selecting domain objects and assembling them into HTML documents in response to user's requests for knowledge units.

2.1 Domain Model

The domain model in PEGASUS is a semantic network of ontology class instances and relations. For example, assuming that a domain ontology has been defined in PEGASUS for a software download site like Tucows, including classes like Product, Category, HigherCategory, LowerCategory, and Catalog (for a description of class definitions in PEGASUS see [5]), instances like the following could be defined:

```
<HigherCategory id="Internet">
  <subCategories>
    <HigherCategory ref="Connectivity"/>
    <HigherCategory ref="Communications"/>
    <HigherCategory ref="E-Mail"/>
    ...
  </subCategories>
</HigherCategory>

<HigherCategory id="E-Mail">
  <subCategories>
    <LowerCategory ref="E-Mail Clients"/>
    <LowerCategory ref="E-Mail Parsers"/>
    ...
  </subCategories>
</HigherCategory>

<LowerCategory id="E-Mail Clients">
  <products>
    <Product ref="Agile Mail"/>
    <Product ref="Allegro Mail"/>
    ...
  <products>
</LowerCategory>

<Product id="Allegro Mail"
         license="Shareware" price="39.95">
  <information> <AtomicFragment>
      With AllegroMail, you can set up...
  </AtomicFragment> </information>
</Product>
```

XML attributes like `license` and `price` correspond to properties of a knowledge unit (of class `Product`), whereas elements like `subCategories` and `products` are relations with other units (the `ref` attribute corresponds to the unit ID's). While we are currently using our own XML extensions to represent the domain model for historical reasons, we are planning to move to some of the currently available ontology definition standards like RDF or OWL [8], with minor modifications to our system.

2.2 Presentation Model

In contrast with other knowledge-based systems that do automatic page generation (e.g. Adaptive Hypermedia systems [3], [17], [21], [22]), PEGASUS provides extensive control over presentation design, by using an explicit presentation model, separate from contents. The separation of content and presentation is achieved by defining a *presentation template* for each class of the ontology. Templates define what parts (attributes and relations) of a knowledge item must be included in its presentation and in what order, their visual appearance and layout.

Templates are defined by using an extension of HTML based on JavaServer Pages (JSP), that allows inserting control statements (between `<%` and `%>`) and Java expressions (between `<%=` and `%>`) in the HTML code. For instance, a template for class `HigherCategory` could be as follows:

```
<% if (availableSpace > 5) { %>                    1
   <widget type="Table" columns="3"                2
           dataflow="wrap">                        3
     <list> <%= subcategories %> </list>           4
   </widget>                                       5
<% } else { %>                                     6
   <table>                                         7
     <tr><td> <%= id %> </td></tr>                 8
     <tr><td> <%= subcategories %> </td></tr>      9
   </table>                                        10
<% } %>                                            11
```

This template indicates that when there is enough available space (which is estimated on a scale from 0 to 10) a table is created in which a subcategory is presented in each cell, left to right and top to bottom (lines 2 to 5). Otherwise a table of two rows and a single column is generated (lines 7 to 10) where the category `id` (line 8) and the list of subcategories (line 9) are displayed. The expression `<%= subcategories %>` is a reference to the multivalued relation `subcategories` of the `HigherCategory` being displayed. The relation points to a list of objects of type Category, which PEGASUS presents using the appropriate template recursively. The `widget` XML tag is a JSP custom tag used to provide a standard set of HTML widgets like tables, input types (buttons, combo boxes, etc.), and lists. Each widget type has specific mechanisms to display domain model data structures, using different strategies to map complex relations between domain objects to display structures.

The resulting page for the Internet category can be seen in Figure 1, where the outer table results from lines 2 to 5 of the template, and the inner tables correspond to lines 7 to 10 applied to subcategories of Internet software (a few details like cell background colors and the tabbed bar have been omitted in the template code for the sake of brevity).

