Using Domain Models for Data Characterization in PBE

José A. Macías and Pablo Castells
Escuela Politécnica Superior, Universidad Autónoma de Madrid
Ctra. de Colmenar Viejo km. 15
28049 Madrid, Spain
+34-91-348{2241, 2284}
{j.macias, pablo.castells}@uam.es

INTRODUCTION

One of the main problems in PBE inferencing is that of deriving abstract data characterizations from concrete objects and values involved in user's manipulations [8]. This involves selecting variables and constants (parameterization) among involved values, and establishing what variables represent. Some variables will be input parameters of the construct or procedure inferred by the PBE system, and some will be expressions that indicate values or objects that are obtained from input parameters by traversing a data structure made of relations, lists and attributes, and operations on them. Our contribution to this workshop sets the focus on the use of rich descriptions of application data to achieve correct and expressive data characterizations.

Data models, or the wider and more conceptual notion of domain models [21], provide an understanding of the knowledge behind the visual representations that the user manipulates in a PBE system. This knowledge can be extremely useful to make sense of the user's actions on visual objects. It can help, for instance, select variables, find relations between visual objects, focus attention on special objects, mark out composite display units, build complex data flow expressions, and disambiguate the intended meaning of user's actions. How to best exploit this knowledge; how to keep track of the relation from visual objects back to the domain knowledge items they correspond to; how much of the domain model should the user be exposed to an in what form; what kind, if any, of underlying representation should be used for the user interface, are a few of the difficult problems and issues that we will put forward and discuss here.

VISIBLE APPLICATION DATA EXAMPLES

Our previous experience in this context includes research on an interface development environment, HandsOn [3, 4], where 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. [3] shows how HandsOn can be used to generate the well-known Minard chart showing Napoleon's march to Moscow (partially shown in Figure 1), consisting of a sequence of line segments whose thickness, color, and endpoints represent the number of troops in Napoleon's army, the temperature, and the geographical coordinates respectively.

Rather than building a generic display, the designer constructs specific displays using specific data and HandsOn generates abstract constructs by generalizing the examples. The data examples disappear when the PBE system infers generic displays, where concrete values are replaced by variables and expressions whose values are computed at runtime.

In HandsOn the user is exposed to the application data model through an explicit view of data examples next to the interface design area (see Figure 1). The design tool allows connecting data to visual components by pointing at and dragging data and display elements. Data examples provide the designer with concrete objects to refer to, and they provide the system with information that the system uses to infer the designer's intent.

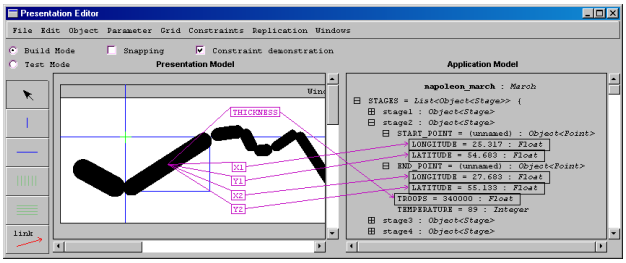

Figure 1. Linking interface objects to data examples in HandsOn

HandsOn analyzes the types and structural properties (e.g. iteration and recursion) of the data to automatically generate presentation constructs. In doing so, the system also examines visual properties and geometric relationships among the objects being manipulated, existing mappings from data to presentations, and the way data is being seen by the designer (e.g. expanded nodes, selected values, focused structures). For instance, if a visual object displays a value that belongs to a list, HandsOn suggests to create a list of visual objects to display the remaining list values. Similarly, recursive display structures can be generated for recursive data structures.

Overall, the types of decisions that HandsOn is able to make include:

- Mapping data structures to custom display structures.
- Generating and adjusting transformation functions (e.g. scaling, type conversion) between display parameters (e.g. endpoints of a line) and application values (e.g. geographic coordinates).
- Propagating changes over replicated objects across display structures.
- Replacing example values by variables and expressions when the design is finished.

HandsOn uses a highly structured and sophisticated internal model of interface displays, based on the presentation model of an existing model-based GUI development tool, Mastermind [2], which supports dynamic presentation functionalities. HandsOn was entirely implemented in Amulet [18], a high-level user interface toolkit that provides a) a constraint system that we used to link display components to application data, and b) an easily inspectable object-based representation of user interfaces that facilitates the exploration of the interface structure.

