Turk J Elec Engin, VOL.12, NO.2 2004, © TÜBİTAK

A Pattern Based Approach to Web Design Formalization

Ahmet SIKICI, N. Yasemin TOPALOĞLU
Ege University, Department of Computer Engineering, 35100, Bornova, İzmir-TURKEY
e-mail: ahmets@staff.ege.edu.tr, yasemin@bornova.ege.edu.tr

Abstract

World Wide Web is a global information network that affects many fields of our lives. The distributed, interlinked, visual and heterogeneous structure of the web makes it an irregular development environment. In this paper we propose a pattern based approach for increasing the effectiveness of development for the Web environment. This approach is based on the representation of the core meaning of each design as a set of patterns and requires the formulation of abstract solutions in a mathematical precision. This way not only reusable design experience will be codified unambiguously, but a smooth transition between the design and implementation phases will be realized.

1. Introduction

The unique properties of the Web as a knowledge medium and an application domain bring about several independent design issues. As well as application logic, Web design emphasizes content, navigation, presentation and visuality. The heterogeneity of the design concerns has led to multiphase design frameworks that possess integration problems. To achieve wholeness in the design, a *base paradigm*, a reference model that can fulfill all the design concerns equally well is needed. Such a *base paradigm* can be the key to the organization of all the development tasks around purely Web-related concerns.

Experience suggests that using mechanically processable formal models in building and integrating tools can yield systems that increase automation and decrease inconsistency and thus produce faster, cheaper, and more reliably [1]. The lack of a formal model and notation for Web design is an obstacle in the implementation of such a discipline. However it is our observation that the ideas that originate from the pattern paradigm [2] are in harmony with the requirements of the Web design, suggesting that the pattern concept can be a good candidate to be a *base paradigm*. In this paper we present an approach that relies on a formalized interpretation of patterns as primary design components with the help of a formal pattern model [3][4]. To illustrate this approach we will introduce Web Design Logic Definition Language (WDLDL) which is a knowledge-based descriptive language that can capture the reusable design semantics inside formal patterns.

The organization of the paper is as follows. The difficulties of designing for the Web will be discussed in the second section of the paper. In the third section the formal approach brought by WDLDL will be introduced and it will be exemplified in the forth section by defining a pattern with it. The fifth section contains the conclusions.

2. The Strategy of Approaching Web Design

The difficulty of developing and maintaining large scale Web systems has led to the introduction of a new class of development tools called the Web Modeling Languages (WML). Some popular WMLs are: OOHDM [5], WebML [6] and Conallen [7]. WMLs employ object oriented design, XML and some specialized models in different proportions. WML approach can be summarized as the installation of the object oriented methods to an environment where they may not look very natural. However a solution is possible with the help of some add-ons for the Web-specific design concerns, of course if additional labor is affordable. The primary issues of Web design are considered to be conceptual design, content, navigation, user interface and transactions. Content, navigation and most of conceptual modeling tasks by nature are not object oriented. The common WML approach is the arrangement of these concerns as phases or layers of design and handling each in its own logic, tools and diagrams. The critical factor for the success of a WML happens to be integration. The most difficult integration task is probably gluing hypermedia structure with the software application logic [8]. Although there has been many useful solutions proposed for this problem, the current state of the Web development process is not as smooth as the state of the art implementations of the software development process. As [9] states it: *The existing tools for building and deploying complex Web sites are inadequate for dealing with the software production processes that involve connecting with underlying logic in a uniform and systematic way.*

The WDLDL approach to the Web development problem aims to provide the means for the unification of all the efforts within a knowledge-based context. First of all WDLDL has been deliberately designed in order to be a solo medium for conceptual, navigational and structural aspects of any piece of design. secondly WDLDL patterns are pluggable to one another, in order to form more complex patterns. Thirdly, the amount of flexibility brought by the WDLDL syntax is in a very productive harmony with the Web culture.

Some unique properties of the Web bring certain technical issues that require special care and can lead to a revision of some traditional design habits. The scattered, interconnected and document-based nature of the Web artifacts does not conform to some cornerstones of the software development culture like *modularity, type checking* and *encapsulation*.

