Improving Design Pattern Adoption with an Ontology-Based Repository

Luka Pavlič, Marjan Heričko, Vili Podgorelec and Ivan Rozman
Institute of Informatics, University of Maribor, FERI
Smetanova ulica 17, SI-2000 Maribor
Slovenia
E-mail: luka.pavlic@uni-mb.si

Keywords: design patterns, semantic web, ontologies, design pattern repository

Received: February 7, 2008

In software engineering, an efficient approach towards reuse has become a crucial success factor. Conceptual simple high level approaches to reuse are the most appropriate for performing it in a useful manner. Design patterns are reliable and an effective high level approach that enables developers to produce high quality software in less time. Unfortunately, the rapidly growing number of design patterns has not yet been adequately supported by efficient search and management tools, making the patterns uninviting for a large part of the software development community. In this way, the issue of managing and selecting design patters in a straight-forward way has become the main challenge.
In this paper, we propose a possible solution for the improvement of design pattern adoption and present a platform that should give design patterns some new and long-overdue momentum. Using our proposed technique for formal design pattern specifications, we have developed an experimental prototype of a new design pattern repository based on semantic web technologies. A new Ontology-Based Design Pattern Repository (OBDPR) has been developed that can also be used as a platform for introducing advanced services. Some fundamental services – searching, design pattern proposing, verification and training services – have already been developed and many others are proposed. Based on the conducted experiments, it is our strong belief that the proposed approach together with the platform's potential -- can significantly contribute to the improvement of design pattern adoption.

Povzetek: Za ponovno uporabo programske opreme je razvita nova metoda z uporabo ontologije in repozitorija.

1 Introduction

Software patterns offer the possibility of achieving reuse in the area of software engineering. In software engineering, several levels of reuse are established. The reuse of concrete software elements such as functions, classes and components have already been well established and practiced on a daily basis. However, if we observe reuse at higher levels of abstraction, i.e. software patterns, reuse is still not practiced on a daily basis.

A pattern is a form of knowledge that is used to capture a recurring successful practice [10]. Basically, a pattern is an idea that has been used in a practical context and probably will be useful in others [24]. As such, software patterns delineate the best practices for solving recurring software design problems and are a proven way of building high quality software [5]. They capture knowledge that experienced developers understand implicitly and facilitate training and knowledge transfer to new developers [17]. One survey [13] has indicated a low adoption of design patterns among practitioners – respondents estimated that no more than half of the developers and architects in their organization knew of, or used, design patterns. Therefore, bridging the gap between the pattern expert communities and the typical pattern user is critical for achieving the full benefits of design patterns [4].

The goal of this paper is to improve design-pattern adoption within the context of a typical pattern use case: a user has to select an appropriate design pattern, understand it and its consequences in detail, and also use it efficiently. To achieve this lofty goal, we will firstly explain how to introduce an appropriate design pattern presentation technique. In order to do this, we will consider several formal design pattern presentation techniques, as presented in this paper. Before we dig deeper, let us clarify the boundaries of our research. While speaking of software patterns we are addressing a whole family of patterns. There are a lot of software pattern types that have been recognized so far. Some authors have proposed a general software-pattern taxonomy [24]:

- Patterns in software analysis, which are the most abstract software patterns;
- Architectural patterns;
- Design patterns;
- Interaction patterns and
- Patterns in software implementation (also known as idioms).

We can, however, split software pattern categories further (e. g. database design patterns, ontology design patterns, communication design patterns etc). It is almost impossible to provide a formal specification for software patterns in general, since every software pattern family addresses a different set of aspects. Software pattern formal specification is, as will be discussed later, necessary in order to provide advanced automatic services. We have limited our research to object-oriented design patterns, since they are the most used and well-known software patterns [27]. This limitation is based on the observation that catalogues with only a few design patterns have clearly been shown to be problematic [5]. However, it is our belief that the approach we propose could eventually also be used with other software patterns, especially when considering the requirement that one could also formalize aspects specific to covering expert knowledge.

In our work, we do not try to formalize all possible aspects. Our goal is to provide a human and machine understandable foundation, primarily to support the design pattern selection process. We do not therefore cope with formalizing the pattern implementation or verification, for instance. Instead, we formalize object-oriented concepts, relationships and expert knowledge on design patterns.

Design patterns improve software design productivity and quality for the following reasons [22]:

- They capture previous design experiences, and make it available to other designers - designers do not need to discover solutions for every problem from scratch.
- They form a more flexible foundation for reuse, as they can be reused in many ways.
- They can be used as a tool for communication among software designers. In fact, this was the original idea of introducing design patterns.

