Proceedings of the IASTED International Conference
WEB-BASED EDUCATION
February 21-23, 2005, Grindelwald, Switzerland

CAPTURING APPLICATION-DOMAIN SPECIFIC PATTERNS IN A WEB APPLICATION: THE E-LEARNING PARADIGM

Dimitra Dimitrakopoulou [2], Maria Rigou [1,2], Spiros Sirmakessis [1,3], 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
&
[3]Technological Educational Institution of Messolongi,
Nea Ktiria, GR-30200, Messolongi, Hellas
{dimitrad;rigou;syrma;tsak;tzimas}@ceid.upatras.gr

ABSTRACT
Designing and maintaining Web applications is one of the major challenges that software industry has to face. Several modeling techniques have been proposed to support this process. In this work we present a methodology for identifying design patterns within an application modelled using WebML, a modelling language for designing data-intensive Web applications. We extend the set of design patterns supported by WebML and exemplify the application of the methodology using an e-learning scenario.

KEY WORDS
Modeling, Design Patterns, Application-Domain Specific Patterns, E-learning, WebML

1. Introduction

Software development -and especially advanced web application development incorporating business logic- as currently practiced, is 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 [1]. This results not only from the fact that modern applications have moved to an extreme degree of sophistication, they manage huge amounts of (usually heterogeneous) data, but they also incorporate complicated business rules and generate personalized web views on-the-fly (thus suffering from excessive speed requirements). It also stems from the need to form large teams of programmers and put them to work jointly on the various subsystems and their integration. As noted by Brooks [2], adding people to a project eventually yields diminishing marginal returns. The amount of capacity gained by recruiting and training developers will fall off asymptotically. The new have to learn from the "old" and ideally from the best 'old' have to offer. We know from experience that there will never be more than a few extreme programmers. The best developers are up to a thousand times more productive than the worst, but the worst outnumber the best by a similar margin [3]. The solution must therefore involve changing methods and practices in a way that allows for making the development of large-scale data intensive applications much more productive. Historically, paradigm shifts have raised the level of abstraction for developers, providing more powerful concepts for capturing and reusing knowledge in platforms and languages [4].

In the early stages the prevailing approach to Web application development, was simply "building the solution", with little emphasis on the development process itself. However, many organizations are now experiencing severe problems in the management of Web sites, as they grow in size and complexity, inter-operate with other applications, and exhibit requirements that change over time.

To this end, several Web application modeling methods have been proposed, based on the key principle of separating data management, site structure and page presentation. HDM pioneered the model-driven design of hypermedia applications and influenced several subsequent proposals like HDM-lite, a Web-specific version of HDM, RMM, Strudel, and OOHDM [5].

Araneus [6] is a proposal for Web design and reverse-engineering, in which the data structure is described by means of the E-R Model and navigation is specified using the Navigation Conceptual Model (NCM). Conceptual modeling is followed by logical design, using the relational model for the structural part, and the Araneus Data Model (ADM) for the navigation aspects. Araneus also includes predefined navigation primitives and does not model presentation at an abstract level.

OOHDM (Object-Oriented Hypermedia Design Method) [7] is concerned with the conceptual modeling, navigation

design, interface design, and implementation of hypermedia applications. Navigational contexts in OOHDM provide a rich repertoire of fixed navigation options.

There exist several proposals for using UML [8] for modeling the architecture of web applications. Web pages are modeled as UML components, distinguishing among their "server side aspects" (i.e., their relationship with middle tiers, databases, and other resources) and their "client side aspects" (i.e., their relationships with browsers, Java applets, ActiveX controls and so on).

UIML is a user interface language designed for abstracting from appliance-specific details while designing the user interface of a Web application. UIML designers model all aspects of a user interface in XML, so that only a portion of the specification (the style section) is appliance-dependent.

WebML [9] builds on several previous proposals for hypermedia and web design language, including HDM, HDM-lite, RMM, OOHDM, and Araneus. It provides graphical, yet formal, specifications, embodied in a complete design process, which can be assisted by visual design tools for expressing a hypertext as a set of pages made up of linked content units and operations, and for binding such content units and operations to the data they refer to.