Modularity of Web systems is threatened by the high semantic coupling between the building blocks. The interconnectedness of the Web brings about the fact that the identity and meaning of an entity is highly dependent on the context in which it appears. Web works become enmeshed in a context that reflects their meaning, use and construction [10]. In the absence of such context knowledge, a Web design should not be seen as a finished product.

In most of the artifacts that constitute the Web it is difficult to find a rigid structure on which a type checking system can operate. Web has taken its basic properties from the physical shape of the Internet and this implies its being global, distributed dynamic, decentralized and non-hierarchical. As a result, community behavior, multiple roles, distributed and overlapping forms are too common to leave any space for reliable structures.

Encapsulation is an effective state of the art tool for coping with complexity. However traditional encapsulation approach is insufficient for the development of Web artifacts. The source of the problem is that, containment is a transitive relation and a useful validation of the type of an artifact requires looking inside the components that form that artifact. As an example let's consider the validation of a type like "illustrated page". The search that should be arranged in order to know if a Web page contains a picture or not, should penetrate into the nested frames and tables and check the whole containment tree. Thus rigid encapsulation is not practical for the Web.

3. A Formal Approach to Web Design

The task of formalizing Web designs requires a direct representation of the design knowledge with a deterministic and precise medium. For the unification of the concerns this medium should be effective in all the design phases and detail levels. The required level of precision should provide mathematical methods to decide if an artifact conforms to a given design.

We have solidified our approach to Web application design around a pattern based formal design language called Web Design Logic Definition Language (WDLDL). As described in [2], in the literature there is quite a big debate on what a pattern really is and whether it can be formalized. Such discussion is out of the scope of this work. We choose to stick to the modest point of view that patterns are packaged, clearly defined, abstract and reusable design fragments and within the context of this definition they can be formalized.

As patterns focus on the relations between the components instead of the components themselves [2], they are in harmony with the Web's decentralized, overlapping and interconnected structure. With their abstraction capability and ability to reflect the flexibility of the semantic networks [11] WDLDL patterns are suitable data structures to carry design knowledge. As well as leaning on our former work on design formalization [3][4] this work has also been influenced by the specialties of the Web. The concepts *modularity, type checking* and *encapsulation* had to be revised according to the functional requirements of the Web environment. In the following subsections, the properties of WDLDL that challenge the problems mentioned in section 2 will be introduced.

3.1. The basic WDLDL approach

WDLDL aims to express a wide spectrum of reusable Web-related design knowledge in an unambiguous form. For conforming to the Web domain, the concepts of *containment* and *connection* are emphasized but it is also able to express relations and constraints. The main characterization of WDLDL is its flexibility that enables the designer to write patterns at arbitrary levels of abstraction. Each design, together with specifying the data, relations and the constraints that distinguishes it, typically contains undefined areas that are customizable in order to reach a number of implementations.

WDLDL is a schematical language that is based on a few basic rules. The most important element in the schema is an ellipse. The ellipse represents the term that the schema aims to define within the schema. Thus the inside of the ellipse represents the content of the term defined, while the outside of it represents its context. Content and context unify in one definition as two zones of a semantic net.

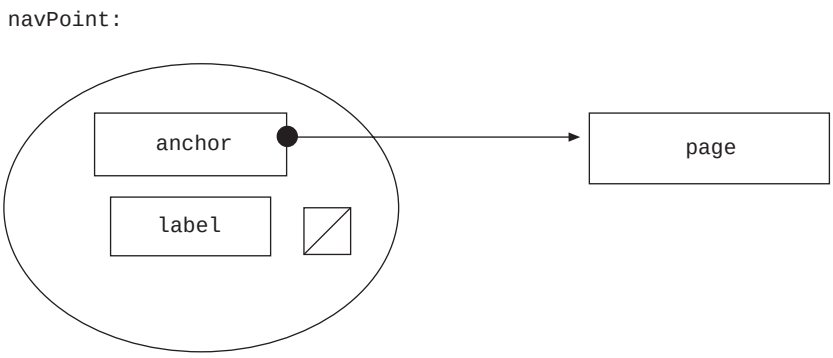

Figure 1. WDLDL schema example.