Although design patterns could help significantly in producing high-quality software, developers are continuing to experience more and more difficulties e. g. when finding patterns to match their design problems. It seems that managing and searching facilities are not catching the growing number of design patterns. In this paper, we will also address this challenge. Since we have had many positive experiences in initiating developers to use design patterns, we have also decided to develop an integral web-based platform, primarily to help select design patterns. The platform (Ontology-Based Design Pattern Repository – OBDPR) presented in this paper is a platform and, as such, provides the basis for automatic and intelligent services to be built on top of formally presented design pattern knowledge. We have developed a set of services, also described in this paper, on top of OBDPR:

- design pattern searching service,
- design pattern proposing service, which can also be used as a design pattern suitability verification service,
- training service.

Based on formal design pattern representation, we have introduced capabilities known from the artificial intelligence area into OBDPR and its services. In this paper, we will present the initial experiment performed, which should demonstrate the usefulness of our approach. Moreover, we are planning to perform additional rigorous experiments to indicate if and how much our platform helps software engineers, especially inexperienced ones. We assume that positive experiences with students could also be achieved with full-time developers.

The structure of this paper is as follows: In the “Related work” chapter we present known techniques in formalizing design patterns and their possible and concrete applications. Based on this, we present our own method for formal design pattern representation in chapter three. In that chapter, we also discuss why and how to use ontologies while addressing challenges related to selecting, understanding and using design patterns. Chapter four gives a detailed insight into the conceptual and technical background of OBDPR. The platform’s functionality and additional services are also described. The results of introducing the platform are presented in chapter five. The findings of the initial experiment, where users were exposed to solving design problems with and without our tools, are also presented. Chapter six shows some future trends in our research activities. The final chapter summarizes the most important points of this paper and concludes it.

2 Related work

Since 1994, when design patterns were introduced, many different approaches have been used for documentation purposes. In general, there are three main categories for descriptions:

- informal representations,
- semiformal representations based on graphical notations such as UML and
- various formal representations, which also include notations using semantic web technologies.

Design patterns are traditionally represented by informal, loosely structured documents. These documents are in a canonical form, which consists of a series of fields (name, intent, applicability, structure, participants, consequences, implementation etc), defined by informal descriptions. They help developers understand patterns, but there are glaringly obvious issues regarding advanced knowledge management possibilities. We can also find several semiformal representations, most of them are based on UML [6, 8, 3, 12]. These representations are efficient for a basic understanding of patterns since they cover their structural elements. They are strongly supported by tools, which enable developers to include design patterns in their solutions in a straightforward way. They are successful at capturing structures (static image, usually shown with class diagrams) and behavior (dynamic image, usually shown with sequence or collaboration diagrams). They do not provide information and knowledge on high level aspects such as intent, usability and consequences, on the other hand. For enabling sophisticated services on design patterns, e.g. the ones listed in the prior chapter, we need fully formal representations. The main goals of formalizing design patterns that are recognized within the community are [22]:

- Better understanding of patterns and their composition. It helps to know when and how to use patterns properly in order to take full advantage of them.
- Resolving issues regarding relationships between patterns. It is not only relevant which design patterns are used to solve a problem, it is also important in which order they are applied.
- Allow the development of tool support in activities related to patterns.

In general, when talking about tool support, researchers are currently trying to develop a formal representation of design patterns, primarily for [22]:

- searching for patterns in existing solutions,
- automatic code generation,
- formal solution validation.

A typical use case for this would be the following:

- Developers include a design pattern in their solution. The code is generated automatically. Tool support mostly includes the design pattern in UML diagrams. Other languages are also supported in some tools (e. g. DPML, RSL, RBML, LePUS – see [22]).
- After further development the solution is completed.
- Testers can use tools for formal testing based on design patterns. Tools can find semantic errors in a syntactically perfect solution. Improvements can also be proposed. Since this functionality can really demonstrate the tool's ability to do some inference, it is supported by almost all tools based on languages that enable code X-ray and inference (see Table 1). PEC (Pattern Enforcing Compiler) goes even further – it includes design patterns in the solution at compile time in order to avoid some errors.

There have been several attempts at introducing formal representations in the design patterns area. Some of them are based on pure mathematics, such as first-order logic, temporal logic, object-calculus, ρ-calculus and others [22]. On the other hand, some authors [7][8] are trying to formalize design patterns and keep them understandable for humans at the same time. It is the idea, similar to the semantic web (to keep data semantically understandable both to human and machine). Their representation is mostly supported with ontologies. It is used primarily to describe the structure of source code, which is done according to particular design pattern. One of those used in the "Web of Patterns" (WoP) project [1] has addressed the area of describing knowledge on design patterns.

As stated above, the authors [22] propose several tools; some of them are available for production environments. However, one could also imagine other tools supporting activities regarding design patterns. For instance: before we introduce a design pattern to our solution, how to select an appropriate one? If we have an idea of using a design pattern, one can imagine if the selected pattern would do the desired job. One could also speculate if there is any design pattern that is more suitable than the one currently used. Those were just a few ideas about how to use formal design pattern knowledge in applications. We have not found complete and proven solutions to these challenges, even if there are a few tries, based on keywords rather on design pattern knowledge (for instance [25]). To summarize, for supporting those and other activities, the authors are trying to formalize several aspects of design patterns. They can be divided into the following areas [21]:

- pattern structure (classes, methods, relationships etc),
- pattern behavior (e.g. method call sequence),
- pattern implementation,
- context prerequisites for using design patterns,
- verifying design and implementation based on patterns,
- pattern compositions.