ONTOLOGY-BASED DOMAIN MODELS FOR DYNAMIC WEB PAGE AUTHORING

In some cases it may be useful to extend the notion of data model to the more conceptual notion of domain model [19], because 1) it provides an even richer description and a deeper understanding of the application knowledge behind what the user sees, and 2) the usage and availability of explicit domain models is becoming increasingly common in knowledge-based applications on the WWW [1]. Under this view, the notion of knowledge base takes the place of application data.

The generalization of the WWW as a universal computing platform, and the unprecedented size of application user communities it bears, has motivated us to take it as an interesting ground for end-user programming research. Starting from the observation that most of today's WWW is made of dynamically generated web pages, and the fact that development of dynamic pages is considerably difficult and requires advanced programming skills, we have taken up the challenge of devising an interactive authoring tool where dynamic web pages can be edited in a WYSIWYG environment similar to a standard HTML editor [15, 23].

In order to tackle such a difficult problem, we have considered the assumption that a domain model and a presentation model are available. We assume an ontology-based domain description [9], and a specification of presentation on a per-class basis. We believe these assumptions are congruent with important trends in current web technology. The emerging semantic web view [1] promotes the construction and widespread availability of explicit models of domain knowledge. Simultaneously, a major recurring motto in the development of the new web technologies is the separation of data (domain knowledge) and presentation (e.g. XML+XSLT, see also XMLC [24], Cuypers [19], PEGASUS [5], to name a few).

The DESK Authoring Tool

We have developed a tool, DESK [13, 14], where authorized users can customize web page generation procedures by editing specific HTML pages produced by a dynamic page generation system. DESK acts as a client-side complement of a dynamic web page generation system, PEGASUS [5, 12], which generates HTML pages from an ontology-based 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 uses the PEGASUS domain model to a) identify pieces of domain contents in the edited page, b) establish relations between them, c) select one (or more) of the involved knowledge items as the root domain object behind the web page, from which all other objects are referred to as relative to this one, d) detect iteration patterns when the user lays out data over structured displays (e.g. records in a table).

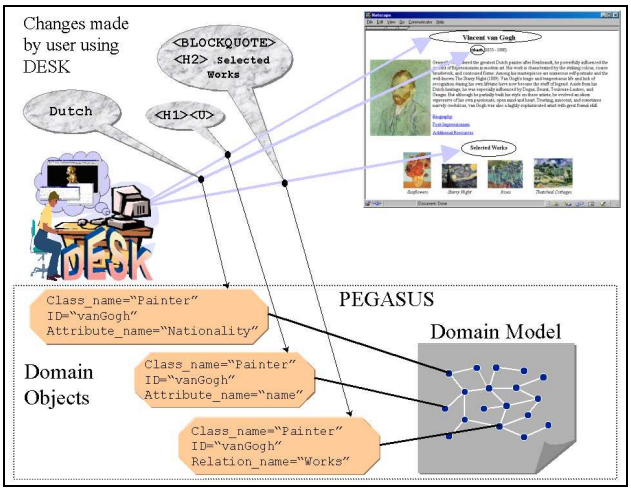

Figure 2. Detecting correspondences between page blocks and domain objects with DESK

For instance, consider a web page like the one shown on the upper-right corner of Figure 2, where information about Vincent van Gogh is presented. Assuming an ontology has been defined in PEGASUS for a virtual museum or a course on history of art, including classes like Painter, Painting, School, and so on, DESK is able to find that this page is displaying attributes (name, birth, short biography) and relations (works, school) of an instance of Painter. If the user adds text, changes the style or the position of a piece of the document (e.g. the thumbnail image on the lower-right corner of the web page in Figure 2), DESK finds a description of this piece that relates it to the main object (van Gogh) in terms of the vocabulary defined by the application domain ontology (e.g. "the *small-image* attribute of the last element in the *selected-works* relation of the object with ID

vangogh"). This information is used by DESK to modify the presentation model for class Painter (or class Painting if appropriate), so that the change is permanent for all objects of class Painter. The van Gogh instance acts as an example for the user to see and change how a painter presentation (by PEGASUS) looks like, and DESK generalizes the modification to the whole instance class.

Reverse Engineering

PEGASUS generates web pages on the fly from a semantic network of ontology instances (the application data/knowledge) as the user implicitly requests viewing domain objects. These requests are internally generated from the navigational interaction of the user with an application supported by PEGASUS. To present an object, PEGASUS finds its class and applies the presentation model associated to the class to generate a web page where selected pieces of the object are displayed. DESK follows the inverse path: it parses the web page and locates the source of page fragments in the domain model, as well as the part of the presentation model that defines how the fragment was presented.

