Mining Design Patterns in the Conceptual Schema of Web Applications

Dimitra Dimitrakopoulou [2], Theodora Katsimpa [1,2], Maria Rigou [1,2], Spiros Sirmakessis [1], Athanasios Tsakalidis [1,2], Giannis Tzimas [1,2]
[1]Research Academic Computer Technology Institute
Internet & Multimedia Technologies Research Unit,
61 Riga Feraiou str., GR-262 21 Patras, Hellas
&
[2]University of Patras, Computer Engineering and Informatics Department
GR-26504, Rio Patras, Hellas

Abstract: The tremendous evolution of Web applications in several productive contexts of our society is laying the foundations of a renewed scenario of software development, where software industry has to invent new approaches for designing and maintaining Web applications. To this end, software community has proposed a variety of modeling methods and techniques. In this work, we propose a methodology for mining design patterns in the conceptual schema of an application modelled using WebML, a modeling language for designing data-intensive applications. We extend the set of design patterns provided by WebML and illustrate a validation experiment based on an e-learning scenario.

Introduction

In the Internet era, the development of Web applications has impressively evolved. In the new scenario, Web applications are becoming the underlying engine of any e-service, including e-learning, CSCW, e-business and e-government. The hypertext architect has to design the application in such a way, that it can efficiently manage huge amounts of data, integrate complicated functions and sophisticated business logic, trying in the same time to provide access to users with different preferences and needs, using a variety of access devices including mobile ones.
Novel challenges are therefore posed to developers. As the market needs increase rapidly and the use of a large number of new technologies evolves at a quick pace, advanced Web application development becomes more and more slow, expensive and error prone often yielding products with large numbers of defects, causing serious problems of usability, reliability, performance, security and other qualities of service.
In order to cope with the high market pressure for quality Web application development and to provide efficient and disciplined development processes and tools able to model the increased technological complexity, the software community has proposed in the last few years several methods and techniques. Hypermedia applications inspired the proposal of RMM and HDM, which pioneered the model-driven design of hypermedia applications and influenced several subsequent proposals like HDM-lite, a Web-specific version of HDM, Strudel and OOHDM (Fraternali 1999). Araneus (Atzeni et. al. 1998) is a proposal for Web design and reverse-engineering. Some extensions to the UML notation have been proposed by Conallen to make the UML language suitable for modeling Web applications (Conallen 1999). Finally, WebML (Ceri et. al. 2002) provides a methodology and a notation language for specifying complex Web sites and applications at the conceptual level and along several dimensions.
The remainder of this paper is organized as follows: Section 2 describes the motivation of our work. Section 3 provides a quick overview of WebML. Section 4 describes a methodological approach for identifying patterns within the conceptual schema of a Web Application, while Section 5 illustrates a validation experiment of the proposed methodology in an instance of an e-learning scenario. Finally, Section 6 provides concluding remarks and discusses future steps.

Motivation

In the early days of e-learning, the emphasis was on providing access to content - tutorials, presentations, e-books, etc. Nowadays, however, providing content online is not the main target, importance is also given on providing

opportunities to learners and teachers to communicate, discuss and collaborate online - either one-to-one or in groups - not necessarily at the same time and certainly not necessarily in the same physical space.
In an effort to keep up with the newly emerged e-learning requirements, modern web-based learning applications have evolved to data-intensive applications and now comprise apart from support for multiple roles of teaching and learning remotely, typical CSCW functionalities. They support site and user management options, course management, assignments mechanism, community tools, as well as collaboration features.
The design and implementation of such applications is a time consuming process that involves many experts coming from both the IT and the pedagogical/social studies field. Current practice has indicated that a lot of effort, time and thus money are wasted on re-designing, re-implementing and re-debugging similar functional e-learning components. This situation calls for a platform-independent modeling, able to communicate the application's functional requirements, design and implementation aspects to the participating teams of experts throughout the various development phases. Moreover, a modeling approach will also provide for consistency, reusability and increased future productivity, taking full advantage of the gained experience and good past practices. WebML featured as a quite appealing choice for our setting, mostly due to the fact that it is specifically designed for data-intensive web applications, it is orthogonal in nature and supports the modeling of personalized views and the definition of user profiles and groups.
Taking the modeling approach one step further, in this work we claim that advantages can be multiplied if apart from modeling complete e-learning applications one could also identify, model and re-use specific functional components of such systems. To this end, we deploy the notion of design patterns to introduce what we call application-domain specific patterns and investigate their model-level identification (i.e. their identification at design time, based on the WebML model of the application). Capturing these patterns at such an early phase in the development cycle allows for major productivity profits.

