

Tailoring Dynamic Ontology-Driven Web Documents by Demonstration

José A. Macías and Pablo Castells
Universidad Autónoma de Madrid (Spain)
{j.macias, pablo.castells}@uam.es

Abstract

In this paper we present DESK, an authoring tool for the automatic customisation of the front-end of web applications as a result of changes that users perform in dynamically generated HTML pages. Our authoring tool uses domain knowledge and presentation knowledge stored in PEGASUS, an automatic web page generation system for rendering ontology-driven knowledge. DESK automatically detects the differences between original and modified web pages and uses heuristics to infer additional knowledge for modelling the context of each change. DESK uses an explicit user model to identify which user can perform each kind of change to the web presentation.

1. Introduction

The dynamic generation of web pages has become commonplace for even the simplest WWW applications today. Since the mid-90's a significant progress has been achieved in the development of systems and tools that simplify the automatic generation of pages, in exchange for affordable limitations on the expressive power provided to developers [1, 13, 14]. With these tools the developer inputs structured knowledge and multimedia resources in a very simple format, and the system takes care of the generation of HTML code on the fly at runtime. In general, such tools generate the code according to a fixed page design that the developer cannot modify.

On the other hand, maintaining knowledge-based web applications is not an easy task for unskilled users due to the complexity and abstraction needed for knowledge modeling inside of this kind of systems. Most of these applications take care automatically of the final web page creation, but it is up to the

developers to maintain and process changes that affect the structure and appearance of the application. For this reason, easy-to-use authoring tools are needed to provide developers with an easy paradigm to change contents and final presentation, as well as to provide the user with a specific language for modeling the final design [3, 4].

In this paper we present DESK (Dynamic web documents by Example using Semantic Knowledge), an authoring tool for modifying information presentation and/or contents in knowledge-based web applications. Our authoring tool aims at providing end-users with an easy and flexible paradigm to make changes to knowledge visualisation by using techniques of Programming By Demonstration [2, 5, 6]. This means that the user only has to perform changes to a web page and supply DESK with the original and the modified web pages. Afterwards our authoring tool detects the changes made by the user, locates their context, and finally carries out these changes. Changes may affect contents (domain model) and/or presentation (presentation model) of web applications.

DESK is based on PEGASUS [3, 4, 10], a system for the dynamic generation of web documents in information systems such as digital libraries, touristic guides, educational applications, on-line museums, and so forth. PEGASUS supports the generation of web presentations, and integrates the management of contents with dynamic aspects like adaptation to users and dependencies between knowledge properties and structure and an explicit model of web page design.

2. Related work

The development of WYSIWYG authoring tools for dynamic web page generation is an inherently difficult problem. Some authoring tools for the development of adaptive hypermedia systems [1]

include interactive editors for introducing contents as well as defining complex knowledge structures [9], but not to design presentations and relations between different models such as domain model, user model, platform model and so on.

We have created DESK to fill this gap, using the Programming By Demonstration paradigm [2, 5, 6] to allow the page modification by a non-expert user. In programming by demonstration the system infers procedural information from examples of what the user wants to achieve. The programming by demonstration paradigm has an intrinsic ambiguity because general information has to be derived from particular cases provided by the user. To solve such ambiguity some strategies have been used, such as monitoring all user interaction (vs. watching only the initial and the final state), using multiple examples (e.g. negative examples), or interactively asking the user to help or decide.

DESK operates taking only one example of the initial and final states. This means that the user does not have to use a specific web editor that monitors all his/her actions. Given that DESK gets a limited amount of information from the user to infer from, the tool uses the domain model to locate the context of the changes carried out by the user.

The extraction of structured information, like the difference and context model used by DESK, from a semi-structured document (HTML code) is very similar to the way wrappers operate [8, 12]. Wrappers provide a uniform access to the information stored in heterogeneous repositories like data bases, files and so forth.

3. Knowledge representation in PEGASUS

PEGASUS makes minimum assumptions about how instructional knowledge is represented [9, 10]. It is our purpose to provide semantic web designers with a simple specification paradigm for non-trivial adaptive presentation constructs, which can be used with different web management systems. In order to allow for different approaches, PEGASUS supports the definition of made-to-measure domain ontologies for the description and conceptual structuring of subject matter (as in [11]). Once the ontology is defined, designers build web presentations by creating domain objects and relating them together using the conceptual vocabulary defined by the ontology. A web presentation is designed by defining an explicit presentation model where presentations are associated to ontology object classes and relations.