Besides templates, the PEGASUS presentation model also includes presentation rules like the following:

```
<Rule>
  <test condition="availableSpace <= 1 "/>
  <presentation> <%= this.asLink() %>
  </presentation>
</Rule>
```

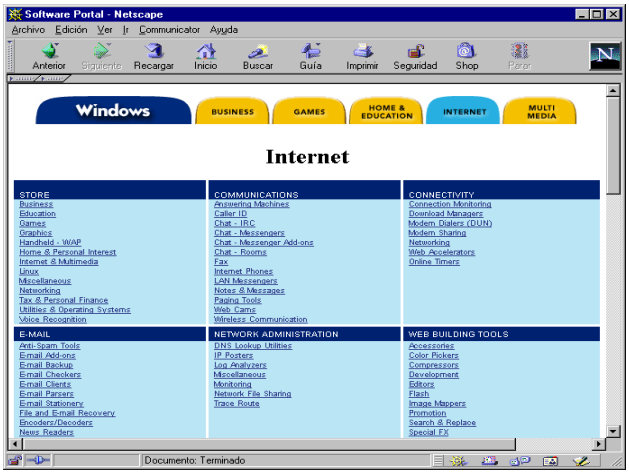

Figure 1. Generated web page for an instance of type `HigherCategory`

In Figure 1, this rule is responsible for presenting third-level subcategories, such as "E-mail Clients", as a link.

Adaptivity is achieved in the presentation model by putting conditions on templates, on parts inside templates, in presentation rules, and over relations between domain objects. These conditions can test properties of the user model (overlay model and user profile), properties of the data, characteristics of the platform, and any other aspect that should influence presentation, like task requirements, user's goals, usage modes (e.g. exploration vs. selective search), etc.

At runtime, the user interacts with the application from a web browser. The interaction with an application built with PEGASUS consists of traversing the semantic network of domain objects. Each time the user moves to an object, PEGASUS responds by generating an HTML page (see Figure 2). In doing so the system 1) resolves the user's request by determining the actual object to move to, 2) locates the instance in the domain model, 3) updates the domain and user models, 4) generates the HTML presentation applying the pertinent rules and the template that corresponds to the object class. In the generated pages links do not point to other pages but refer, explicitly or descriptively, to other domain objects.

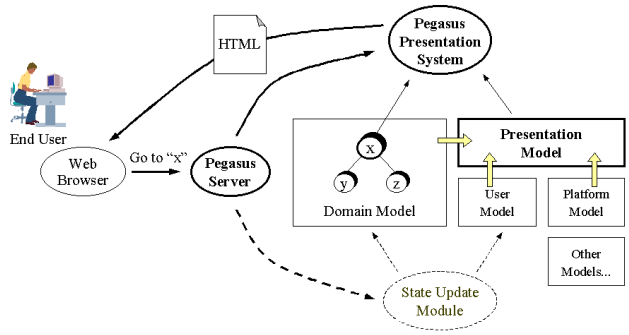

Figure 2. PEGASUS Architecture

From the PEGASUS point of view, the unit of interaction with the user are HTTP requests. User model updates are carried out by taking only into account the information extracted from client requests. Platform and user interface characteristics are captured in the client by means of JavaScript code that the system inserts in the generated HTML pages, and the information is returned to the server as part of the HTTP request when the user clicks on links and buttons. This assumption greatly simplifies the system architecture and the integration with external tools and modules. In exchange, it means that the system is not explicitly aware of user activity between two requests, and presentation is not updated in this interval either. A finer but far more complex and bandwidth-sensitive approach could be supported by generating Java user interface components (applets) that interact with the user, and communicate directly with the server to query and update the domain and user models.

3. OVERVIEW OF DESK

With DESK the user can modify the design of dynamic web documents by editing the HTML code generated by PEGASUS, instead of using the PEGASUS modeling language to edit the abstract internal models. DESK identifies domain values, model fragments, and presentation constructs in the HTML code, from which it infers meaningful transformations on the models. The user only knows about the HTML document and needs not be aware of the underlying models.

DESK has a client-side and a server-side. The client-side looks like a conventional HTML edition tool, where the designer edits web pages. The editor monitors the user and generates a *monitoring model* that is sent to the DESK server-side. The server-side processes the monitoring model, infers changes to the PEGASUS models, and generates a report that is sent to the user for feedback.

3.1 Monitoring the User

Our authoring tool uses a set of suitable heuristics that consist of advanced searching and locating algorithms for obtaining both syntactic and semantic information from user's actions in the edition of a document. Such heuristics also use available knowledge coming from the semantic network and the domain ontology supplied by PEGASUS for mapping changes in HTML edition to such domain structures. Therefore, DESK records all basic editing actions (insert text, change text style, etc.) for each user action, and attempts to find out the syntactic context by applying *low level heuristics* (H_L), packing the context information and user's actions into *constructors primitives* to make up the monitoring model (Figure 3).