It depends on the formal representation goal for which area of pattern will be formalized (see Table 1). For details on several methods see [22]. In Table 1, we summarize the most important techniques available today. We believe it is important if a method has only theoretical foundations or if it actually has direct tool support. In Table 1, we also show the aspects being formalized by a certain method (we limit the summary to a static and dynamic aspect of a design pattern).

Table 1: The most important formal methods for representing design patterns

Name	**Tool support?**	**Static or dynamic aspect**	**Purpose**
DPML	☑	both	MDA (Model-Driven Architecture)
RSL	☑	both	MDA, code verification
OCSID		both	
SPINE	☑	static	Code verification
SPQR		static	Code X-ray
Object-Calculus		both	
RBML	☑	both	MDA
Slam-SL	☑	both	Inference in general
ODOL (OWL)	☑	static	Pattern Repository
URN		both	Ease of use
PEC	☑	static	Enforcing compiler
LePUS	☑	static	Pattern Repository
TLA		dynamic	

FOL		static	
Prolog		static	
BPSL		both	Inference in general

After reviewing related works and the benefits of using ontologies, which will be explained later, we also decided to employ them in our work. Although there are some ontologies available (e.g. ODOL), we did not use any existing one. Having a separate ontology is not a problem, since there is a possibility of connecting ontologies in a straightforward way. As will be seen in subsequent chapters, we can benefit from combining our solution with others – especially the WoP project [1].

3 The role of ontology in OBDPR

3.1 Using semantic web technologies in OBDPR

The idea of the semantic web allows automatic, intelligent inferring of knowledge, supported by ontologies. The basic idea of the semantic web is a different organization and storage of data and, subsequently, new possibilities for using this data [19]. The barrier that prevents more advanced usage of available data is believed to be the semantic poorness of today's solutions. The vast majority of data is presented as a very simple, non-structured human readable and human understandable material. The result is an inability to make real use of the enormous amount of available "knowledge". In order to overcome these difficulties, the concept of meta-data was introduced into the core of the semantic web. Using meta-data, so called smart agents can be used to search for information by content and to infer on gathered concepts. As a foundation, there has been a lot of work done with regard to common formats for the interchange of data and the common understanding of common concepts. This allows a person to browse, understand and use data in a more straightforward way, and a machine to perform some intelligent tasks on data automatically. Furthermore, semantic web ideas can be used in an internal enterprise information system for knowledge management in a different way to introduce new intelligent services. In the semantic web, knowledge is represented as graphs, and written down in an XML-based language called RDF (Resource Description Framework) [16]. RDF deals with URIs (another W3C standard for naming resources globally unique). The advanced use of semantically annotated data can only be accomplished using ontologies in RDFS or OWL (Web Ontology Language) [14] documents. There is also a language for efficiently querying RDF-represented knowledge, SPARQL [18]. The whole stack of semantic web technologies is available and described in [20] (see Figure 1).

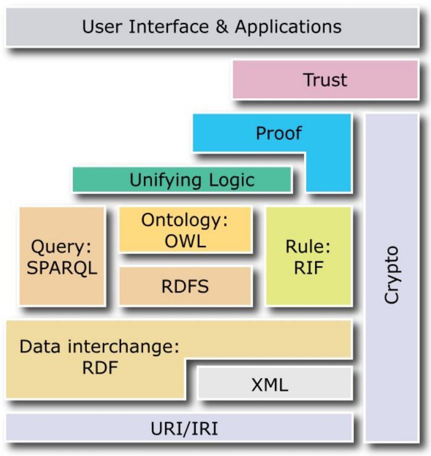


Figure 1: The semantic web technologies stack [20]

3.2 A new ontology

The semantic web allows knowledge to be expressed in a way that enables machine processing and its use in web environments by both intelligent agents and human users [23]. It is considered to provide an efficient way of presenting data, information and knowledge on the internet or in the scope of a global interconnected database. Since many semantic web technologies have reached high community consolidation and have become W3C standards (including RDF and OWL) it can also be considered a long-term platform for intelligent services based on a common knowledge base [20].

One of the enabling approaches used in the semantic web is metadata. It is supported by the concept of ontologies and has its foundation in W3C standards. Ontology describes the subject domain using the notions of concepts, instances, attributes, relations and axioms. Among others, concepts can also be organized into taxonomies whereby inheritance mechanisms can be used in ontology. Ontologies are built on description languages, such as RDF(S) and OWL, and add semantics to the model representation. Their formal, explicit and shared nature makes them an ideal object repository for catalogues.