3.1 Domain model

Once the ontology has been defined, the domain model is constructed by creating semantic networks of domain objects, using the classes and relations defined in the ontology. For example, assuming classes like *Talk* and *Speaker* have been defined for storing information about conferences and events, the following example shows how an instance of a Talk might be defined (the ontology for our example is inspired on an ontology published at the DAML web site [7]).

```
<Talk ID="DESK"
  title="DESK Autoring Tool"
  uri="desk.html" date="20/03/2002"
  duration="20 min.">
  <Abstract>
    <AtomicFragment> DESK is an
      Authoring tool for making
      modifications to web pages...
    </AtomicFragment>
  </Abstract>
  <Speakers>
    <Speaker ID="JAMI"
      name="Jose Antonio Macías"
      organization="UAM"
      home_page=
        "http://www.ii.uam.es/~jamacias"
      email="j.macias@uam.es">
      <Picture>
        <AtomicFragment url="jami.jpg" />
      </Picture>
      <Biography>
        <AtomicFragment
          url="jami.html" />
        </Biography>
      </Speaker>
    <Speaker ID="PCA"
      name="Pablo Castells "
      organization="UAM"
      home_page=
        "http://www.ii.uam.es/~castells"
      email=
        "pablo.castells@uam.es">
      <Picture>
        <AtomicFragment url="pca.jpg" />
      </Picture>
      <Biography>
        <AtomicFragment url="pca.html" />
      </Biography>
      </Speaker>
    </Speakers>
  <Bibliography>
    <BibItem ref="Macias2001a" />
    <BibItem ref="Macias2001b" />
    <BibItem ref="Castells2001" />
    <BibItem ref="Castells1999" />
  </Bibliography>
  .....
</Talk>
```

XML attributes like `title` and `duration` belong to properties of the current knowledge unit, whereas `Picture`, `Speaker` and `Abstract`, for instance, are relationships between knowledge units.

3.2 Presentation model

In PEGASUS, 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. In these templates, the designer can use all the presentation constructs of the HTML language (lists, tables, frames, links, forms, etc.), and insert, using very simple Java expressions, the domain items to be presented. For instance, a fragment of a very simple template for class `Talk` could be as follows:

```
<table><tr><td><h2><%=date%></h2></td>
<td><h1><%=title%></h1> </td></tr>
<tr><td><h2><%=duration%></h2></td>
<td><h2><%= Abstract %></h2></td></tr>
<tr> <td> <h2>
<%= this.getTalksRelated() %>
.....
<%= Bibliography %></h2></td>
<td><%=uri%><center>Speakers</center>
<%= Speakers %>
</td></tr></table>
```

The designer uses this template by referencing attributes and relations as shown in boldface. Then PEGASUS accesses these values and represents the internal information by extracting it from the domain model. Figure 1 shows a web page generated using this template.

The goal of DESK is to enable the designer to change the presentation procedure by editing an HTML page like the one shown in Figure 1, instead of editing an abstract specification like the presentation template shown above.

3. DESK as an authoring tool

DESK enables the user to modify both domain knowledge and page design by editing HTML code from the pages generated by PEGASUS, without using the specific modelling language of PEGASUS.

DESK follows the inverse of the path followed by PEGASUS, starting from the HTML code to end analysing the constructor models. DESK achieves this changes by using different types of heuristics and programming by demonstration techniques for reasoning on both domain and presentation model, and deducing changes from contents or page design depending on the differences shown between the modified and the original document.

PEGASUS can be seen as implementing a transformation over a domain model D and a presentation model P , that results in the creation of a web page. For a knowledge unit $x \in D$, PEGASUS generates a web document D_{HTML} . If we represent such transformation as a function $f(D, P, x) \rightarrow D_{HTML}$, DESK represents the inverse function $f^{-1}_{D, P, x}(D_{HTML}, D'_{HTML}) \rightarrow D', P'$, where D'_{HTML} represents the document modified by the user, and D' and P' are the domain and presentation models after being modified by DESK.



Figure 1. Web page generated for a knowledge unit of class `Talk`

In general terms, the inverse path that DESK follows has a higher level of ambiguity than the direct path generated by PEGASUS, because for a change to the document by the user several interpretations are possible and different generalisations 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.