In Figure 1 WDLDL syntax has been shown over a plain example. The example defines a basic navigation link and this pattern has been named as `navPoint`. The content of the `navPoint` includes an anchor and a label. A label is a piece of text that one can see on the screen and click on it with the mouse. The anchor contains the Web address of the page that is reached by clicking on the label. The slashed rectangle is a *terminator node*. The presence of the terminator node implies that there can be no more nodes in that zone (content). If one adds an extra node into the ellipse and violates the terminator node, then the definition is no more a `navPoint` and this can be detected mechanically. In the context zone of the schema the Web page targeted by the navigation link is shown. An arrow that represents a navigational link indicates that the `anchor` points to this page.

The code below, is the representation of the navPoint pattern in the verbal version of WDLDL. This representation is structurally isomorphic and semantically equivalent to the graphical version. The major syntactical structure of *verbal-WDLDL* is given in Appendix.

```
Pattern navPoint
{ nodes
  { :Self
      { anchor:A;
        label;
        Ends;
      }
    page:P;
  }
  links A navigates P;
}
```

A verbal-WDLDL definition is composed of a variable declaration section (var), a node definition section (nodes) and a link definition section (links). Variable declaration section deals with the creation of numerical variables, which are usually related to multiplicity constraints. In this example we don't need that section since there is no such constraint in this pattern that requires one. Node definition section creates nodes and determines their attributes and the nesting relations between them. The links section defines the links between the nodes, together with link attributes. The *nodes* section always contains the single node called Self that represents the ellipse of the graphical notation, thus the position of the pattern's self with respect to its context.

3.2. Conceptual modularity

Although the Web environment depresses modularity in the architectural sense, it is possible to achieve another type of modularity with the help of patterns. However the term modularity can be misleading when the subject is patterns because it is normal for the patterns to overlap while implementing a solution. In this case the division of labor takes place on the conceptual level only and in order to identify the modules one has to rely on precise definitions of the patterns, instead of physical borders.

WDLDL's approach to the coupling problem is to define the artifacts together with the context that they require. A WDLDL definition is like a big x-ray picture that shows the position of an artifact within a context as well as its inner structure. Every definition contains both the content and context that its subject requires and possesses all the required connections between its elements.

3.3. Irregular type-checking

WDLDL contains mechanisms that aim to deal with the structural irregularities of the Web. Although it is difficult to find stable structures on Web artifacts, with a language that is sufficiently abstract, it is possible to codify the invariables of a dynamic system. When working on an abstract layer, similarities may be discovered between seemingly diverse systems.

WDLDL patterns are defined as a combination of variables and constraints that can be nested within one another. This way a great description domain can be spanned. The skeleton of WDLDL is a dual system of set and network patterns. While the *graph logic* is dealing with the relations between the entities, the *set logic* makes statements about the contents of the entities. The set and the graph concepts are interrelated and one can pass from one to the other within a definition. *All the links of a node*, constitutes a set while any set can be represented with a graph that contains a central node that represents the set that is connected to its elements with a bunch of containment relations.

Another interesting aspect of the set logic is the presence of terminator elements. A terminator element indicates the absence of a special type of element (node or link) in any instance of the pattern. Terminator elements can work on any type of node or link (terminator links and terminator nodes). Terminators are important for two reasons: First of all, they can prevent unexpected evolution of the designs into random forms. Secondly, with them it is possible to make *negative definitions* like a *menuless page*.

It is also possible to use logical operators in WDLDL definitions. With the XOR operator patterns with diverse instances can be defined. With the AND operator different views of a structure can be defined, each as a separate definition.

With such flexibility it is possible to make general but precise definitions about the extendible sides of a design. When the logic of a dynamically changing structure is defined as a pattern, it is possible to determine where it can shrink and where it can expand without violating the base design.