With the presented facts, we also justify our decision to use ontologies as well as other semantic web technologies to provide a basis, not only for design pattern descriptions, but also for future intelligent services:

- Ontologies in the semantic web has its foundation in W3C standards.
- Ontology-based design pattern descriptions are computer readable and therefore suitable for automated (computer) processing.
- Transforming OWL and RDF based design pattern representations into other kinds of representations (in textual or graphical form) can be achieved easily with simple transformations.
- Enabling technologies are well established, recognized and extendable.

- They enable the exchangeability of design pattern descriptions in a straightforward way.
- The semantic web introduces technology that enables knowledge to be distributed.
- More and more OWL–enabled tools are available which can use and manipulate an ontology-enabled knowledge base.

OBDPR's underlying ontology is implemented using OWL. A core ontology fragment is shown in Figure 2. We use a hierarchical organization of pattern containers. Every pattern container may contain several pattern containers and patterns. This enables us to capture several divisions of design patterns, not only those found in fundamental literature. Every pattern can be included in several containers; the same is true for containers. Patterns themselves are connected in a more logical way by means of related, similar, composed patterns and pattern hierarchies (also mentioned as a pattern language by some authors). Not only patterns and pattern containers themselves are included in the ontology, but there are also real-world examples using patterns to give more meaning to the OBDPR user ("TestCase" class).

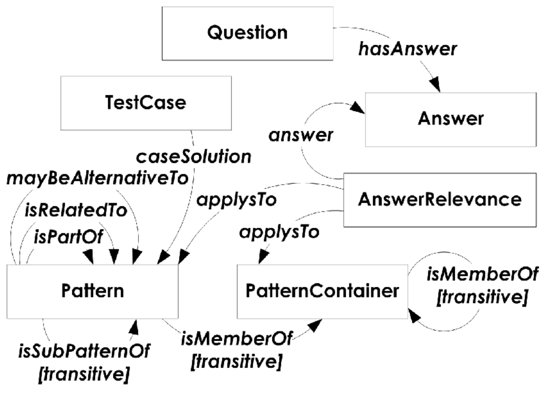


Figure 2: Core of OBDPR ontology

There are many benefits to using such ontology. Beware of transitive relations. Using a relation which has transitive properties can help significantly when dealing with design patterns and design-pattern containers. For instance: a service, built on top of OBDPR, has direct access to all members of a particular pattern container – without performing advanced searching. A pattern language (i. e. interrelated patterns) can also be presented in straightforward way. The solution is also prepared for connecting our own ontology and ODOL (ontology in WoP project [1]). We can introduce the relation theSameAs (a relation supported by OWL) between our Pattern concept and Pattern in ODOL. So we have automatic access to a formal representation not only to expert knowledge, but also the structure and behavior of a particular design pattern. Those were just a couple of strong mechanisms supported by the presented ontology.

Furthermore, the expert knowledge aspect is also supported by the presented ontology. Design pattern experts can provide experiences in question-answer pairs, which enables them to capture their implicit knowledge on design patterns. Not only experts can give experiences to tell which design pattern is used in a particular real-life situation ("Question" class), but they can also specify more possible solutions to a real-life situation ("Answer") with specified probability ("AnswerRelevance"). This value ranges from 0% to 100% and tells the user how likely it is that their particular candidate ("Pattern" or "PatternContainer") is used when the answer to a given question is confirmed as positive. Answers and possible candidates can easily be updated or added to questions at any time with the aid of a rich user-friendly web interface. For instance:
Question: *How do you want to create objects?*
Possible answer: *Separate construction of a complex object from its representation so that the same construction process can create different representations.* → You should use the *Builder pattern (100%).*
Possible answer: *Ensure that the class has only one instance.* → You should use the *Singleton pattern (100%).*
Possible answer: *Create objects without prior knowledge about their concrete classes.* → You can use several design patterns: *Prototype (33%)*, *Abstract Factory(33%)* or *Factory Method(33%).*

This knowledge can also be used by services, run on top of OBDPR in order to achieve intelligent functionalities, such as a guided question-answer dialogue for selecting patterns or verifying design decisions.

4 OBDPR – repository and platform

OBDPR is completely based on semantic web technologies. As a data store, it uses RDF. Since we do not want to rediscover all design pattern knowledge from scratch, we have also integrated knowledge found in other data sources (e. g. Wikipedia, Sun J2EE BluePrints, GoF online patterns etc). These are transformed to RDF, integrated and supported by the presented ontology (see Figure 3). Furthermore, OBDPR is not just a design pattern repository. It is a platform for building intelligent services to improve design pattern adoption. As such, it includes several functionalities:

- It holds design pattern descriptions, containers and an expert knowledge repository.
- Allows design pattern experts to annotate patterns with additional knowledge.
- Integrates knowledge on a particular design pattern from the web (Wikipedia, Sun Blueprints etc) and additional data sources.
- User-friendly transformations of raw RDF data.
- Indexes all the integrated data for supporting full text-search capabilities.
- Full access to RDF data to services built on the platform including questions and answers, which will enable intelligent services to use expert system-like proposing or validating services.

