Formalization of web design patterns using ontologies

Susana Montero, Paloma Díaz, and Ignacio Aedo

Laboratorio DEI. Dpto. de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad 30. 28911 Leganés, Spain
smontero@inf.uc3m.es, pdp@inf.uc3m.es, aedo@ia.uc3m.es
http://www.dei.inf.uc3m.es

Abstract. Design patterns have been enthusiastically embraced in the software engineering community as well as in the web community since they capture knowledge about how and when to apply a specific solution to a recurring problem in software systems. However, web design involves both cognitive and aesthetic aspects, for that there are several design patterns that describe the same problem but from different points of view and with different vocabulary, so it is more difficult to understand and to reuse that knowledge.
To achieve a common vocabulary and improve reusability we propose to formalize web design patterns by means of ontologies. At the same time the ontology would allow us to express web design patterns in a formal way, that can be understood by a computer. So a design pattern knowledge base can be integrated in a software tool for web application modeling in order to suggest web design patterns that better suit some design requirements. Moreover, changes and alterations needed for applying design patterns to the design can be proposed.
This paper describe the core ontology using DAML+OIL and show how web design patterns can be described by our ontology.

KEYWORDS: web applications, design patterns, ontologies, hypermedia

1 Introduction and motivation

Web applications [1] have been extremely demanded in different areas like e-commerce, tele-education, health, and libraries, both to provide a web interface for existing information systems and to create new applications. Even though most developers skip the conceptual design phase and directly go to the implementation stage using authoring tools like DreamWeaver or NetObjects' Fusion because they allow for setting up a website in a very short period of time, there are tools based on hypermedia design methods that perform a conceptual design [13].

Conceptual modeling allows for describing the general features of an application in an abstract and implementation independent way, producing applications of better quality, usability and maintainability. However, there is little guidance on how to match models to implementations.

[1] We use the term hypermedia application as a synonym for web application since the later is a particular case of hypermedia application

In order to mitigate this gap between conceptual modeling and application implementation design patterns have been enthusiastically embraced both in the software engineering community [6] and later in the web community [14] as a effective mean for reusing good design solutions. A design pattern *captures knowledge of how and when to apply the solution to a recurring problem, besides providing a shared vocabulary for expressing and communicating such a knowledge*. Moreover, the use of design patterns for web applications can increase the quality of their design and decrease the time and cost of design and implementation [7].

Although the spirit of design patterns is to capture design experience in a form which can be used effectively, we find a twofold problem. On the one hand, using patterns is not a trivial process. The designer or developer has to have a high knowledge of the design process and high level of abstraction. On the other hand, analysing web design patterns there is a large number of them published and some of which describe the same problem but from different points of view. Consequently, reuse is hindered by the absence of a common language and a catalog of patterns in such a way that designer could be assisted.

In order to face up these problems we propose to conceptualize the web design pattern knowledge using an ontology, since both design patterns and ontologies have as main motivation *share and reuse knowledge bodies* [16].

The aim of this paper is to introduce an ontology in order to have a formal representation of expert knowledge captured in web design patterns and to facilitate their application in computational form. Not only web design patterns could be described by means of a common vocabulary in order to be shared and reused by the web community, for example by means of repositories, one of the main pending issues in this area [7,9], but incorporated in software tools of some hypermedia design methods to help designers in their practical application.

To conceptualize the domain, we identify two different areas: hypermedia models and design patterns. On the one hand, most hypermedia design methods [13] are based on hypermedia models. They define the components describing any hypermedia and web application. On the other side, design patterns are described by means of a format like the one used in [6].

The rest of the paper is organized as follows: next section presents those meaningful concepts identified from our domain knowledge. Section 3 defines the ontology to specify web design patterns and how a concrete web design pattern can be formalized with the ontology encoded in DAML+OIL. Section 4 presents some related works. Finally, some conclusions and future works are drawn.

2 Domain description: hypermedia and design patterns

In ontological engineering, the knowledge is investigated in terms of its origin and elements from which knowledge is produced [12]. Web design patterns are described by a number of particular terms (e.g. nodes, links, etc.) and have a fixed format (e.g. name, problem, etc.). Thus, in a first approach, we need to combine knowledge from two different areas: *hypermedia models* and *design patterns*. The formers provide a vocabulary of terms and concepts to describe the general features of a hypermedia application in a

conceptual way. The latters describe a specific problem and propose solutions for them using a specific format.