An Overview of Web Modeling Language - WebML

WebML (Ceri at al. 2002) is a visual language for specifying the content structure of a Web application and the organization and presentation of contents in one or more hypertexts. In order to design a Web application using WebML, the designer at a first level specifies the entities and relationships that constitute the data schema of the application using the E-R model, thus organizing the content. The next step is hypertext design, where the designer identifies schemes expressing the content composition into pages and content units, specifies the way units and pages are connected in order to form a hypertext and maps units to the main entities and relationships of the data schema. Site views, areas, pages and content units constitute the overall structure of the hypertext. A *site view* is a hypertext, designed to address a specific set of requirements, according to the different users or the different publishing devices. It is partitioned into *areas*, which in turn may consist of other *sub-areas* or *pages* with a homogenous purpose. Pages are the actual containers of information delivered to the user; they are made of content units that are elementary pieces of information, extracted from the data sources by means of queries and which are published within pages. In particular, as described in (Tab. 1), content units denote alternative ways for displaying one or more entity instances. In order to specify a *data unit*, the designer should define a *source* (the name of the entity from which the unit's content is extracted) and a *selector* (a condition, used for retrieving the actual objects of the source entity that contribute to the unit's content).

Data unit	Multidata unit	Index unit	HierarchicalIndex	Scroller unit	Delete	Modify
Entity [conditions]	Entity [conditions]	Entity [conditions]	Entity1 [Selector1] NEST Entity2 [Selector2]	Entity [conditions]	Entity [conditions]	Entity [Conditions] <param := value>
It displays a set of attributes for a single entity instance.	It displays a set of instances for a given entity.	It displays a list of properties of a given set of entity instances.	It is a variant of index, in which the index entries are organized in a multi-level tree.	It represents a scrolling mechanism for the elements in a set of instances.	It deletes one or more objects of a given entity	It updates one or more objects of a given entity.

Table 1: Some basic WebML units. The whole set is described in (Ceri et al. 2002).

Formatted: Centered

Within site views, content units and pages are interconnected in a variety of configurations by means of links, yielding to composite navigation mechanisms. Besides representing user navigation, links between units also specify the transfer of certain information (called context) that the destination unit uses for selecting the data instances to be displayed. Apart from content publishing, WebML allows specifying data update operations, like the creation, modification and deletion of instances of an entity, or the creation and deletion of instances of a relationship (Tab. 1). Operations do not display data and are placed outside pages. Special purpose operations, such as e-mail sending, login, logout, and e-payment, can also be specified. (Ceri at al. 2002)

A Methodology for Extracting Design Patterns

The notion of design patterns as tools that describe a piece of design experience and/or expert advice and make it reusable, was initially conceived by the architect C. Alexander, in the context of architecture and urban planning (Alexander et al. 1997): "*… Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem, in such a way that you can use this solution a million of times over…*". Nowadays, the use of patterns has been further extended in diverse domains. In the field of software engineering, design patterns are increasingly used to capture expertise in object-oriented programming (Gamma et al. 1995). In this case, design patterns capture common design problems, study different solutions, document good designs that solve problems faced during any serious software development project and analyze the trade-offs. More recently, design patterns have been introduced in the Web modeling field as well, for describing the navigation and structure of Web applications (Schwabe et al. 1997, Bernstein 1998, Nanard et al. 1998, Garzotto et al. 1999). The availability of design patterns, which offer verified solutions to typical page configuration requirements, further facilitates the task of the hypertext architect and enforces a coherent design style over large and complicated applications, augmenting hypertext regularity and usability.

A primitive set of design patterns has already been identified in WebML, comprising compact and consistent, one-step solutions, applicable in real-life scenarios of Web applications. WebML introduces patterns for data design, hypertext design (cascaded index, filtered index, filtered scrolled index, guided tour, indexed guided tour, object viewpoint, nested data, hierarchical index with alternative sub-pages), as well as patterns for content management operations (object creation-deletion- modification, relationship creation-deletion, create-connect pattern, cascaded delete) (Ceri et al. 2002, Ceri et al. 2004). A *pattern* in WebML, typically consists of a *core specification*, representing the invariant WebML unit composition that characterizes the pattern, and a number of *pattern variants*, which extend the core specification with all the valid modalities in which the pattern can start (*starting variants*) or terminate (*termination variants*). Starting variants describe which units can be used for passing the context to the core pattern composition. Termination variants on the other hand, describe how the context generated by the core pattern composition is passed to successive compositions in the hypertext (Fraternali et. al. 2002).