The remaining of this paper is structured as follows: Section 2 describes the motivation of this work. Section 3 provides a short overview of WebML. Section 4 introduces the methodology for capturing application-domain specific patterns. In section 5, the methodology is applied for identifying a calendar pattern in an e-learning application. Section 6 concludes the paper and discusses future steps.

2. Motivation

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. In more detail, they support *site and user management* options (accounts, access levels, role assignments, etc.), *course management* (material authoring and placement, definition of correlations among topics, specification of prerequisite topics, recommendations for next topics, etc.), *assignment mechanisms* (creation and announcement of learning tasks, activation of reminders, assignment submission, grading, detailed teacher feedback, etc.), *community tools* (chat rooms, forums, FAQ sections, discussion boards, etc.), as well as *collaboration features* (global and personal calendars, tasks and ToDo's, libraries of shared resources, group-based work spaces, etc.).

Designing and implementing such applications is a process that involves many experts coming from both the IT and the pedagogical/social studies domain. Experience in big software houses 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.

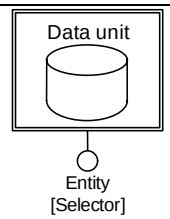
3. WebML, a quick view

WebML 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 [9]. The first step of designing in WebML is to specify the *data schema* of the Web application, in order to express the organization of contents using E-R primitives. The next step is *Hypertext Design,* which produces schemes expressing the composition of content and the invocation of operations within pages, as well as the definition of links between pages.

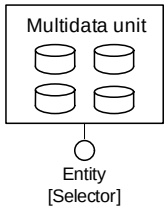
The overall structure of the hypertext is defined in terms of *site views*, *areas*, *pages* and *content units*. A *site view* is a hypertext, designed to address a specific set of requirements. It is composed of *areas*, which are the main sections of the hypertext, and comprise recursively other sub-areas or pages. *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 published within pages. More particularly, as described in Table 1, content units denote alternative ways for displaying one or more entity instances.

Unit specification requires the definition of 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).

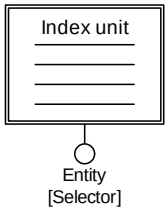
Within site views, links interconnect content units and pages in a variety of configurations yielding to composite navigation mechanisms. Besides representing user navigation, links between units also specify the transferring of some information (called *context*) that the destination unit uses for selecting the data instances to be displayed [10].



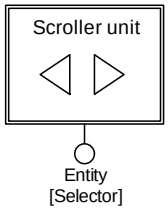
Data unit: It displays a set of attributes for a single entity instance.



Multidata unit: It displays a set of instances for a given entity.



Index unit: It displays a list of properties, also called *descriptive keys*, of a given set of entity instances.



Scroller unit: It represents a scrolling mechanism, based on a block factor, for the elements in a set of instances.

Table 1. Some basicWebML content units (visual notation and description) . The complete set of units can be found in [CFBBCM02].

4. Capturing Application-Domain Specific Patterns

Design patterns as tools that describe a piece of design experience and/or expert advice and make it reusable, were introduced by the architect C. Alexander, in the context of architecture and urban planning [11]: *"… 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 many other domains. In the field of software engineering, design patterns are increasingly used to capture expertise on object-oriented programming [12]. More recently, design patterns have been introduced in the Web modeling field for describing the navigation and structure of a Web application [13, 14, 15, 16].

In WebML, a primitive set of design patterns has been identified that includes commonly applied and consistent solutions that are applicable to any Web application. Patterns have been discovered for data design, by identifying typical roles of information objects within the data schema (i.e., core concepts, interconnection concepts, access facilitators), and typical data sub-schemas constructed around such roles (i.e., core sub-schema, interconnection sub-schema, access sub-schema) [17]. Patterns have also been defined for hypertext design, by identifying unit compositions representing typical hypertext navigation chains [9] and content publishing (cascaded index, filtered index, filtered scrolled index, guided tour, indexed guided tour, object viewpoint, nested data, hierarchical index with alternative sub-pages). Moreover, WebML also introduces patterns for content management operations (object creation-deletion-modification, relationship creation-deletion, create-connect pattern, cascaded delete).