2.1 Hypermedia models

Web applications are based on the hypertext paradigm: associative navigation through a heterogeneous information space. Therefore they are considered as a particular hypermedia application.

Hypermedia models, such as the Dexter [11], HDM [8] or Labyrith [3], propose a number of elements and constructs for describing the structure and behaviour of any hypermedia application. Thus, a basic vocabulary of terms and concepts about the web domain may be defined from them. Taking into account only the elements related to their structure, all hypermedia models include:

- **Node** is an information holder able to contain a number of contents. Examples of nodes are a web page, a frame or a pop-up window.
- **Content**, is a piece of information of any type like text, audio, graphics, animations or programs.
- **Link** is a connection among two or more nodes or contents. A link is defined between two set of anchors.
- **Anchor** is source or target of a link, and determines a reference locus into a node and/or content.

An example of a web application marking their different elements is illustrated in figure 1. Each web page is a node where several text and image contents are placed. Some contents have anchors that set up the links offered to the users.

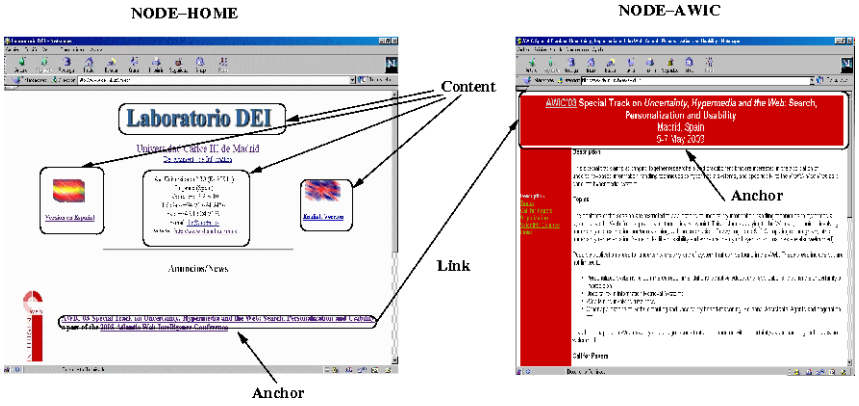

Fig. 1. Example of the elements of a web application

Depending on models we can find more complex elements such as composite nodes or contents and virtual or dangling links to describe hypermedia applications. However, the above elements are crucial to represent the knowledge of the hypermedia domain.

2.2 Design patterns

Design patterns usually are expressed in natural language. Our aim is not to conceptualize the semantic underlying natural language, but to make them understandable from a computational point of view. Thus, we focus on the way they are described. They usually contain certain essential elements that describe the context where problem occurs, the statement of the problem, a description of the relevant forces and how to generate the solution. In our case, we take into account the following parts of a design pattern according to the format specified in [6]:

- **Name** is used to unambiguously identify the pattern itself. For instance, in [7] we find a pattern called *Index Navigation*.
- **Category** is used for classifying the pattern according to several criteria such as its purpose or scope. Web design patterns are usually organized by design activity and the level of abstraction where they are applied. For example, the *Index Navigation* pattern belongs to the *navigational category* because its design activity is concerned with organizing the navigation through the hypermedia application. Moreover, the pattern is described in a detailed way regardless of the implementation.
- **Problem** describes the scenario in which the pattern can be applied. For example, the *Index Navigation* pattern is applied to provide *fast access* to a *group of elements* with the possibility of choosing one of them.
- **Solution** describes how to obtain the desired outcome. For example, in order to apply the *Index Navigation* pattern, *links* should be defined from the entry point to each *node*, and from each node to the entry point.
- **Related-patterns** relates a pattern with others if they address similar or complementary problems or solutions. Thus, another pattern related with the *Index Navigation* pattern is *Hybrid Collection*. This pattern provides an easy-to-use access to a small group of nodes, allowing both to traverse the group of nodes in a sequential way and to access specific one of them.

3 Modeling an ontology for web design patterns

Domain conceptualization has been developed following an iterative and incremental process. Domain knowledge acquisition was achieved from our background in hypermedia area as well as the survey of several hypermedia models and design patterns, as mentioned in section 2. Due to the large number of web design patterns published [9] we could check if acquired concepts allowed to conceptualize the knowledge and thus, to obtain the needed feedback. Finally, the ontology was formalized by a language.