In what follows, we present in detail a methodology for extending the primitive set of design patterns as identified in WebML. Our objective is to capture patterns in the process of modeling a Web Application. We call these patterns *Application-Domain Specific Patterns (ADSPs).* We claim that our methodology, if applicable to the conceptual schema of a domain-specific Web Application, is more efficient, since the probability of capturing patterns within applications of similar context increases.

ADSPs are constructs (of units, pages, areas and sub-areas) within the conceptual schema of a specific-domain Web application modelled with WebML, that when applied to various instances of the schema, can solve a specific problem in an optimal way. As in the case of predefined WebML patterns, they also have starting and termination variants.

A first (and modest) approach towards capturing the occurrences of such patterns within the hypertext schema of a specific Web Application modelled in WebML, would be the following:

Let SV_1, SV_2,…, SV_N be all the site views of the conceptual schema.

By taking the intersection of all the site views in pairs, we compute various sets of candidate ADSPs, that can be further intersected, so as to retrieve and identify a larger set of ADSPs. Thus, a first set of candidate patterns is captured. The obtained set so far is rather small and incomplete when taking into account the pattern variants that exist in a conceptual schema and the fact that we have not considered the existence of the predefined WebML Patterns. A more sophisticated approach to capture ADSPs follows:

Step 1: Iteratively, we traverse each site view of the Web Application conceptual schema and search for occurrences of predefined WebML patterns (patterns for content publishing and content management). Fig. 1 depicts the retrieval of a cascaded unit and a filtered index, within a site view. This can be achieved using

XSL (Fraternali et al. 2002). The XSL language (XSL 2001) allows writing pattern-matching rules that can be applied to an XML document for generating a new XML document. Each rule contains a matching part for selecting the target XML elements, and an action part to transform the matched elements. The XSL documents serve therefore the purpose of extracting the instances of patterns from the XML specification of the WebML conceptual schema. It should be mentioned that when traversing the site views for capturing the predefined patterns, the various pattern variants are taken into account. Every pattern found is stored in a *pattern occurences repository*, along with its starting and termination variants. The repository also stores the frequency of each pattern occurrence.

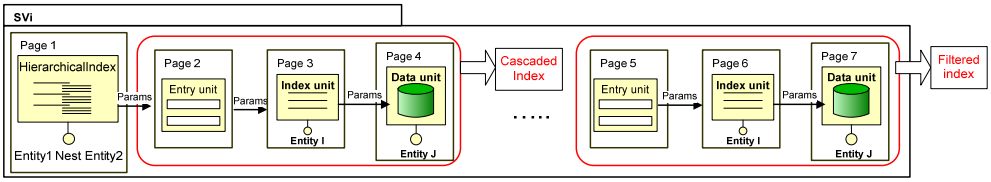

Figure 1: Retrieval of predefined WebML patterns in a site view

Step 2: The purpose of this step, is to create a more uniform conceptual schema of the Web application thus enabling the easier extraction of ADSPs in the steps to follow. Taking into account the various predefined WebML pattern variants, and utilizing XSL rules, we substitute, where possible, the variants found within each site view with a default pattern variant. The default pattern variant is the one having the maximum occurrence frequency (in case that more than one patterns are assigned the maximum frequency, we choose the variant found first). Upon completion of this step, we have a new generated XML definition of each site view.

Step 3: We traverse each newly generated site view in order to locate the same constructs (units and operations navigation chains) along with their variants that do not already belong to the predefined WebML set of patterns. This way, we extract a first set of candidate ADSPs per site view. Fig. 2 depicts the retrieval of a pattern within a site view. The various candidate ADSPs identified are also stored in the patterns repository, along with their frequency. Following the same procedure as in step 2 we compute a new XML definition of each site view substituting, where possible, the variants with the default pattern variant aiming to create a more canonical schema.