Low level heuristics determine the syntactic context of user actions, which will be used by the server-side. Low level heuristic are grouped into the following modules:

- The context location module finds out the nearest context of text insertion and deletion and maps meaningful blocks of HTML code to domain objects.
- The special structure location module identifies presentation structures (i.e. tables, selection lists, etc.) in which the changes occur. This module knows about rows and columns and how data structures are mapped to different types of widget.

- The constructor primitive module generates a structured monitoring model of user actions and surrounding context, composing atomic interaction events into higher-level editing primitives.

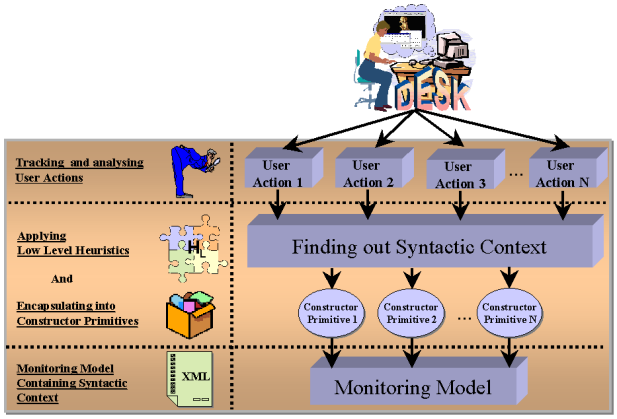

Figure 3. DESK client-side

3.2 Inferring Transformations

Figure 4 shows the back-end architecture of DESK. The client-side sends the monitoring model to the DESK server-side, where inference takes place.

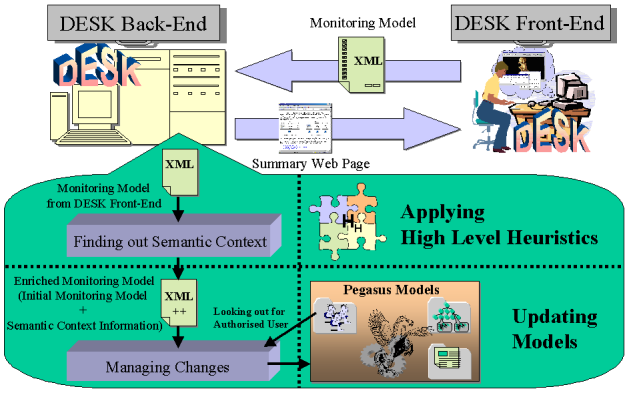

Figure 4. DESK server-side

The server-side builds a semantic context for applying changes to the PEGASUS models using *high level heuristics* (H_H). High level heuristics find the semantic context in terms of the domain model. High level heuristics are grouped into the following modules:

- The context location module finds out the semantic context in terms of ontology definitions and domain objects, and adds it to the monitoring model.
- The presentation context module generates domain object access expressions and locates in the PEGASUS presentation model the domain objects that were identified by low-level heuristics, in order to determine where exactly the changes will be made.
- The disambiguation module deals with changes that involve multiple contexts. The system attempts to solve ambiguities using contextual information. When it fails, it prompts the user for help.

After the monitoring model is enriched with semantic context, the change management module applies the changes to PEGASUS models only if the user is authorized, sending a summary report to the user.

4. AN EXAMPLE

The DESK client looks like a standard HTML editor, but it maintains a structured model of the user interaction internally. Figure 5 shows the example from Figure 1 being edited in DESK, where a) the text "Applications" has been inserted after the "Internet" literal, and b) a few items from the table have been cut and pasted onto a combo box and a selection list. Items can added or deleted from both widgets in a pop-up window that is opened by double-clicking on the widget.