We have chosen DAML+OIL to describe the domain ontology. DAML is a markup language based on XML and is built on top of Resource Description Framework (RDF) and its schema (RDFS) [2] which is compatible with emerging web standards. OIL combines frame-like syntax with the power and flexibility of a description logic (DL) which facilitates reasoning services. The OilEd editor [3] has been used for the definition and description of classes, slots, individuals and axioms of the ontology.

[2] http://www.w3c.org/RDF/

[3] http://oiled.man.ac.uk/index.shtml

3.1 Domain ontology

Our ontology is defined in three different layers. The first layer is represented by pattern components and hypermedia elements. Each of them can be reused in other ontologies because they have been defined in an independent way regardless of hypermedia design patterns. This layer is the basis to represent the second layer, hypermedia design patterns. Concrete instances of this second layer are considered the third layer.

Figure 2 shows the whole class hierarchy for the first and second layer. The knowledge of design patterns is represented by subclasses of the class PatternComponent. The subclasses of the class HypermediaElement represent elements of a web application. Finally, the class HypermediaPattern gathers the format of a design pattern with hypermedia elements that take part in the problem and solution of the pattern. The only new concept is the class Actor which represents the state of elements that take part both in the problem and in the solution of a design pattern. This allows to express design patterns independently of their domain.

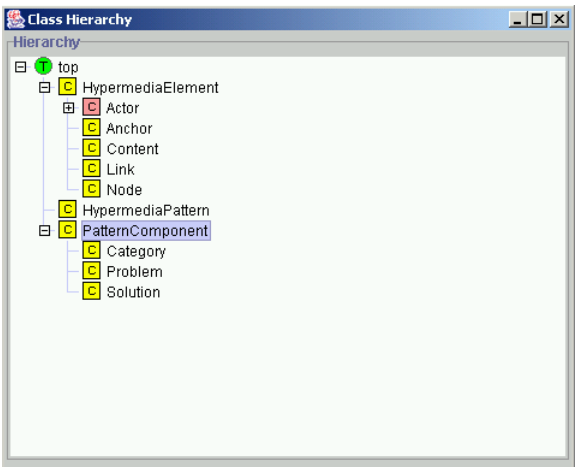

Fig. 2. Hierarchy derived from the ontology

Next, the subclasses of PatternComponent and HypermediaPattern with their slots are briefly described. The subclasses of HypermediaElement are the same concepts mentioned in section 2, so they do not need further explanation.

Regarding design patterns, only three of their components are represented as a class: Category, Problem and Solution. The rest of the elements are slots of the class HypermediaPattern.

The class Category classifies a pattern according to the values of *scope* and *level*. The former represents the design activity where a pattern can be applied such as navigation or presentation. The latter defines the level of abstraction, such as conceptual, detailed or implementation.

class-def Category

subclass-of PatternComponent
slot-constraint *scope*
value-type string
slot-constraint *level*
value-type string

Both the class Problem and the class Solution are described by means of the specific state of Actors that participate in the scenario in which a pattern is applied and its desired outcome respectively.

class-def Problem
subclass-of PatternComponent
slot-constraint *aim*
value-type string
slot-constraint *state*
has-value Actor

class-def Solution
subclass-of PatternComponent
slot-constraint *state*
has-value Actor

The slot *state* can only be filled with a class Actor. This means that these components can be reused to describe design patterns in other areas like object orientation or human computer interaction. For the purposes of this ontology Actor is defined as a HypermediaElement by means of the axiom `equivalent` between Actor and HypermediaElement. Moreover the class Problem has the slot *aim* that describes the use motivation of a pattern by means of keywords. These descriptions can be reinforced with the use of other axioms. For instance, it is not possible to be both a problem and a solution so these concepts are described as disjoint.

Finally, the concept HypermediaPattern is defined as:

class-def HypermediaPattern
slot-constraint *name*
value-type string
slot-constraint *hasCategory*
has-value Category
slot-constraint *hasProblem*
has-value Problem
slot-constraint *hasSolution*
has-value Solution
slot-constraint *relatedTo*
has-value HypermediaPattern

The slots describe that a hypermedia design pattern must have a name, a category, a problem, a solution and related patterns. From the definition of this last concept web design patterns can be described by its individuals.

3.2 Describing a web design pattern

The classes described above belong to the first and second layer of our ontology. In the third layer concrete web design patterns are instanced from those levels. Next we show an example of how to make these instances.