3.4. Encapsulation control

In WDLDL, the containment relation does not tell whether the containee is nested just inside the container or inside a series of nested containers. Being just under the topmost container is a special case which must be stated explicitly. While defining, it is possible to put constraints on the order or number of the layers of nesting. As the nesting chains are sequential, they are a part of the graph logic. By playing with the depth of nesting and the properties of the intermediate containers we can produce *nesting patterns*.

mainSection:

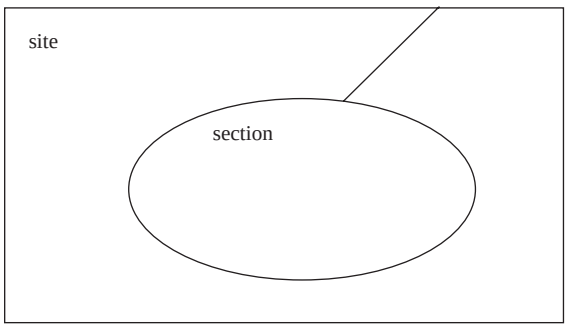

Figure 2. Representation of immediate nesting.

In Figure 2, a pattern called `mainSection` has been defined in order to illustrate the case of being

immediately nested. The diagonal line that connects the nodes section and site shows that the section lies just inside the site (so is a main section).

The situation when a contained unit is linked to a unit that is outside the container, is called an encapsulation violation. Encapsulation violations are indicated by the arrows that penetrate through the borders of the nodes. In WDLDL, units can be thought of as membranes that separate their content from their context. All the relations between the content and context pass through these membranes. As there is no order between the links that violate the encapsulation, the case can be handled by set logic it is possible to reach patterns of encapsulation violation.

Although its name reminds of an error condition, an encapsulation violation is a meta-pattern that can be observed at every phase of software development process. For Web applications however, its presence and absence can be a design issue. For example when we state that there is no navigational link from site X to any search engine, it is possible to codify this as: There is no navigational link that starts from within site X passes through the border of X and targets a page which is a part of a search engine site. The target of the constraint is primarily the surface of the site X and certain types of links are forbidden to penetrate through the borders.

X:

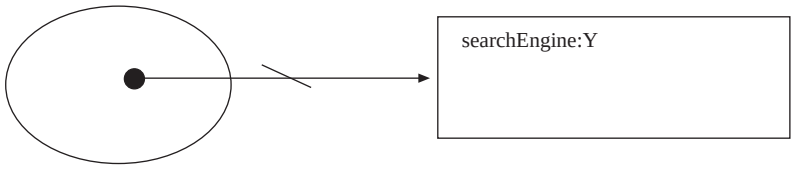

Figure 3. Encapsulation violation.

In Figure 3, it is possible to see the representation of the definition in WDLDL. Navigational links that state within X and end within a `searchEngine` have been banned with a *terminator link*.

The main idea behind WDLDL is that Web artifacts can be classified according to some criteria that are meaningful in terms of Web development. It is important to know for both usage and reuse to know what kind of problem is being addressed by a certain pattern. The forms of expression in WDLDL can code many kinds of statements in order to cover the main web design issues about content, context, relations and the general topological structure.

In contrast with the strict structural approach that is preferred in software development, in a knowledge-based approach, components, page layouts and site topologies are handled not only with a constructionist approach, but with the aim of recognizing patterns on existing implementations, categorizing them under standard names and storing them as reusable components. Thus the components can and should have a semantical load as it is the case with the visual components like top menu, site map and navigation pane; page layouts like main page, non-illustrated page and info-page and topological site frameworks like catalog, presentation, picture album, navigational center, Web ring and virtual city.

When the design definition does not contain expressions that require the interpretation of a designer or a programmer, the design environment is suitable for automation. According to our point of view, such a definition is called a *formal definition*. A formal definition is written in a *formal notation* that employ symbols and relations instead of anecdotes and prescriptions written in natural language and thus usually is machine interpretable. In [1] Luqi and Goguen emphasize the usage of *formal notations* as a requirement for applying formal methods. WDLDL has been designed for an automated development environment, where the tools that operate on the higher layers not only can tell if a given design adheres to a WDLDL pattern,

but they can also produce new patterns by nesting or superposing the existing ones.