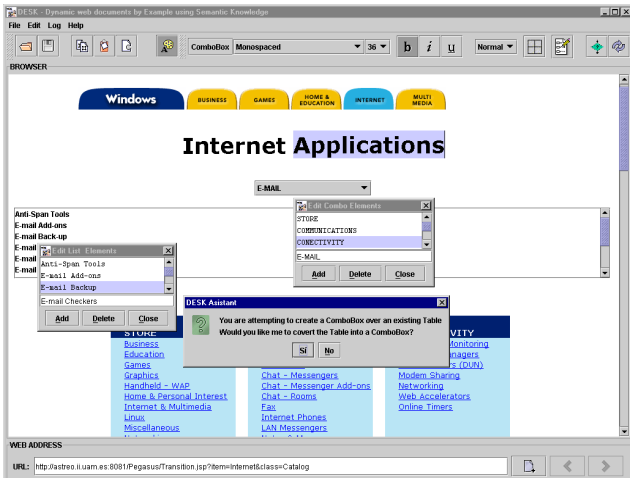

Figure 5. The DESK authoring tool

The user's purpose is to substitute the table display of product categories by a combo box, where one of the higher categories can be selected, and a list widget showing the subcategories of the selection. DESK detects the user's intent and suggests him/her to automatically replace the whole table by the two widgets. Figure 6 depicts the resulting page after DESK has changed the presentation model on the server-side. As we can see, the change to the `title` attribute affects the presentation template twice since the `title` attribute is rendered in both the tabbed catalog panel and the previously mentioned literal. This finally results in replacing the literal "Internet" by "Internet Applications" at both mentioned places, furthermore a combo box and a selection list have been added, replacing the previous table in the document.

4.1 Client-Side Processing

The monitoring model is being continually updated to reflect the actions that the user is doing. DESK has a set of heuristics for user's intent recognition in typical design transformations. For instance DESK knows about substituting certain presentation structures for others. Rather than checking whether the heuristics

can be applied at each step, which would be very inefficient, DESK has a pre-activation agent that checks for lighter conditions. Only one agent is needed in order to continuously check the monitoring model to detect certain types of actions (i.e. copy actions between widgets), the agent being activated when detecting such actions. The agent looks for partial clues that hint the system towards specific heuristics, which involve a more detailed analysis of actions and objects involved. The agent behavior is configured by a set of transformation hints like the following.

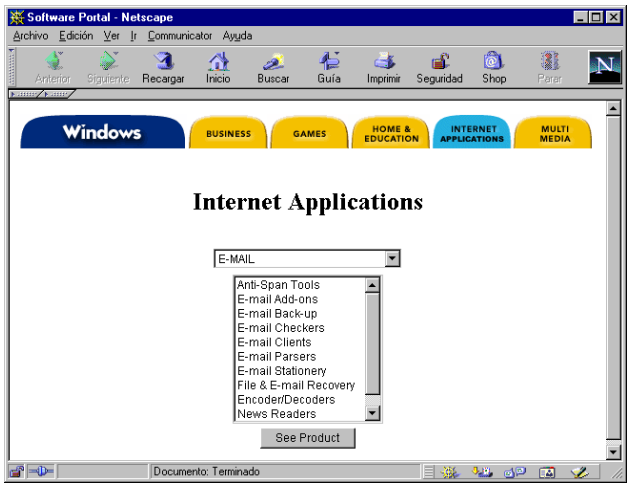

Figure 6. Resulting web page after the presentation model is changed

```
<TransformationHint searchLength="100">
  <widget type="Table"
          changeTo="ComboBox,List" />
  <Condition action="Creation"
             object="ComboBox" />
  <Condition action="Creation"
             object="List" />
  <Condition action="PasteFragment"
             from="Table" to="ComboBox"
             repeat="3" />
  <Condition action="PasteFragment"
             from="Table" to="List"
             repeat="3" />
  <Condition fact="Relation" from="ComboBox"
             to="List" />
</TransformationHint>
```

This hint activates a heuristic to transform a table into a combo box and a selection list when the following conditions hold: a combo box has been created, a list has been created, domain fragments have been pasted from a table into the created widgets at least three times, and there is some relation between data in both widgets. The `searchLength` attribute represents how many actions back in the monitoring history the agent will look at a time. Once activated, the transformation heuristic will carry out more elaborated tests to verify whether and how the transformation should actually be applied, which involves recognizing iteration patterns and coordinated data flow over complex display structures.