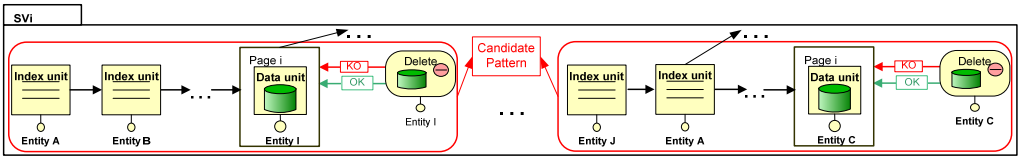

Figure 2: Retrieval of an ADSP within a site view

Step 4: Up to this point, we have located the ADSPs of each site view independently and have stored them in the patterns repository. In this step we traverse each area, sub-area and page of all the site views of the WebML conceptual schema (by comparing site views pair-wise) in an effort to extract ADSPs that exist in different site views of the schema but have not yet been identified when traversing a single site view. If two site views contain the same navigation chains of units or operations, we claim to have found one more candidate pattern and store it in the repository along with its frequency. Fig. 3 represents the identification of a candidate ADSP that has been retrieved comparing two site views of the conceptual schema. Again, pattern variants are substituted with the default pattern variant, where possible.

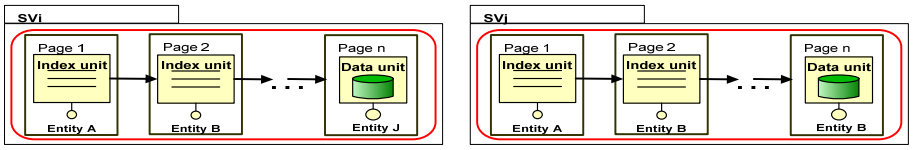

Figure 3: Retrieval of an ADSP from different site views

Step 5: We repeat steps 3 and 4 in order to capture larger –possibly combinations of- ADSPs, taking into account the pattern variants identified in each iteration, until no new pattern is found.

Step 6: We examine every ADSP within the repository and in case it contains a predefined WebML content management pattern, but one or more of the remaining patterns is missing, we add the respective missing patterns. For instance, if we locate an ADSP whose termination variant is a create pattern, we complement it by adding the modify and/or delete pattern(s).

Step 7: The various patterns stored in the repository, are further clustered according to the core, access, interconnection and personalization subschemas and sorted according to their occurrence frequency. So far, we have succeeded in making a first categorization of the patterns.

Upon completing of the above steps, the designer has access to a library that contains a set of ADSPs along with their variants. This library is composed of basic patterns and combinations of larger patterns that can be extended (in terms of usage) by defining new variants.

In order to acquire a larger set of ADSPs and to improve their consistency, the same methodology can be applied to conceptual models of different Web applications in the same application domain. Consistent patterns allow users to integrate past experience into future explorations or, even better, to predict how an unfamiliar section of the application will be organized (Fraternali et. al. 2002).

A Validation Experiment: The e-Learning Scenario

In order to exemplify the application of the above mentioned methodology, we refer to an instance of an e-learning scenario, in which we identify an application-domain specific pattern.

In the example depicted in the following figures, we capture an ADSP, traversing two distinct site views of an e-learning application. The first is a student's site view which consists of an Announcement area and the second is a teacher's site view including a Forum area.

Comparing the two site views, we can easily identify the existence of an ADSP. This pattern consists of a hierarchical index unit followed by an index unit, which is in turn followed by a scroller unit that enables the paging process and is linked through an automatic link with a multidata unit. When applying the methodology, this pattern is stored in the repository. Apart from this pattern, in the case of the teacher's site view, two variants of the previously identified pattern can be extracted. One of the variants extends the pattern with an object creation pattern (as shown in (Fig. 5), the multidata unit is linked with an entry unit used to supply values to a create unit) and the other one extends the pattern with an object modification pattern (the multidata unit is linked with an entry unit used to supply values to a modify unit).