4. Site Navigation Menu Pattern

In this section, we want to illustrate the properties of WDLDL with a real Web pattern. The pattern that will be introduced here describes a menu usage that is very common on the Web. This structure that we call Site Navigation Menu provides navigation from any position on a site to one of the main sections of the site.

1. The essential features of the design can be summarized as follows:
2. The menu that is being described by the pattern takes place in every page of the site.
3. The menu possesses a navigational link to every main section of the site.
4. The menu possesses a link to the home page of the site too.
5. The menu does not have any other links.

In Figure 4, the pattern's representation in WDLDL is shown. The most interesting fact about the pattern is that it takes place inside a big node labeled as `site:X`. This node represents the site over which the menu will provide navigation. A separator labeled with the logical operator `AND` separates this node into two zones. When a zone is divided with an AND separator, the zone is defined as having two views which are simultaneously valid. In the example the site X has been viewed from two different perspectives. One of these views interprets the site as a set of pages. The other view organizes it as a hierarchy of sections.

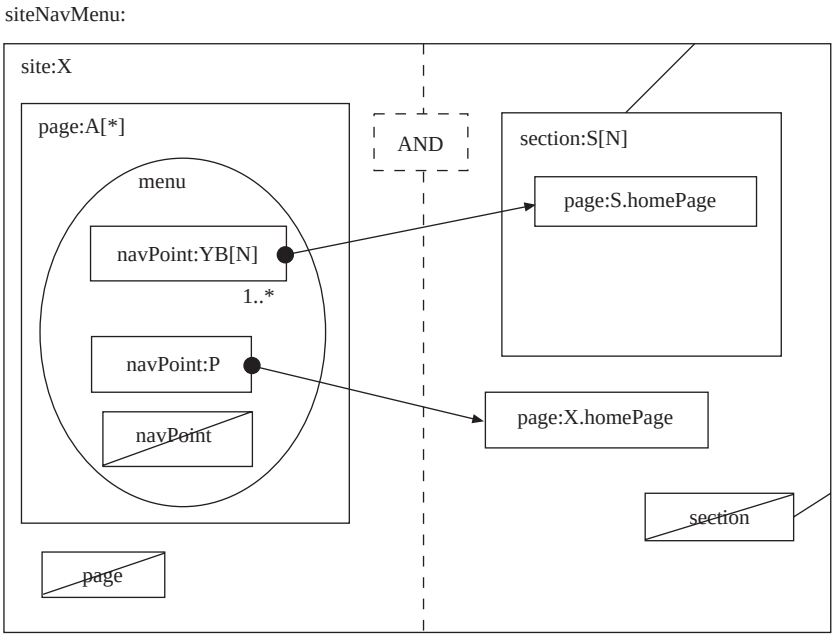

Figure 4. Site navigation menu pattern.

The statement about the set of pages is basic: There should be a menu on each page. `page:A[*]` means multiple pages, and as sharing its scope with a page terminator, it can be interpreted as *every page A*. *The ellipse* inside this node which has been labeled as menu indicates that what is being defined is a

menu that takes place inside every page of the site. The choices of the menu are of type `navPoint` which has been defined in 3.1. N of these choices point to the home pages of the main sections of the site and one of them points to the home page of the site itself with navigational links. By terminating the `navPoint` type in the menu, it has been prevented from having any more choices.

The menu's navigational targets have been defined in the interpretation on the right side. According to this definition, a *main section* is a section that lies immediately inside the site and it has a local home page of its own. By using the same multiplicity variable for both the main sections and the corresponding menu choices these have been paired one to one. By terminating the main sections of the site (with the terminator element on the lower right corner) it has been ensured that all the main sections are covered.

Each home page of each main section is being targeted by a menu choice. Along with these, the site's home page too exists inside the site but outside all the sections. This page has been targeted by a menu choice too.