- A set of real world examples and appropriate design patterns solutions in order to enable services to be used to train users or to demonstrate the appropriate use of design patterns in real-world examples.

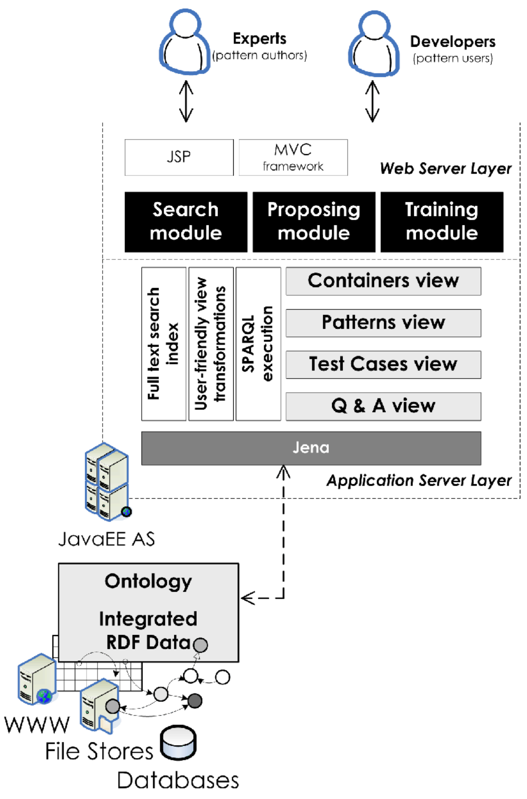


Figure 3: OBDPR architecture

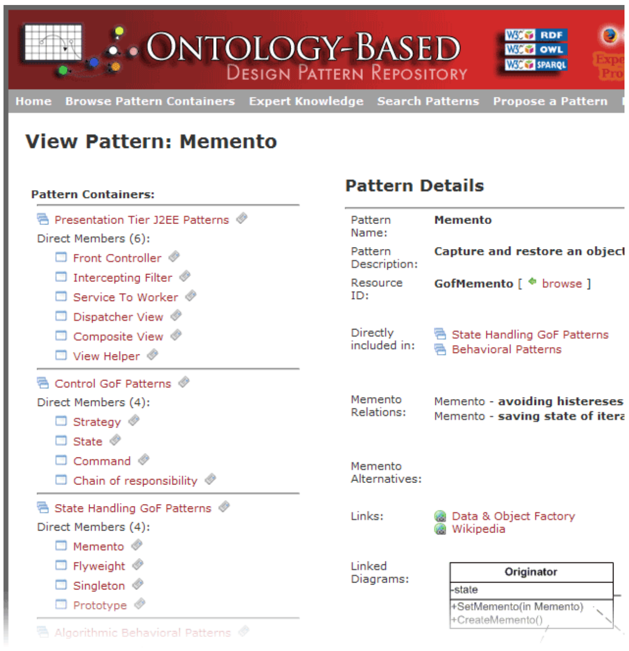
To achieve all the above-mentioned functionalities efficiently, we have also used other standard-based tools. For example, to access data written in RDF we use the Jena framework [9]. It is also exposed for services that will run on top of a platform. Figure 4 shows the user interface that uses an RDF presentation of design patterns, accessed with SPARQL queries. The data is then transformed to show a user-friendly view on design pattern container structure and selected design pattern details. This view is provided with minimum coding effort and is truly one of the most successful experiences within OBDPR.

The current OBDPR prototype implements all functionalities mentioned at the start of this chapter. It also offers services built on top of them:

- A full-text search service,
- A design pattern proposing service and
- A training service

All of them are primarily intended to help the design pattern novice.

At the moment, the OBDPR prototype includes all design patterns found in GoF [5] and J2EE [2] design pattern catalogues. It is not limited to those since it is possible to include additional patterns – even those



View Pattern: Memento

Pattern Containers:

Presentation Tier J2EE Patterns
Direct Members (6):
Front Controller
Intercepting Filter
Service To Worker
Dispatcher View
Composite View
View Helper

Control GoF Patterns
Direct Members (4):
Strategy
State
Command
Chain of responsibility

State Handling GoF Patterns
Direct Members (4):
Memento
Flyweight
Singleton
Prototype