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 describe instead how the context generated by the core pattern composition is passed to successive compositions in the hypertext [18].

In our approach, we extend the primitive set of design patterns identified within WebML, focusing on specific application domains with the objective to capture such patterns in the process of modelling a Web application.

Application-Domain specific patterns are constructs (of units, operations, pages, areas and sub-areas in the case of WebML) within the conceptual schema of a specific-domain Web application, that when applied to various instances of the schema they can solve in an optimal way a specific problem.

A rather simplified approach to 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 Web. If we take the intersection of all the site views in pairs we compute various sets (S_1, S_2,…, S_N) of application domain specific patterns. That is:

$S_1 = SV_1 \cap SV_2$

$S_2 = SV_2 \cap SV_3$

.....

$S_N = SV_{N-1} \cap SV_N$

This way we can capture a first set of patterns. If we take into account the pattern variants that exist in a conceptual schema, the set obtained so far is small and incomplete.

A more sophisticated methodology to capture application-domain specific patterns follows:

Step 1: We traverse each site view of the conceptual schema of a specific Web Application modeled with WebML and find the standard (predefined in WebML) pattern occurrences. We take into account the various pattern variants and create a new XML definition of the site view substituting where possible the variants with the default pattern in an effort to create a more uniform schema. We can achieve that using XSL [18]. The XSL language [19] allows writing pattern-matching rules that can be applied over 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 pattern variants from the XML specification of the WebML conceptual schema, thus generating a new XML document specifying all the retrieved occurrences, and on document further transformation can be applied.

Step 2: We traverse each new site view and try to find common patterns and their variants within the site view that do not belong to the already predefined WebML set of patterns. This way, we extract a first set of application-domain specific patterns per site view. Based on the various pattern variants that we have found we create a new XML definition of the site view substituting where possible the variants with the default pattern in order to create a more canonical schema.

Step 3: We traverse each area, sub-area and page of all the site views of the WebML conceptual schema and extract new patterns and their variants. We also substitute the variants here where possible.

Step 4: We repeat steps 2 and 3 in order to capture larger patterns taking into account the pattern variants identified in each iteration, until no further pattern is found.

Step 5: We cluster the various patterns according to the core, access, interconnection and personalization subschemas in order to make a first categorization of the patterns.

At this point the designer has a first library of application-domain specific patterns 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.

The same methodology can be applied to conceptual models of different Web applications in the same application domain, so as to build a larger set of application-domain specific patterns and to improve their consistency. 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 [18].