5. Conclusions

When development environments target the Web or similar hypermedia, a high degree of flexibility that can not be provided with the conceptual background of conventional software development paradigms is often required. Instead of the multi-phase approaches that are based on classical software development paradigms and enhanced with knowledge representation add-ons, it seems to be more suitable to work for a new paradigm that is suitable for the Web.

We think that the Web Design Logic Definition Language is a step towards the construction of such a paradigm. As well as the knowledge representation method that employs nodes and links, the degree of flexibility in the interpretation of concepts like containment, encapsulation and context, are factors that increase the description power of the language. With the increasing descriptiveness it can be possible to refer all aspects of Web development with one conceptual tool and unify the design of the artifact. Working on a uniform medium also gives way to large scale automation that enhances the performance and quality of development.

References

[1] Luqi, Goguen, J. A., "Formal Methods: Promises and Problems", *IEEE Software*, pp. 73-85, January 1997.

[2] Coplien, J.O., Patterns and Art., C++ Report 12(1) pp. 41-43, 2000.

[3] Sıkıcı, A. and Topaloglu, N.Y., "A Knowledge Based Approach to Design with Patterns." in T.Hruska and M.Hashimoto (eds.): Knowledge Based Software Engineering, IOS Press, pp. 133-136, 2000.

[4] Sıkıcı, A. Topaloglu, N.Y. "Towards Software Design Automation with Patterns." *Informatica* Vol. 25, No. 3, pp. 309-317, 2001.

[5] Schawbe, D., Rossi, G., Barbrosa S., "Abstraction, Composition and Lay-out Definition Mechanisms in OOHDM", in Cruz,I.F., Maks, J. and Wittenburg, K. (eds.) Proceedings of the ACM Workshop on effective Abstractions in Multimedia, San Fransisco, CA (1995).

[6] Ceri, S., Freternali, P., Bongio, A., "Web Modelling Language(WebML): A Modelling Language for Designing Web Sites." WWW9 Conference, Amsterdam (2000).

[7] Conallen, J., "Modeling Web Application Architectures with UML." Communications of the ACM 42(10), pp. 63-70, 1999.

[8] Gu, A.,Henderson-Sellers, B.,Lowe, D., "Web Modelling Languages: The Gap Between Requirements And Current Exemplars." The Eighth Australian WWW Conference (2002).

[9] Gomez, J., Cachero, C. and Pastor, O., "Conceptual Modeling of Device-Independent Web Applications." *IEEE Multimedia* April-June pp. 26-39, 2001.

[10] December, J. and Ginsburg, M., HTML and CGI 3.2 Professional Reference Edition. http://docs.rinet.ru:8080/CGI3.2/

[11] Rich, E. and Knight K., Artificial Intelligence. Mc Graw Hill (1991).

Appendix

WDLDL EBNF Definitions

<Pattern_Defn>:- **Pattern** <Pattern_Name>
{ [**var** <VarBlock>]
nodes{<Node>}
links{<Link>}**}**
<VarBlock>:-**{**{<VarDecl>}**}**|<VarDecl>;
<NodeBlock>:-**{**{<Node>}**}**|<Node>;
<LinkBlock>:-**{**{<Link>}**}**|<Link>;
<VarDecl>:-<RangeConst>:<Var>
<Node>:-<Node_Interface><Node_Content>|<Node_Interface>;
<Node_Interface>:- <Type>[**:**<NodeName>][<Multiplicity>] [<Nesting>]
<Node_Content>:-**{**{<Node>}**}**
<Multiplicity>:-[(<RangeConst>|<NumConst>|<Var>)]
<Nesting>:-**Within**[(<Node_Interface>|**Nothing**)]
<Link>:-<NodeName>[<Multiplicity>]<Type>[:<LinkName>] [<Multiplicity>]
<Pattern_Name>:-<LowerCName>
<NodeName>:-**Self**|<Full_Name>
<LinkName>:-<Full_Name>
<Full_Name>:-<Capitalized_Name>{([<Numeric>]|
.<Capitalized_Name>)}
<Numeric>:-<Num_Constant>|<Var>
<Var>:-<Capitalized_Name>
<RangeConst>:-<Number>..<Number>
<NumConst>:-<Number>