Algorithmic Behavioral Patterns

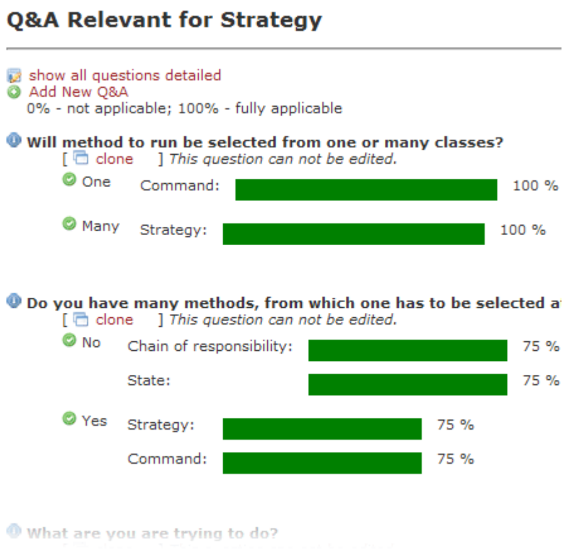
Pattern Details

Pattern Name:	Memento
Pattern Description:	Capture and restore an object
Resource ID:	GofMemento [browse]
Directly included in:	State Handling GoF Patterns Behavioral Patterns
Memento Relations:	Memento - avoiding histereses Memento - saving state of itera
Memento Alternatives:	
Links:	Data & Object Factory Wikipedia

Linked Diagrams:	Originator
	-state
	+SetMemento(in Memento)
	+CreateMemento()

Figure 4: Simple pattern view

recognized in a particular enterprise. Additional expert knowledge can also be provided. Implementing this knowledge is supported for services as well as for browsing with a user-friendly interface – using relatively simple SPARQL queries and transformations (see Figure 5).


Q&A Relevant for Strategy

show all questions detailed
Add New Q&A
0% - not applicable; 100% - fully applicable

Will method to run be selected from one or many classes?
[clone] *This question can not be edited.*

One	Command:	100 %
Many	Strategy:	100 %

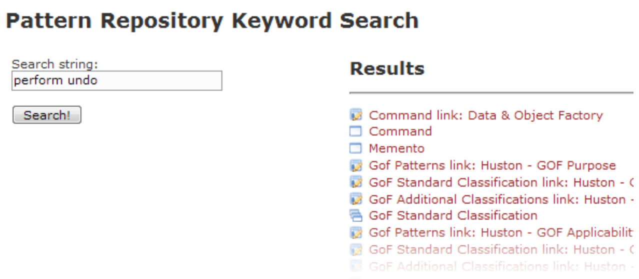
Do you have many methods, from which one has to be selected a
[clone] *This question can not be edited.*

No	Chain of responsibility:	75 %
	State:	75 %
Yes	Strategy:	75 %
	Command:	75 %

What are you are trying to do?

Figure 5: Expert knowledge view

The implementation technology for OBDPR is Java EE with a Jena [9] framework for accessing and performing core semantic operations on data and ontology. A simple user interface framework with basic functionalities like a raw and user-friendly view on the repository is prepared as previously shown. A framework is fully prepared to host additional services, which are developed in the future. To address this requirement efficiently, we have implemented an MVC design pattern into our solution. As a case, we have implemented three services on top of the platform. Since they are the core ones, they can also be used by other services. They enable us to use full text search capabilities in OBDPR (Figure 6) as well as training (Figure 5) and proposing services (figure 7). Not only is the data in OBDPR indexed for a full text search, but also data from the web, such as design-pattern-related content from Wikipedia and other design pattern related pages. The underlying ontology also improves the full text search capabilities.

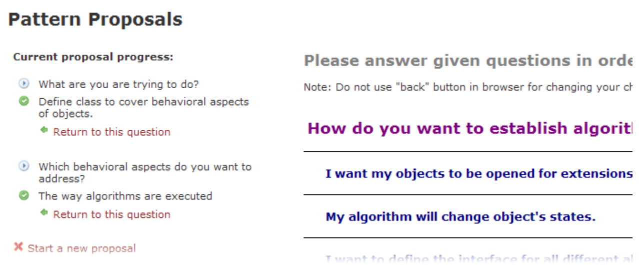

Pattern Repository Keyword Search

Search string:
perform undo

Results

Command link: Data & Object Factory
Command
Memento
Gof Patterns link: Huston - GOF Purpose
GoF Standard Classification link: Huston - (
GoF Additional Classifications link: Huston -
GoF Standard Classification
Gof Patterns link: Huston - GOF Applicabilit
GoF Standard Classification link: Huston - (
GoF Additional Classifications link: Huston -

Figure 6: Search module


Pattern Proposals

Current proposal progress:

What are you are trying to do?
Define class to cover behavioral aspects of objects.
Return to this question

Which behavioral aspects do you want to address?
The way algorithms are executed
Return to this question

Start a new proposal

Please answer given questions in orde

Note: Do not use "back" button in browser for changing your ch

How do you want to establish algorit

I want my objects to be opened for extensions

My algorithm will change object's states.