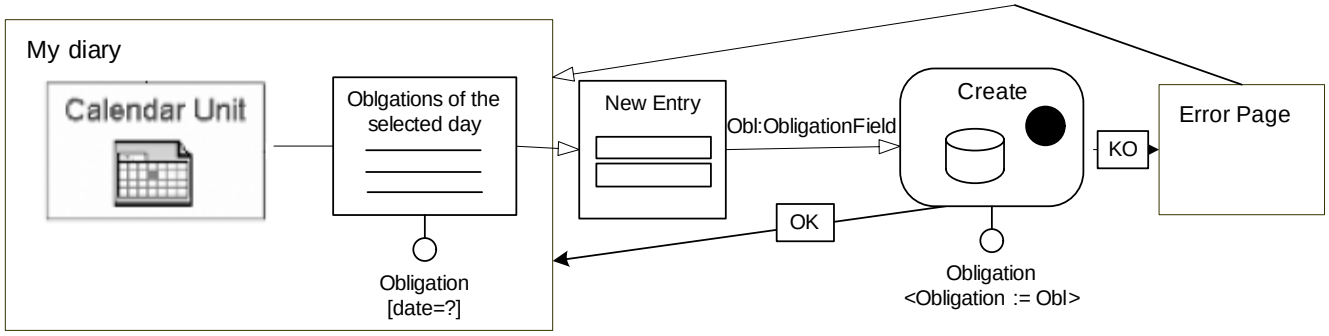


Figure 1. An instance of the student's site view modeling their diary

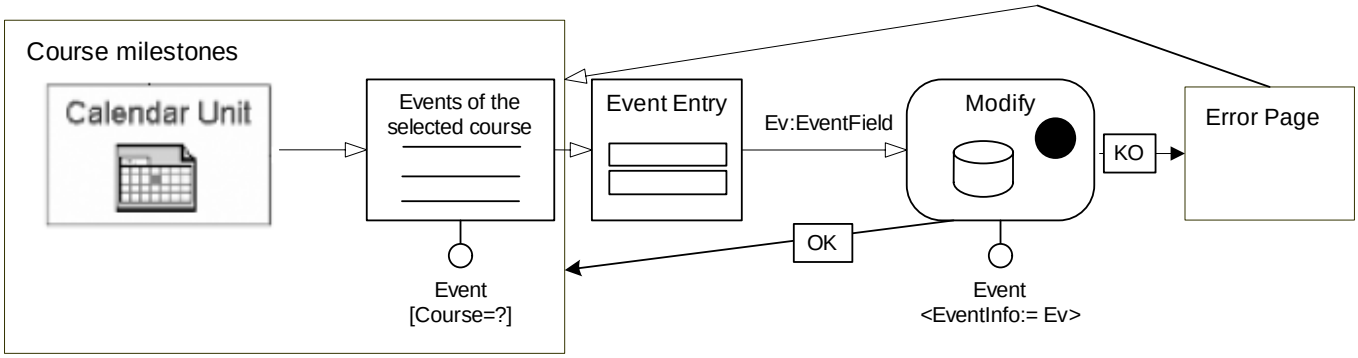


Figure 2. An instance of the teacher's site view modeling course milestones

5. Applying the Proposed Methodology: An Example

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 this example we demonstrate a calendar mechanism handled in different ways by the various system users. Calendar usage depends on the group the user belongs to and the site view it is related with. Figures 1 and 2 show an instance of a professor's and a student's site view respectively. The calendar unit used in both site views is a plug-in unit linked contextually to an index unit. Conceptually, a plug-in unit is a reusable component, characterized by a set of (required or optional) type parameters, and by a set of output parameters [9]. The input parameters can be fed to the unit by input links, and the output parameters can be associated with output links of the unit to be used in the selector of other units.

In the student's site view the calendar outputs a value of type date, corresponding to the day selected by the user from the calendar. This value is associated as a parameter with the output link of the calendar unit, and used in the selector of the index unit, to produce the list of obligations of the date selected by the student. The entry unit is used to supply values to a create unit. The create operation is triggered by the navigation of the outgoing link of the entry unit, which is rendered as the submit button of the form, and creates an Obligation object. The create unit has two output links. The OK link points to the "My Diary" page and is associated with the default link parameter. The KO link points back to the Error Page where a "Creation failed" message is displayed and the user returns to "My Diary" page.

In the professor's site view the calendar outputs a value of type date, corresponding to selected course events. This value is associated as a parameter with the output link of the calendar unit, and used by the selector of the index unit, to produce the list of events of the course selected by the professor. An entry unit is used to supply values to a modify unit and allow users to modify an event. The modify unit is activated by a second link, exiting the entry unit; this link has an explicit parameter (called Ev), which holds the value of the input field of the entry unit, used in the assignment <EventInfo:=Ev> of the operation. The OK link leads to the "Course milestones" page and the KO link points to the Error Page where a message "Modification failed" is displayed and the user returns to the "Course milestones" page.

Observing more carefully both site views one easily discerns the existence of an application-domain specific pattern. First we identify a Calendar pattern which consists of the calendar unit and an index unit referring to student's obligations for a selected date. In the student's site view this pattern is linked to a Create pattern. The Create pattern in WebML is built around the Create operation, which allows adding new data in entity instances, through a data entry unit. The core specification of the pattern consists of a data entry unit with a Create operation unit. Two sets of variants may be defined for the Create pattern, the starting variants, referring to the modality in which the pattern begins, and the termination variants, referring to pattern termination. In this case we have a Start-with-index-unit variant and Same Page termination variant. In the professor's site view the Calendar pattern is connected to a Modify pattern which in WebML is built around the Modify operation and allows updating the content of an entity instance with new data by means of a data entry unit. The core specification of the pattern consists of the composition of a data entry unit, providing a form for inputting new data for the instance attributes to be modified by a Modify operation unit. We have again a Start-with-index-unit variant and a Same Page termination variant.