This backward transition from syntactic blocks to semantic blocks can be seen as a reverse engineering problem, and as such is a non-trivial task [22]. Our current approach is based on a simple search of text and multimedia fragments in the domain knowledge base. Devising a smarter search is an open issue in our work. Other main difficulties are cutting out the right syntactic blocks in the page to be found in the KB, and removing the ambiguity when the search yields multiple results. To solve the latter, DESK uses heuristics such as requiring that found contents are connected to each other in the domain model, and priorizing the closeness of knowledge units in the semantic network. We carry out the former by looking for hints in user actions (e.g. selection of blocks), domain contents (e.g. readjust block boundaries when the search yields a partial match), and the syntactic structure of the display (e.g. paragraphs, table cells, etc.).

DESK uses an implicit display model based on different kinds of pre-programmed presentation widgets such as tables, selection lists, combo boxes, trees and so forth, as supported by HTML. Because the most flexible construct for structured layout in HTML are tables, an important part of our work in DESK is concerned with specific strategies for treating complex mappings from application data structures to nested tables. The considerable number of research works related to table analysis and interpretation that can be found in the literature [6, 7, 10] proves that table parsing is a difficult problem and an interesting object of study by itself. We believe that the introduction of models of domain knowledge in this frame brings about interesting views on the problem, that are particularly pertinent in the context of web applications and HTML as a standard for web user interface presentation. In particular, while other systems infer data structures from tables in HTML documents, in DESK the data structure description is taken from a dynamically built structured model of user's actions related to domain information.

DISCUSSION

Building dynamic information visualization interfaces from examples requires elaborate data characterizations when the underlying domain knowledge has a complex structure, as is the case in many knowledge-based web applications and information systems. The usage of ontologies, i.e. explicit descriptions, to organize and share knowledge in such systems is becoming an increasingly popular approach. We propose to exploit these explicit models of domain knowledge, which are available for free (from the PBE system developer point of view), to improve the reach and precision of PBE techniques, and in particular as a highly valuable source of information for data characterization.

The use of a data model was already present in one of the earliest PBE systems, Peridot [16], in a very simple form. Peridot lets the user create a list of sample data to construct lists of user interface widgets. In Gold [17] and Sagebrush [20] the user can build custom charts and graphics by relating visual elements and properties to sets of data records. The data model in Peridot consists of lists of primitive data types. Gold and Sagebrush assume a relational data model. Our view in this regard is that it is interesting to lift these restrictions and support richer information structures, as proposed in our current and earlier research work.

One interesting issue when domain or data models are used in a PBE system is whether and to what extent the model should be visible for the user. There is a whole range between completely hiding the domain model and showing a full literal (abstract) view of it. Moving along this axis means trading simplicity for expressive power. For instance, HandsOn does show data, but in the form of specific examples, easier to have in mind and manipulate than an abstract model. The explicit manipulation of data in HandsOn has an additional advantage: keeping track of the relation from visual objects to data is not a problem, as all these links are defined by the user in the system, so that HandsOn can store and remember them. In DESK the data-presentation relation is known by the page generation system, PEGASUS, but this information is lost when DESK gets the generated page, and has to be recovered by the PBE system in a difficult and costly reverse engineering process.

DESK uses a more expressive model of application knowledge that HandsOn, but completely hides it from the user to stay within the strict WYSIWYG principle, thus requiring zero awareness from the user of the internal knowledge representation. In exchange, DESK has important expressive limitations: it is not possible to modify the way visual components are linked to domain objects, and it is not possible to add new object part presentations that are not already present in a page. This means, in particular, that it is not possible to build a presentation from scratch, and the user can only customize existing designs. We have followed this approach as an experiment to investigate how far one can go without giving up on the WYSIWYG requisite, but there is nothing that impedes augmenting DESK with views of the domain model to provide the expressive capabilities

needed to remove these limitations, except a compromise in simplicity of use.

Other sources of information for PBE inferencing used by DESK, besides models of application domain knowledge, include spatial properties and relations in the edited display, knowledge about page design practice (page layout, table layout, standard spatial patterns), know-how about interactive design manipulation, and structure in user's actions and behavior (e.g. order and sequencing, iterative patterns). For the underlying representation of the visual constructs being created or modified by example, DESK does not need such a sophisticated model as HandsOn does because a) the editing functionalities and gestures needed to manipulate data examples and interface components in the HandsOn environment require a detailed description of the involved objects and imply frequent readjustments in data flow relations, whereas DESK provides more limited capabilities based on HTML editing, and b) the run-time implementation platform for target constructs in HandsOn is a full-fledged window-based toolkit (Amulet), while in DESK the (PEGASUS) interface model essentially builds upon the much simpler HTML user interface model.