The monitoring model consists of a sequence of constructor primitives that reflect the actions performed by the user, in the context of semantic information inferred by low level DESK heuristics. The following monitoring model fragment shows two editing primitives: the insertion of the `Applications` literal by the designer and the transformation of a table into a combo box and a selection list.

```
<InsertText>
  <Text> Applications </Text>
  <Context start="09" end="21"
           before="T01" after="TB01">
    <Text> Internet </Text>
  </ Context>
</InsertText>
<ChangeWidget>
  <From type="Table" id="T01"
        relation="subCategories"
        class="HigherCategory"
        objectID="Internet"/>
  <To   type="ComboBox" id="C01"
        relation="subCategories"/>
        class="HigherCategory/>
  <To   type="List" id="L01"
        relation="subCategories"
        class ="LowerCategory" />
</ChangeWidget>
```

In the case of text insertion one can see how DESK has found context information about the change, that is, where the information is located: starting at the ninth character position following the literal “Internet”, and ending at the twenty-first position. Context semantics reflect the fact that the insertion has been done between the tabbed bar `TB01` and the table `T01` (all widgets are internally assigned an ID by DESK). The widget transformation heuristic has generated a high-level action involving domain semantics (classes, relations and domain objects), and relationships between widgets.

4.2 Server-Side Processing

The monitoring model constructed by the client-side is sent to the server to apply the changes to PEGASUS models. The server-side analyzes the monitoring model, starting from the first action (text insertion) and using high-level heuristics to locate information in the domain model to add the following semantics:

```
<Context class="Category" attribute ="id"
         objectID="Internet"/>
```

In this case the DESK server-side decides that the change should affect the domain model itself, rather than the presentation template. DESK finds the object with ID “Internet” (instance of `Category`) in the domain model, and changes the `id` attribute to “Internet Applications”.

From the second entry (widget change) in the monitoring model the system decides to modify the presentation model and no contextual information is added. The table widget in the presentation template for `HigherCategory` is substituted by a combo box and a selection list widget. Appropriate domain fragment access expressions are generated, and the presentation template results as follows.

```
<% if (availableSpace > 5) { %>
```

```
   <widget type = "ComboBox">
     <items> <%= subCategories %> </items>
     <selectedItem> <%= selectedID %>
     </selectedItem>
   </widget>
   <widget type = "List">
     <items>
       <%= subCategories.item(SelectedID)
           .subCategories %>
     </items>
   </widget>
<% } else { %>
   <table>
     <tr> <td> <%= id %>  </td> </tr>
     <tr> <td> <%= subcategories %>
               </td> </tr>
   </table>
<% } %>
```

The variable `SelectedID` above is an input parameter used for selection widgets like the combo box. These parameters are internally generated and controlled by the system, depending on the number of input values needed for the widgets.

5. RELATED WORK

Monitoring user actions is a common practice to provide so-called intelligent interaction between users and web applications. One example of this approach is AVANTI [19], in which the system tracks the user interaction in a web browser designed for universal accessibility, as a front-end of the AVANTI information system. This tool uses Unified User Interfaces (U^2Is) from Unified User Interface Development methodology (U^2ID) to achieve self-adaptability, monitoring any kind of user during the navigation, including eldery and disabled people. To this respect, DESK monitors user's actions by using configurable and parametrical information from different knowledge models. An intelligent agent tracks user's steps during the interaction to infer atomic changes, this way our authoring tool supplies the user with an easy-to-use web browser and WYSIWYG edition tool to automate changes to web presentations.

Turquoise [16] is also an intelligent browser and editor for the web that allows users to create dynamic pages by example rather than by writing program code. With such authoring tool, users without programming experience can create scripts that combine data from several web pages, automate repetitive browsing or editing tasks, convert other data formats into HTML, and process submitted forms. Scripts are demonstrated by familiar browsing and editing actions, which Turquoise records and generalizes into a program. Like DESK, Turquoise is based on the Programming By Demonstration paradigm [4], [6], [7], where the system infers procedural information from examples of what the user wants to achieve. Turquoise operates by inferring scripts from user's actions by copying HTML contents to a special window. Similarly, in Scrapbook [20] the users can demonstrate which portions of web pages they are interested in by creating a personal page, that is, selecting data on a web browser and copying it to the single personal page. Web data is copied directly from Netscape Navigator using APIs of those browsers. Once the personal page is created, the system automatically updates it by extracting the user-specified portions from the latest web pages. Thus, the user can browse only the necessary information on a single page and avoid repetitive access to multiple web pages. In contrast, DESK allows the user to freely edit the HTML document, inferring a great deal of changes to domain and presentation models.