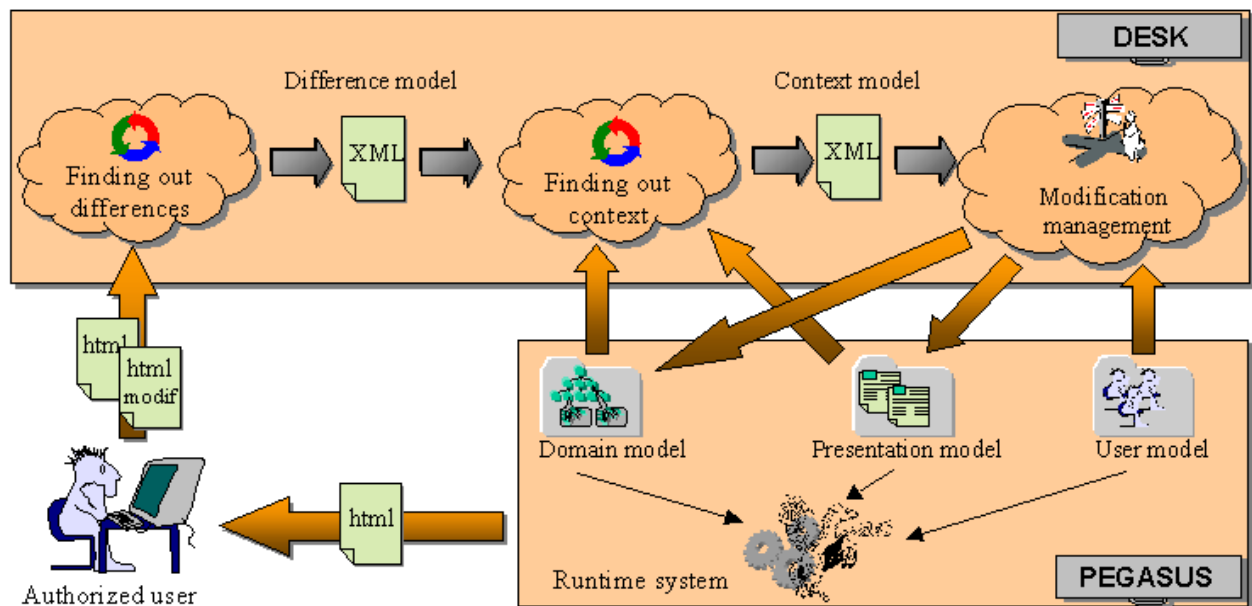


Figure 2. DESK Architecture

DESK can successfully infer the following types of changes in documents:

- Insertion, deletion and modification of HTML fragments along the presentation template.
- Insertion, deletion and modification of HTML tags surrounding domain elements along the presentation template.
- Deletion of both text and multimedia references in the domain model.

DESK identifies what kinds of changes have to be accomplished, taking into account the profile of the user. Currently DESK cannot deal with other changes than the ones listed above. In particular, it cannot modify the domain structure (relationships between knowledge units), or include new unit parts inside of a presentation's template.

Figure 2 shows the architecture of DESK. To begin with, the designer or any authorised user modifies the HTML generated by PEGASUS and sends to DESK the HTML page modified as well as the original one. Once DESK has received both pages, the first module of DESK tries to find out all known differences between both the original and modified page, reporting a structured difference model. The next module tries to find out the context of the previously modelled differences, using both the domain and the presentation model to build a structured context model of each difference, capturing details like the location of each modification, class and attributes involved, domain objects in the presentation

template. Finally a third module tries to manage and carry out each change, using the previously generated context model. This module also modifies the information contained in the domain and presentation model, updating the presentation correctly.

The change management module uses the user profile information to detect if a user could change or not the presentation depending on her/his role in the system. The next time the user navigates over the presentation, the changes will show according to the modifications made previously. In the next section we present an example to illustrate all the steps we have just described.

4. Explicit DESK models

As described before, DESK manipulates different types of models, coded in XML, where the information generated in each step is modelled and stored. In addition to the domain model used by PEGASUS, DESK builds and manages both a difference model and a context model.

4.1 Difference model

In the most basic sense, the difference model encodes structured information about the syntactic differences found during the comparison of the files sent to DESK by the user.

```

<DifferenceModel>
  <Diff id="1">
    <Original line="38" action="change">
      <h1> DESK Authoring Tool </h1>
    </Original>
    <Modified line="38" action="change">
      <h2> <u> DESK Authoring Tool
      </u> </h2>
    </Modified>
  </Diff>
  <Diff id="2">
    <Original line="38" action="change">
      <h2> 20/03/2002 </h2>
    </Original>
    <Modified line="38" action="change">
      <h2> 21/03/2002 </h2>
    </Modified>
  </Diff>
  <Diff id="3">
    <Modified line="60" action="adition">
      <blockquote>
        <h2> Related Bibliography </h2>
      </blockquote>
    </Modified>
  </Diff>
  .....
</DifferenceModel>

```

This code shows a fragment of the XML specification used to represent the difference model. For each difference, DESK generates a `Diff` element which codes the original and modified states of the text. Each element has an attribute called `action`, which indicates what kind of change is being done (addition, deletion, and change). In this example we can see how the user has changed the appearance of the literal `DESK Authoring Tool` (`<Diff id="1">`), adding a `` HTML-tag and reducing the text size to `<H2>` HTML-tag. The second modification (`<Diff id="2">`) shows us how the user has changed the date of the speech, from 20/03/2002 to 21/03/2002, and finally the third modification (`<Diff id="3">`) reflects how the user has added a new line of HTML code.