In [7] some web design patterns can be found. One of these is the `Index Navigation` pattern that was already explained in section 2.2. Since our approach is based on the idea of viewing a design pattern as a set of actors that play two different roles: the problem and the solution state, the design pattern must first be projected onto two states.

Bellow it is described as an instance of the class HypermediaPattern with DAML+OIL.

```
<rdf:Description rdf:about="#IndexNavigation">
        <rdf:type>
            <daml:Class  rdf:about="#HypermediaPattern"/>
        </rdf:type>
<ns1:name>
            <xsd:string xsd:value="Index Navigation"/>
        </ns1:name>
        <ns0:hasCategory rdf:resource="#CategoryNC"/>
        <ns0:hasSolution rdf:resource="#solutionIndexNavigation"/>
        <ns0:hasProblem rdf:resource="#problemIndexNavigation"/>
        <ns0:relatedTo rdf:resource="#HybridCollection"/>
    </rdf:Description>
    <rdf:Description rdf:about="#CategoryNC">
        <rdf:type>
            <daml:#Category"/>
        </rdf:type>
        <ns0:level>
            <xsd:string xsd:value="Conceptual"/>
        </ns0:level>
        <ns0:scope>
            <xsd:string xsd:value="Navigation"/>
        </ns0:scope>
    </rdf:Description>
    <rdf:Description rdf:about="#problemIndexNavigation">
        <rdf:type>
            <daml:Class rdf:about="#Problem"/>
        </rdf:type>
        <ns1:aim>
            <xsd:string xsd:value="fast access"/>
        </ns1:aim>
        <ns0:state rdf:resource="#aNode"/>
    </rdf:Description>
    <rdf:Description rdf:about="#solutionIndexNavigation">
        <rdf:type>
            <daml:Class rdf:about="#Solution"/>
```

```
        </rdf:type>
        <ns0:state rdf:resource="#aNode"/>
        <ns0:state rdf:resource="#aNodeIndex"/>
        <ns0:state rdf:resource="#aLink"/>
    </rdf:Description>
</rdf:RDF>
```

In order to make this instance the following slots have been filled as:

- *name* is `Index Navigation`
- *hasCategory* is an instance of the class Category whose slots *level* and *scope* have the values `conceptual` and `navigation` respectively.
- *hasProblem* is an instance of the class Problem. The slot *aim* has the value `fast access` and the slot *state* has an instance of the class Node. That means that we should apply this pattern to provide fast access to a group of nodes.
- *hasSolution* is an instance of the class Solution whose slot *state* has three values. The value `aNode` represents the group of nodes on which this design pattern is applied. The value `aNodeIndex` hold the set of the entry points to `aNode`. Finally, the value `aLink` represents links from the node `aNodeIndex` to each member of `aNode` and vice versa.
- *relatedTo* is a instance of the class HypermediaPattern whose value is `HybridCollection`, that is the other pattern which is related to.

This example represents the *Index Navigation* pattern in a formal way and is expressed in a understandable computational language, DAML+OIL. Therefore it is can be shared with the rest of the web community and integrated in a software development environments for web applications. A knowledge base with web design patterns formalized with this ontology is being built.

4 Related Works

There are two approaches to consider for the formalization of design patterns for their automatic processing. One of them employs formal methods to represent design patterns. For example, Cornils and Hedin [2] model design patterns using reference attributed grammars with syntactic and context-sensitive rules. Eden et al. [4] define LePlus (LanguagE for Patterns Uniform Specification) to represent design pattens as logic formulas which consist of participants (i.e. classes, functions or hierarchies) and relations imposed amongst. Smith and Stotts [15] use an extension of sigma calculus which defines relationships between the elements of object oriented language for expressing design patterns. Although, these approaches provide rigorous reasoning, it is hard to understand the structure and relationship to represent a design pattern and even more to provide design guidelines to the practitioners.

The second approach implements solutions for the automatic processing of design patterns. These solutions are too much dependent of software tools in order to be used directly as a formalization, although some ideas can be taken into account. For example,

Gomes et al. [10] describe a design pattern as a set of participants with their properties that specify when the design pattern is applicable and an operator which defines how to apply the design pattern. Abin-Amiot et al. [1] define a meta-model to describe the structural and behavioural aspects of the design pattens. It is made up of a set of meta-entities which represent participants and each entity contains a collection of elements, representing relationships among entities. Florijn et al. [5] represent design patterns as a collection of fragments where each of them is a design element (i.e class, method, pattern, etc.) with particular information and roles that contain references to other fragments.