I want to define the interface for all different al

Figure 7: Proposing module

5 Preliminary experiment

We preliminary tested our approach and a platform for advising on design patterns by conducting an experiment on a group of software developers who had different levels of expertise in the field of design patterns. A series of 19 design problems were presented to each of them. Each participant tried to identify the most appropriate design pattern to solve each of the design problems. The experiment was done in four phases: in the first phase, participants had to answer a few questions concerning their development background and level of expertise. In the second phase, they were given the opportunity to solve their design problems without any assistance and/or tools. In the third phase, the platform was provided to help them solve the same set of design problems. In the end, a post-experiment survey was conducted to gather participants' opinions regarding the usefulness of the platform.

We invited 10 software developers to participate in the experiment. According to their own assessment of their design patterns expertise, five of them self-described as "good", two as "very good" and two as "excellent". Only one participant claimed poor knowledge of GoF design patterns. The level of expertise was assessed in terms of how many patterns of the GoF catalogue a participant could identify. A comparison of the results achieved in the second and third phase showed that only one participant did not make any progress when using the platform. For the rest of the participants, the platform helped identify, at a minimum, an additional 50 percent in correct solutions. Using the statistical analysis of the results (paired t-test) we determined that the difference between the number of correct solutions found both with and without the platform is statistically significant (P = 0.000706). The results of the post-experiment survey have shown that only one of the participants found the use of the platform to be less efficient than searching for an appropriate solution without the platform. A decreased standard deviation in the results showed that the efficiency of the less-experienced developers (according to design patterns) became more similar to experienced developers. At the same time, the mean value of successfully solved problems rose significantly. We can conclude that developers with less experience in the area of design patterns benefited the most from using the platform. By comparing the frequency of the correct solutions representing a particular design pattern and the frequency[1] of use for the same pattern in practice, we found that they did not directly correlate. This can be at least partly explained by the fact that the design problems presented to participants were not of equal complexity. The second possible explanation is that a higher frequency of use does not necessarily mean that the design pattern is better known or indeed easier to understand.

It should be noted that the results have to be taken with some caution, since the number of participants was quite small. Nevertheless, the experiment shows that our platform for advising is helpful to those developers who have a relatively poor knowledge of design patterns. For developers who already have solid expertise, this approach does not offer as many advantages, because the questionnaire does not follow their thought processes (mind maps). However, should a wider variety of design patterns be covered by the platform, the differentiation between experienced developers and those who lack expertise would presumably decrease.

For detailed information about experiment please see [26]. In that paper you can find detailed experiment structure, proposing service is also discussed in depth.

6 Further work

We have developed an integral web-based platform, primarily to help select design patterns. To strengthen confidence in the results, some rigorous real life experiments should be performed in addition to this initial one. They might show if and how much OBDPR helps with formally presented design patterns when adopting design patterns. As previously shown, some

[1] Frequency of use was taken from http://www.dofactory.com/Default.aspx

preliminary experiments have already been performed. Since the results were promising (design pattern adoption rose significantly) we are quite confident that we are on the right track. More rigorous experiments are planned for the future.

Besides experimenting with a repository, there is also another idea to expose OBDPR to simple software interfaces. It would not only enable further integration but can also enable the development of plug-ins for the most popular development tools, such as Eclipse, NetBeans or Visual Studio. Having OBDPR always at hand during development would certainly seem beneficial.

Even existing services need some improvements before going into production. For example, we are trying to personalize the proposing service. The proposing component could learn about the user from past proposals and, for instance, ask personalized questions or ask more questions to verify possibly contradictory answers. The idea behind the proposing service includes verifying if the developer knows what a certain question mean. This could be achieved with question redundancy: if the developer answers a question with an option that prefers pattern A and another question with an option that does not prefer pattern A, it is possible that the developer is confused. With this in mind, we can reduce questions asked during the proposing service, if we consider the developer's past dialogues. OBDPR enables the analysis of exhaustive logs of usage. We have data on each proposing process if a selection is well done. If not, we can review which question was shown to be problematic and where the user starts to get confused (by measuring several attributes for each question including repetition number, time spent, premature finishing etc). This can be used as guidance for experts to review and improve questions and answers or to provide more questions connecting particular candidates.

After performing research activities by means of experimenting with the platform on industry developers, we plan to develop a holistic methodology for design pattern selection. It will include both a design pattern expert and user activities. OBDPR will be given the role of an enabling tool for the developed methodology. To take full advantage of formalized design patterns aspects, there is basically no limitation for creating additional services on top of the platform.

7 Conclusions

The platform (Ontology-Based Design Pattern Repository – OBDPR) presented in this paper is the basis for automatic and intelligent services built on top of formally presented design pattern knowledge. The main aim of OBDPR is to introduce formal methods of design pattern representation in order to drawn upon capabilities known from the area of artificial intelligence. It also simultaneously keeps patterns in human-friendly form. Therefore, the semantic web approach and technology were used. OBDPR addresses the challenges of selecting, understanding and using design patterns in the rapidly increasing number of design patterns.