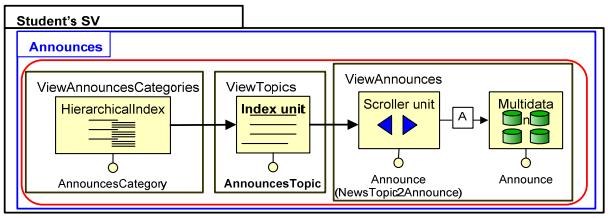

Figure 4: Student's site view

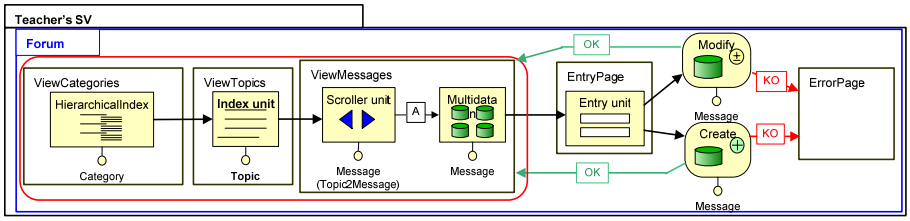

Figure 5: Teacher's site view

Since pattern variants containing content management patterns have been retrieved, we should extend the repository with variants derived by the missing content management patterns. For instance, a variant containing the object deletion pattern in place of the object creation pattern should be stored in the repository, as well as all the possible combinations computed using all the content management patterns.
Thus, although we have retrieved only one common navigation chain, we have constructed and stored in the patterns' repository more ADSP variants.

Conclusions and Future Work

This paper introduced the notion of application-domain specific patterns and illustrated a methodology for mining them during the process of (or after) modeling the conceptual schema of a Web Application designed using WebML. ADSPs complement WebML patterns in the process of modeling a Web application and can form a valuable tool for hypertext architects in future application development, as well as for the effective redesign of various applications within a specific domain.
In the future, we plan to extend the methodology by means of supporting the quality evaluation of a Web application. Moreover, we will extend the patterns recognition mechanism based on their semantic context, in order to support automatic mining and usability evaluation of the ADSPs, since to this point the architect has to make the final decision on the usefulness and the variants of an identified design pattern.

References

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. & Angel, S. (1997). *A Pattern Language*. Oxford University Press, New York.
Atzeni, P., Mecca, G., & Merialdo, P. (1998). Design and Maintenance of Data-Intensive Web Sites. *Proc. EDBT*, (pp. 436-450).
Bernstein, M. (1998). Patterns of Hypertext, *Proc. of HyperText'98*. Pittsburgh, PA..
Ceri, S., Dolog, P., Matera, M., & Nejdl, W. (2004). Model-Driven Design of Web Applications with Client-Side Adaptation. *Proc. of ICWE'04*, Munich, Germany, LNCS 3140, Springer Verlag.
Ceri, S., P. Fraternali, A. Bongio, M. Brambilla, S. Comai, & M. Matera (2002). *Designing Data-Intensive Web Applications*. Morgan Kauffmann.
Ceri, S., Fraternali, P., & Matera, M. (2004). Conceptual Modeling of Data-Intensive Web Applications, *IEEE Internet Computing*, *6*(4), (pp.20-30).
Comai, S., Matera, M., & Maurino, A. (2002). A Model and an XSL Framework for Analysing the Quality of WebML Conceptual Schemas. *Proc. of IWCMQ'02 - ER'02 International Workshop on Conceptual Modeling Quality*. Tampere, Finland.
Conallen J. (1999). Modeling Web application architectures with UML. Communications of the Association for Computing Machinery 42(10): (pp. 63–70).
Fraternali, P., Matera, M., & Maurino, A. (2002). WQA: an XSL Framework for Analyzing the Quality of Web Applications. *Proc. Of IWWOST'02, Malaga, Spain*.
Fraternali, P. , Paolini, P. (1998). A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. *Proc. EDBT 1998* (pp. 421-435).
Gamma, E., Helm, R., Johnson, R. & Vlissedes, J. (1995). *Design Patterns - Elements of Reusable Object Oriented Software* (Addison Wesley).
Garzotto, F., Paolini, P., Bolchini, D. & Valenti, S. (1999). Modeling-by-Patterns of Web Applications. *In*
Rossi, G., Schwabe, D., Lyardet F. (1999). Web Application Models are More than Conceptual Models. *in eds, Proc. Int. Workshop on the World Wide Web and Conceptual Modeling* (pp. 239-252). Paris.
Schwabe, D., Garrido, A. & Rossi, G. (1997). Design Reuse in Hypermedia Applications Development. *Proc. of ACM Hypertext '97* (pp. 57-66). Southampton, UK
Nanard, M., Nanard, J. & Kahn, P. (1998). Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. *Proc. of Hypertext'98* (pp. 11-20). Pittsburgh, PA.
XSL. Extensible Style sheet Language (W3C Recommendation, http://w3.org/TR/XSL/, October 2001).
Fraternali, P. (1999). Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. *ACM Computing Surveys*, Vol. 31, No. 3, (pp. 227-263).