All of these approaches, including the our one, are based on the representation of design patterns as a set participants and their relationships among them. However our approach has two distinguishing points. On the one hand, the specification of design patterns is made with the same elements as the design methods to describe applications. Moreover, design patterns are applied during design process, so they are independent of programming languages. On the other hand, the use of ontologies for the formalization of design patterns provide us not only the same aspects as the formal approaches such as elements, relations, rules and reasoning (e.g. description logic) but enabling communication and knowledge reuse between different systems and/or people.

5 Conclusions and future works

This paper deals with a problem of knowledge sharing and reuse, the web design pattern knowledge. It presents a domain ontology that describes the concepts and properties of an application area from the elements that are required for expressing web design patterns in a formal way.

Our ontology has been defined from the primitives of several well-known hypermedia models as well as from the generic format of a design pattern, both of them in an independent way. This approach allows designers to exchange and share knowledge of web design patterns among different hypermedia methods as well as to integrate web design patterns into the design process.

Moreover, parts of the ontology can be reused and extended. On the one hand, the knowledge of design patterns can be used to build the others in other domains. On the other hand, the knowledge domain of hypermedia elements can be used as a hypermedia model itself.

Since our ontology was intended for automatic application purposes we are currently working on describing tasks needed to detect and apply web design patterns that better suit different design views and its later integration into a hypermedia design tool.

Acknowledgements

This work is supported by The Ariadne project funded by "Dirección General de Investigación del Ministerio de Ciencia y Tecnología" (TIC2000-0402)."

References

1. H. Albin-Amiot and Y. Guéhéneuc. Meta-modeling design patterns: Application to pattern detection and code synthesis. In *Proceedings of the ECOOP Workshop on Automating Object-Oriented Software Development Methods*, pages 57–64, June 2001.
2. A. Cornils and G. Hedin. Tool support for design patterns based on reference attribute grammars. In *Proc. of WAGA'00, Ponte de Lima, Portugal*, 2000.
3. P. Díaz, I. Aedo, and F. Panetsos. Labyrinth, an abstract model for hypermedia applications. description of its static components. *Information Systems*, 22(8):447–464, 1997.
4. A. H. Eden, A. Yehudai, and J. Gil. Precise specification and automatic application of design patterns. In *Proc. of International Conference on Automated Software Engineering (ASE '97)*, pages 143–152, Lake Tahoe, CA, USA, 1997.
5. G. Florijn, M. Meijers, and P. van Winsen. Tool support for object-oriented patterns. In *Proceedings of ECOOP'97*, Finland, 1997.
6. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
7. F. Garzotto, P. Paolini, D. Bolchini, and S. Valenti. Modeling-by-Patterns of web applications. In *Advances in Conceptual Modeling: ER '99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling*, pages 293–306, Paris, France, 1999.
8. F. Garzotto, P. Paolini, and D. Schwbe. HDM- a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, 1993.
9. D. German and D. Cowan. Towards a unified catalog of hypermedia design patterns. In *Proceedings of 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, 2000.
10. P. Gomes, F. Pereira, P. Paiva, N. Seco, P. Carreiro, J.L. Ferreira, and C. Bento. Using CBR for automation of software design pattern. In *Proceedings of the European Conference Case-Based Reasoning (ECCBR'02)*, pages 534–548, 2002.
11. F. G. Halasz and M. Schwartz. The dexter hypertext reference model. In *Proc. of World Conference of Hypertext*, pages 95–133, 1990.
12. R. Mizoguchi, T. Sano, and Y. Kitamura. An ontology-based human friendly message generation in a multiagent human media system for oil refinery plant operation. In *Proc. of the IEEE SMC99, IEEE Systems, Man and Cybernetics Society*, 1999.
13. S. Montero, P. Díaz, and I. Aedo. Requirements for hypermedia development methods: A survey of outstanding method. In *Proc. of Advanced Information Systems Engineering, 14th International Conference, CAiSE*, pages 747–751, 2002.
14. G. Rossi, A. Garrido, and S. Carvalho. *Design Patterns for Object-Oriented Hypermedia Applications. Pattern Languages of Programs II*. Addison-Wesley, 1996.
15. J. Smith and D. Stotts. Elemental design patterns: A link between architecture and object semantics. Technical Report Technical Report TR02-011, Univ. of North Carolina at Chapel Hill, March 2002.
16. R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data Knowledge Engineering*, 25(1-2):161–197, 1998.