In the paper, we have presented several important components for our approach:

- The proposed formal design pattern presentation technique can easily be used by automatic intelligent services as well as by human users. It can additionally be integrated with existing presentations, especially ODOL [1].
- A new, fully functional OBDPR with the capability to serve as a basis for more advanced services (we have so far developed a searching, proposing, training and validating service).
- The platform and services have initially been exposed to real-world usage. The initial experiment has encouraged us to carry on with our work and perform additional, rigorous experiments.

We are confident that our work can contribute to an increase in using design patterns, especially by helping to find a suitable design pattern for a given situation. This issue constitutes a great challenge for the typical developer. OBDPR was therefore developed primarily to capture design patterns, explicit and implicit expert knowledge, and to enable the further development of intelligent services and to test our belief that we can improve design pattern adoption.

Introducing semantic web concepts and technology into the design pattern field has revealed itself to be the correct solution so far. It creates new possibilities for making design patterns more approachable for software engineers.

8 References

[1] A. H. Eden et al, Precise Specification and Automatic Application of Design Patterns, International Conference on Automated Software Engineering, IEEE Press, 1997.

[2] Core J2EE Patterns, http://java.sun.com/blueprints/corej2eepatterns.

[3] D. K. Kim at al, A UML-based Metamodeling Language to Specify Design Patterns, Proceedings of the Workshop Software Model Eng. (WiSME) with Unified Modeling Language Conf. 2003, October 2003.

[4] D.Manolescu, W. Kozaczynski, A. Miller, J. Hogg, "The Growing Divide in the Patterns World", IEEE Software, Vol. 24, No. 4., July/August 2007, pp. 61-67.

[5] E. Gamma et al, Design patterns: Elements of reusable object orientated software, Addison Wesley Longman, 1998.

[6] Gerson Sunyé et al, Design Pattern Application in UML, ECOOP'00, http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/1850/18500044.pdf.

[7] J. M. Rosengard, M. F. Ursu, Ontological Representations of Software Patterns, KES'04, Lecture Notes in Computer Science, Springer-Verlag, 2004, http://w2.syronex.com/jmr/pubs/2004/ontology-pattern.pdf.

[8] J. M. Rosengard, M. F. Ursu, Ontological Representations of Software Patterns, KES'04, Lecture Notes in Computer Science, Springer-Verlag, 2004, http://w2.syronex.com/jmr/pubs/2004/ontology-pattern.pdf.

[9] Jena Semantic Web Framework, http://jena.sourceforge.net.

[10] L. Rising, "Understanding the Power of Abstraction in Patterns", IEEE Software, July/August 2007, Vol. 24, No. 4., pp. 46-51.

[11] L. Rising, The Pattern Almanac 2000, Addison Wesley, 2000.

[12] Marcus Fontoura and Carlos Lucena , Extending UML to Improve the Representation of Design Patterns, Computer Science Department, Pontifical Catholic University of Rio de Janeiro.

[13] Microsoft, Microsoft Patterns & Practices, http://msdn.microsoft.com/practices.

[14] OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features.

[15] R. Singh, Drive: An RDF Parser for .NET, http://www.driverdf.org/.

[16] RDF/XML Syntax Specification, http://www.w3.org/TR/rdf-syntax-grammar.

[17] Schmidt, D.C., "Using Design Patterns to Develop Reusable Object-Oriented Communication Software", Communications of the ACM, October 1995.

[18] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query.

[19] T. Berners-Lee, "Business Model for the Semantic Web", http://www.w3.org/ DesignIssues/Overview .html.

[20] W3C, "Semantic Web", http://www.w3.org/2001/ sw/.

[21] J. Helin, P. Kellomäki, T. Mikkonen. Patterns of Collective Behavior in Ocsid. p. 73-93. Design Patterns Formalization Techniques. IGI Publishing. March 2007.

[22] T. Taibi, Design Patterns Formalization Techniques, United Arab Emirates University, UAE, Preface, IGI Publishing. December 2006.

[23] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American, 284 (5) (2001) 28-37.

[24] P. Bonillo, N. Zambrano y Eleonora Acosta, Methodologic Proposal for Business Process Management sustained in the use of Patterns, Journal of Object Technology, vol. 7, no. 7, September - October, pp. 131-145, http://www.jot.fm/issues/issue_2008_09/article5.

[25] P. Gomes, F.C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. Ferreira, C. Bento, Selection and Reuse of Software Design Patterns Using CBR and WordNet, 15th International Conference on Software Engineering and Knowledge Engineering, SEKE 2003, Proceedings, (SEKE'03), pp. 289-296.

[26] M. Heričko, I. Brejc, L. Pavlič, V. Podgorelec, A Question-Based Design Pattern Advisement Approach, Journal of Systems and Software, Submitted to publication in November 2008.

[27] U. Zdun, Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis, Software: Practice & Experience, vol. 37, no. 9, p 983-1016, Wiley, 2007.