4.2 Context model

The context model provides semantic information about the differences found in the difference model. This way it is possible to locate exactly where the changes have to be applied in the domain and presentation models.

```

<ContextModel>
  <DiffContext id="1">
    <Context class_name="Talk" ID="DESK"
      attribute_name="title" />
    <Tags text_found=

```

```

      "DESK Authoring Tool">
        <h1/> <u/> </Tags>
    </DiffContext>
  <DiffContext id="2">
    <Context class_name="Talk" ID="DESK"
      attribute_name="date" />
    <ReplaceBy> 21/03/2002 </ReplaceBy>
  </DiffContext>
  <DiffContext id="3">
    <Location place="after">
      <%= Bibliography %> </Location>
    <Insert> <blockquote>
      <h2> Related Bibliography </h2>
    </blockquote>
    </Insert>
  </DiffContext>
  .....
</ContextModel>

```

The XML code above shows the context model coming from the difference model. This model is built using the domain model for extracting information (e.g. attributes and classes where the modified literal is represented) located in the knowledge units. In this example, the appearance of the text "DESK Authoring Tool" is reflected on `DiffContext` number 1, where the class `Task` and attribute `title` is found in the domain model to describe such a difference. As we can see in the second difference context (`<DiffContext id="2">`) a change in the domain model has been modelled, locating the precise attribute and class where the difference happens. But in the third case (`<DiffContext id="3">`), the change has been found in the presentation model (the presentation template), where the new HTML code will be inserted right after the element `Bibliography` (`<Location place="after">`). This information is crucial because there is not a straight relation between the modified page and the original one, the location of insertions, deletions and changes being inferred directly by DESK.

After the difference and context model have been created, a third module will apply the context model to carry out every modification in the presentation as well as in the contents.

Once the changes in the presentation have been done, DESK returns a special report page showing a summary of the whole process, informing about any special events, errors, or warnings that might arise along the process. DESK also asks the user for confirmation, and prompting him/her for help if necessary to resolve ambiguities or taking hard decisions.

5. Conclusions

Our authoring tool provides automatic support for the customization of dynamic web documents based on comparing the pages generated by the system with a modified version provided by the end-user. DESK is based on PEGASUS, a system used to represent the semantic information structured by models that allow a clear separation between contents and presentation. DESK uses domain information stored in PEGASUS and presentation models for finding the context of changes made by user. Our authoring tool also determines whether the user is enabled to do these modifications depending on a user model. With DESK the user only needs to take care of editing HTML pages using any standard HTML editing tool such as PageMaker or Netscape Composer. Using rich structured models for representing differences and context makes each step easier and allows undoing changes. In the future we will use such models to store historic information about changes. At present, we are extending DESK to deal with more complex cases, like those involving presentation elements that are generated from JAVA expressions (as opposed to HTML literals) in presentation templates. We are also working on augmenting DESK with a WYSIWYG editor where all user actions are monitored during the edition process, thus providing the tool with more complete information to reason about, allowing for a more precise and correct response.

Acknowledgements

The work reported in this paper is being partially supported by the Spanish Interdepartmental Commission of Science and Technology (CICYT), projects numbers TEL1999-0181 and TIC2001-0685-C02-01.

References

1. 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.
2. 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.
3. Castells, P., Macías, J.A.: Un sistema de presentación dinámica hipermedia para representaciones personalizadas del conocimiento. *Actas del 2º Congreso de Interacción Persona-Ordenador (Interacción 2001)*. Salamanca, Junio 2001.
4. 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.
5. *Communications of the ACM. The Intuitive Beauty of Computer Human Interaction. Special issue on Programming by Demonstration*, 43, 3, March 2000.
6. Cypher A. (ed.). *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.
7. DAML - The DARPA Agent Markup Language Homepage, <http://www.daml.org>.
8. 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.
9. Macías, J.A. and 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.
10. Macías, J. A., Castells, P.: Adaptive Hypermedia Presentation Modeling for Domain Ontologies. To appear in *proceedings of 10th International Conference on Human-Computer Interaction (HCI '2001)*. New Orleans, Louisiana, 2001.
11. Murray, T.: Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. *Journal of the Learning Sciences* 7, 1, 1998, 5-64.
12. Muslea, I.: Extraction Patterns for Information Extraction Tasks: A Survey. *Proceedings of AAAI Workshop on Machine Learning for Information Extraction*. Orlando, Florida, July 1999.
13. Vassileva, J.: Dynamic Course Generation on the WWW. *Actas 8th World Conference on Artificial Intelligence in Education (AIED'97)*. Kobe, Japón, 1997, 498-505.
14. Weber, G. and Specht, M.: User modeling and Adaptive Navigation Support in WWW-based Tutoring Systems. *Proceedings 6th International Conference on User Modeling (UM97)*. Sardinia, Italy, 1997.