In the TriIAs system [1] the user interacts with a web-based application like a travel planning agent that is intended to create a schedule for a trip satisfying all of the constraints entered by the User. TriIAs calls an information extracion trainer that is able to learn new wrappers for extracting relevant pieces of information from web documents. Wrappers [10], [18], provide a uniform access to the information stores in heterogeneous repositories like data bases, files and so forth. The extraction of structured information, such as context semantics used and generated by DESK from a semi-structured document (HTML and JSP code), is very similar to the way wrappers operate. In DESK, an agent tracks the user during the interaction for relieving him/her from having to complete repetitive tasks (e.g. copying items from a widget to another) being completed by the authoring tool. In that sense, our agent uses semantic enriched information present in the monitoring model, as well as semantics coming from our ontology and domain-knowledge, making easy the inference step, without the necessity of using extraction languages.

WebSheets [23] is also a WYSIWYG authoring tool based on the Programming By Demonstration paradigm for building dynamic web pages that access and modify databases. WebSheets uses QBE (Query By Example) by which queries are generated in the authoring environment from user actions, using an SQL-like language to access databases for requesting information. In DESK the domain information is stored using an ontology-based semantic model rather than a relational model. To this end, our authoring tool can afford complex changes made by the user, relating domain objects and widget structures, so that the variety of changes can be quite heterogeneous, and all of them can be applied to both domain and presentation model.

On the other hand, in [2] a model-based approach for web engineering is proposed. This work consists of an architecture for the reverse engineering of web pages, with a view to apply forward engineering subsequently. Reengineering methods are then applied to produce new user interfaces for multiple contexts of use, thus creating a capability to rapidly produce user interfaces for different computer platforms, access devices, etc. To this respect, DESK provides a reengineering mechanism by following the inverse of the path followed by PEGASUS, starting from the HTML code back to the constructor models [12].

6. CONCLUSIONS

The development and maintenance of dynamic interfaces for web applications is a costly task. Our work aims at combining the ease of use of an interactive authoring tool with the power of the model-based and knowledge-based approach. DESK provides the designer with an intuitive authoring environment capable of editing complex web page designs. Our authoring tool is based on the Programming by Example paradigm, where the user supplies the system with an example of what s/he wants to get and the system infers the changes to dynamic page generation procedures.

By tracking the user as s/he edits pages, DESK obtains enriched information from user's actions, comparing such elaborated information with semantic domain knowledge to provide the user

with continuos assistance during the edition process. Changes are automatically carried out in the server by using domain knowledge from PEGASUS, using presentation templates for making generic changes to the way complex data structures are visualized. DESK tries to infer maximal information from user's actions and from existing semantic knowledge. All this semantic information is parameterized, editable and independent from the application domain.

In a sense DESK follows the inverse path of a dynamic page generation system, here PEGASUS. This path has a higher level of ambiguity because for a change to the document by the user several interpretations are possible and different generalizations can be considered. DESK solves this ambiguity by using heuristics like extracting structured knowledge from the domain model of PEGASUS. Sometimes this is not enough and the system can make mistakes. The purpose of this work is not to build a system that never fails, but to make a useful authoring tool capable of inferring correct actions in a reasonable number of cases. DESK can successfully infer the following types of changes in documents:

- Insertion, deletion and modification of HTML fragments along the presentation template and generation rules.
- Insertion, deletion and modification of HTML tags surrounding domain elements along the presentation template and generation rules.
- Creation, deletion, modification and automatic transformation between widgets (i.e. combo-boxes, selection lists, and tables).
- Insertion, deletion and modification of both text and multimedia references in the domain model.

A working prototype of DESK has been developed where most of the features described in this paper have been implemented. We have considered several scenarios dealing with domains like academic institution information, e-commerce catalogs, travel information, and educational materials. We have worked with ontologies taken from standard repositories like the ontology libraries from the DAML project. We are currently improving our tool to deal with more complex edition cases such as advanced widget manipulation, and dealing with new models like user and platform.

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

[1] Bauer, M., Dengler, D., Paul, G. Instructible Information Agents for Web Mining. Proceedings of the International Conference on Intelligent User Interfaces (IUI'2000). 2000, January 9-12, New Orleans, USA, 21-28.

[2] Bouillon, L., Vanderdonckt, J. Eisenstein, J. Model-Based Approaches to Reegineering Web Pages. Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2002, 18-19 July 2002, Bucharest, Romania, 86-95, 2002.

[3] Brusilovsky, P., Eklund, J., Schwarz, E. Web-based Education for all: a Tool for the Development of Adaptive Courseware. Computer Networks and ISDN Systems, 30, 1-7, 1998.