ACKNOWLEDGMENTS

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

REFERENCES

1. Berners-Lee, T., J. Hendler, O Lassila. The Semantic Web. Scientific American, May 2001.
2. Castells, P., Szekely, P., Salcher, E. Declarative Models of Presentation. International Conference on Intelligent User Interfaces (IUI'97). Orlando, 1997, pp. 137-144.
3. Castells, P., Szekely, P. Presentation Models by Example. In: Duke, D.J., Puerta A. (eds.): Design, Specification and Verification of Interactive Systems '99. Springer-Verlag, 1999, pp. 100-116.
4. Castells, P., Szekely, P. HandsOn: Dynamic Interface Presentation by Example. Proceedings of the HCI International'99. Munich, 1999, pp. 188-1292.
5. Castells, P., Macías, J.A. An Adaptive Hypermedia Presentation Modeling System for Custom Knowledge Representations. World Conference on the WWW and Internet (WebNet'01). Orlando, 2001, pp. 148-153.
6. Chen, H.; Tsai, S.; Tsai, J. Mining tables from large sale HTML texts. 18th International Conf. on Computational Linguistics (COLING), 2000, pp. 166-172.
7. Cohen, W. et al. A flexible Learning System for Wrapping Tables and Lists in HTML Documents. WWW Conference. Honolulu, 2002, pp. 232-241.
8. Cypher A. (ed.). Watch What I Do: Programming by Demonstration. The MIT Press, 1993.
9. Gruber, T. R. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2), pp. 199-220, 1993.
10. Hurst, M. F. The Interpretation of Tables in Texts. PhD. Thesis. University of Edinburgh, 2000.
11. Little, S., J. Geurts, and J. Hunter. Dynamic Generation of Intelligent Multimedia Presentations through Semantic Inferencing. 6th European Conference on Research and Advanced Technology for Digital Libraries (pages 158-189). Springer, 2002.
12. Macías, J.A., Castells, P. A Generic Presentation Modeling System for Adaptive Web-based Instructional Applications. ACM Conf. on Human Factors in Computing Systems (CHI'2001). Seattle, April 2001.
13. Macías, J.A., Castells, P. Dynamic Web Page Authoring by Example Using Ontology-Based Domain Knowledge. International Conference on Intelligent User Interfaces (IUI'2003). Miami, Florida. January 2003, pp. 133-140.
14. Macías, J.A., Castells, P. DESK-H: building meaningful histories in an editor of dynamic web pages. To appear in 11th International Conference on Human-Computer Interaction (HCII'2003). Creta, Grece, June 23-27, 2003.
15. 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.
16. Myers, B. A. Creating User Interfaces by Demonstration. Academic Press, San Diego, 1988.
17. Myers, B. A., J. Goldstein, M. Goldberg. Creating Charts by Demonstration. Proceedings of the CHI'94 Conference. ACM Press, Boston, April 1994.
18. Myers, B. A., et al. The Amulet 2.0 Reference Manual. Carnegie Mellon University Tech. Report, 1996.
19. Ossenbruggen et al. Towards 2nd and 3rd Generation Web-Based Multimedia. 10th International World Wide Web Conference. ACM Press, 2001, pp. 479-488.
20. Roth, S. F. et al. Interactive Graphic Design Using Automatic Presentation Knowledge. CHI'94 Conference. Boston, April 1994, pp. 112-117.
21. Spyns, P, R. Meersman, M. Jarrar. Data modelling versus Ontology engineering. SIGMOD Record 4, Dec. 2002.
22. Vanderdonckt, J., Bouillon, L., Souchon, N. IEEE 8th Working Conf. on Reverse Engineering Stuttgart, October 2001. IEEE Press, pp. 241-248.
23. Wolber, D., Su, Y., Chiang Yih. Designing Dynamic Web Pages and Persistence in the WYSIWYG Interface. International Conference on Intelligent User Interfaces (IUI'2002). San Francisco, January 2002, pp. 228-229.
24. Young D. H. Enhydra XMLC Java Presentation Development. Sams, 2002.