Both site views depict two variants of the same complex pattern which consists of two individual patterns as described above and can be reused in many cases to solve different problems in the same application-domain. We have a new pattern which consists of the Calendar pattern that we have identified and a pattern that belongs to the primitive set of patterns supported by WebML and can differ depending on the site view.

6. Conclusions and Future Work

This paper introduced the construct of application-domain specific patterns and presented a methodology for their identification within the conceptual schema of an application designed using WebML. These patterns extend the primitive set of design patterns supported by WebML, and can form a useful tool for software architects in future application development, as well as for the effective redesign of various applications within a specific domain.

To this point, the methodology developed is semiautomatic, since the architect has to make the final decision on the usefulness and the variants of an identified design pattern. In the future, we plan to extend the methodology to fully support automatic procedures for pattern identification based on their semantic context as represented in an application's structural model.

References:

[1] J. Greenfield, The Case for Software Factories, *Journal Articles on MSDN Library*, July 2004.

[2] F Brooks, *The Mythical Man-Month* (Addison-Wesley, 1995)

[3] B Boehm, *Software Engineering Economics* (Prentice Hall PTR, 1981)

[4] D Roberts, & R. Johnson, Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks, *Proc. of Pattern Languages of Programs*, Allerton Park, Illinois, September 1996.

[5] S. Montero, P. Diaz, & I. Aedo, Requirements for Hypermedia Development Methods: A Survey of Outstanding Methods, *Proc. of the 14th International Conference on Advanced Information Systems Engineering*, 2002, 747 – 751

[6] P. Atzeni, G. Mecca, and P. Merialdo, Design and Maintenance of Data-Intensive Web Sites, *Proc. EDBT 1998*, 436-450.

[7] G. Rossi, D. Schwabe, F. Lyardet, Web Application Models are More than Conceptual Models, in eds, *Proc. Int. Workshop on the World Wide Web and Conceptual Modeling*, Paris, Oct. 1999, pp.239-252.

[8] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide* (The Addison-Wesley Object Technology Series, 1998).

[9] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, & M. Matera, *Designing Data-Intensive Web Applications* (Morgan Kauffmann, 2002).

[10] S. Ceri, P. Dolog, M. Matera, & W. Nejdl, Model-Driven Design of Web Applications with Client-Side Adaptation, *Proc. of ICWE'04*, Munich, Germany, LNCS 3140, Springer Verlag, 2004.

[11] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, & S. Angel, *A Pattern Language* (Oxford University Press, New York, 1977).

[12] E. Gamma, R. Helm, R. Johnson, & J. Vlissedes, *Design Patterns - Elements of Reusable Object Oriented Software* (Addison Wesley, 1995).

[13] D. Schwabe, A. Garrido, & G. Rossi, Design Reuse in Hypermedia Applications Development, *Proc. of ACM Hypertext '97*, Southampton, UK, 1997, 57-66.

[14] M. Bernstein, Patterns of Hypertext, *Proc. of HyperText'98*, Pittsburgh, PA, 1998.

[15] M. Nanard, J. Nanard, & P. Kahn, 1998. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates, *Proc. of Hypertext'98*, Pittsburgh, PA, 1998, 11-20.

[16] F. Garzotto, P. Paolini, D. Bolchini, & S. Valenti, Modeling-by-Patterns of Web Applications, *In Proceeding of the ER'99 Workshop "World Wide Web and Conceptual Modeling"*, Paris, France, 1999, 293-306.

[17] S. Ceri, P. Fraternali, & M. Matera, M, Conceptual Modeling of Data-Intensive Web Applications, *IEEE Internet Computing, 6*(4), 2004, 20-30.

[18] P. Fraternali, M. Matera, & A. Maurino, WQA: an XSL Framework for Analyzing the Quality of Web Applications, *Proc. Of IWWOST'02*, Malaga, Spain, 2002.

[19] *XSL. Extensible Style sheet Language* (W3C Recommendation, http://w3.org/TR/XSL/, October 2001).