[4] Castells, P., Szekely, P. Presentation Models by Example. En: Duke, D.J., Puerta A. (eds.): Design, Specification and Verification of Interactive Systems '99. Springer-Verlag, 1999, pp. 100-116.

[5] Castells, P. Macías, J.A. An Adaptive Hypermedia Presentation Modeling System for Custom Knowledge Representations. Proceedings of the World Conference on the WWW and Internet (WebNet'2001). Orlando (Florida), October 2001.

[6] Communications of the ACM. The Intuitive Beauty of Computer Human Interaction. Special issue on Programming by Demonstration, 43, 3, March 2000.

[7] Cypher A. (ed.). Watch What I Do: Programming by Demonstration. The MIT Press, 1993.

[8] Dean, M., D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language 1.0 Reference W3C Working Draft 29 July 2002. Available at http://www.w3.org/TR/owl-ref.

[9] Fulantelli, G., Corrao R., Munna, G. Enhacing User Interaction on the Web. Proceedings of the WebNet'99 (Honolulu, Hawaii, USA October 1999), AACE, 403-408.

[10] Huang, Anita W. Aurora A Conceptual Model for Web-Content Adaptation to Support the Universal Usability of Web-based Services. Conference on Universal Usability (CUU 2000). Arlington VA, USA, 2000.

[11] Macías, J. A., Castells, P. A Generic Presentation Modeling System for Adaptive Web-based Instructional Applications. Proc. Conference on Human Factors in Computing Systems (CHI'2001), Extended Abstracts. Seattle, Washington, 2001.

[12] Macías, J.A., Castells, P. Tailoring Dynamic Ontology-Driven Web Documents by Demonstration. Proceedings of the 6th International Conference on Information Visualisation (IV 2002). London (England), 2002.

[13] Macías, J.A., Castells, P. Interactive Design of Adaptive Courses. In Computers and Education – Towards an Interconnected Society, M. Ortega and J. Bravo (eds.). Kluwer Academic Publishers, Dordrecht (The Netherlands), 2001.

[14] Macías, J. A., Castells, P. Adaptive Hypermedia Presentation Modeling for Domain Ontologies. To appear in proceedings of 10th International Conference on Human-Computer Interaction (HCII '2001). New Orleans, Louisiana, 2001.

[15] Macías, J. A., Castells, P. An Authoring Tool for Building Adaptive Learning Guidance Systems on the Web. Active Media Technology. J. Liu et al. (Eds.), pp.. Lecture Notes in Computer Science, LNCS 2252. Springer-Verlag Berlin Heidelberg 2001. 268-278. ISBN 3-540-43035-0.

[16] Miller, R., Myers B. Creating Dynamic World Wide Web Pages By Demonstration. Carnegie Mellon University School of Computer Science, CMU-CS-97-131 and CMU-HCII-97-101, 1997.

[17] Murray, T. Authoring Kowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. Journal of the Learning Sciences 7, 1, 1998, 5-64.

[18] Muslea, I. Extraction Patterns for Information Extraction Tasks: A Survey. Proceedings of AAAI Workshop on Machine Learning for Information Extraction. Orlando, Florida, July 1999.

[19] Paramythis, A., Savidis, A., Stephanidis C. AVANTI: a universally accesible web browser. Proceedings of the HCI'2001 (New Orleans, USA, August 2001), Lawrence Erlbaum Associates, Publishers, 91-95.

[20] Sugiura, A., and Koseki, Y. Internet Scrapbook: automating Web browsing tasks by programming-by-demonstration. Proceedings of the 7th International WWW Conference (Brisbane, Australia, April 1998). Elsevier Science.

[21] Vassileva, J.: Dynamic Course Generation on the WWW. Proceedings of the 8th World Conference on Artificial Intelligence in Education (AIED'97). Kobe, Japan, 1997, 498-505.

[22] Weber, G., Specht, M.: User Modeling and Adaptive Navigation Support in WWW-based Tutoring Systems. Proceedings 6th International Conference on User Modeling (UM97). Sardinia, Italy, 1997.

[23] Wolber, D., Su, Y., Chiang Yih. Designing Dynamic Web Pages and Persistence in the WYSIWYG Interface. Proceedings of the International Conference on Intelligent User Interfaces (IUI'2002). San Francisco, California, USA. January 13-16 2002, pp. 228-229.