Computer Science and Information Systems 11(2):645–664 DOI: 10.2298/CSIS130824025P

A Question-Based Design Pattern Advisement Approach

Luka Pavlič, Vili Podgorelec, and Marjan Heričko

Faculty of Electrical Engineering and Computer Science, University of Maribor
2000 Maribor, Slovenia
{luka.pavlic, vili.podgorelec, marjan.hericko}@uni-mb.si

Abstract. Design patterns are a proven way to build flexible software architectures. But the selection of an appropriate design pattern is a difficult task in practice, particularly for less experienced developers. In this paper, a question-based design pattern advisement approach will be proposed. This approach primarily assists developers in identifying and selecting the most suitable design pattern for a given problem. We will also propose certain extensions to the existing Object-Oriented Design Ontology (ODOL). In addition to the advisement procedure, a new design pattern advisement ontology will be defined. We have also developed a tool that supports the proposed ontology and question-based advisement (OQBA) approach. The conducted controlled experiment and two surveys have shown that the proposed approach is beneficial to all software developers, especially to those who have less experience with design patterns.

Keywords: design patterns, pattern selection, ontology, semantic web, selection algorithm.

1. Introduction

One of the basic characteristics of any engineering discipline is that new systems are built and developed from existing, already proven reusable elements using well known approaches and best practices. Reuse has become an essential and important strategy - also in the area of software and information systems development. The reuse of concrete assets and software elements such as functions, classes and components has already been well established and continues to be practiced on a daily basis. Attention should also be placed to reuse at higher levels of abstraction i.e. to software patterns. A pattern is a form of knowledge for capturing a recurring successful practice [1]. Design patterns capture the best practices for solving recurring software design problems. They explicitly capture knowledge that experienced developers understand implicitly and facilitate training and knowledge transfer to new developers [2]. A survey conducted by the Microsoft Patterns and Practice Group [3] indicated a low adoption of design patterns among practitioners: respondents estimated that no more than half of the developers and architects in their organization use software patterns. Therefore, bridging the gap between expert design pattern communities and the typical design pattern user is critical for achieving the full benefits of design patterns [4].

In [5] we can find an observation that it might be difficult to find a suitable design pattern even in a catalogue such as the GoF (Gang Of Four) Design Pattern Catalogue – with no more than 24 patterns. Several hundred software patterns have already been published. The Pattern Almanac [6] published in 2000, provided a list of over 700 previously published patterns organized into 70 categories. Since then the number of software patterns and catalogues has increased significantly. Consequently, software developers have been experiencing more and more problems identifying appropriate catalogues and finding patterns that match their design problems. Useful patterns might easily be overlooked. That is why a manual identification and application of patterns is not efficient enough. For the efficient identification and selection of suitable design patterns, automated support is of critical importance.

There were some efforts within the research community to automate the application of design patterns [7,8,9]. Mainly these efforts were focused on code generation and the identification of design patterns in existing designs and/or source code and less on the selection of suitable design patterns. Although it is not reasonable to believe that the responsibility for selecting design patterns will be completely delegated to tools, they could provide at least some advice for design pattern(s), which may be potentially useful in a given situation. In order to be able to develop and provide efficient automated tools, we need an adequate design pattern description approach. An appropriate knowledge representation technique should be computer-readable, based on standard technologies, extendable and relatively simple for developers to work with. Consequently, the main focus and aim of this work is twofold:

– To define a means for capturing information and knowledge on design patterns in a form that would enable a computer to process and use it in a more intelligent way whilst also keeping it readable for humans.
– To provide assistance for software developers searching for an appropriate design pattern for a given design problem.

The main idea of the proposed question-based approach originates from our undergraduate course on software design patterns. Our students are taught how to identify suitable design patterns and/or choose the most appropriate one when they are unsure about two or more design patterns (e.g. whether to use the Adapter or Façade pattern). In order to select the most appropriate design pattern some questions -- such as those presented in Table 1 -- have proven to be helpful.

Table 1. Sample "recipe" on how to differentiate the applicability of two design patterns

Question		
Could the necessary functionality be found in the existing classes?	Yes	Yes
Do we need a simpler interface for existing classes?	Yes	No
Is there a predefined interface that a class under development should be compliant with?	No	Yes
Are class objects expected to demonstrate polymorphic behaviour?	No	Probably
Do we want to change the way of how existing functionalities are used?	Yes	No
Possible solution	Facade	Adapter

Figure 1 shows the holistic view of the proposed Ontology and Question-Based Advisement (OQBA) approach. An ontology-based technique is used to represent knowledge on design patterns, catalogues, pattern containers and pairs of questions/answers used for reasoning regarding a suitable pattern for a particular design problem. This design-pattern expert knowledge, gathered in the Ontology-Based Design Pattern Repository, is then used for guiding the question/answer interaction with the developer in order to identify and propose a suitable design pattern, as well as to verify its applicability for a given situation.

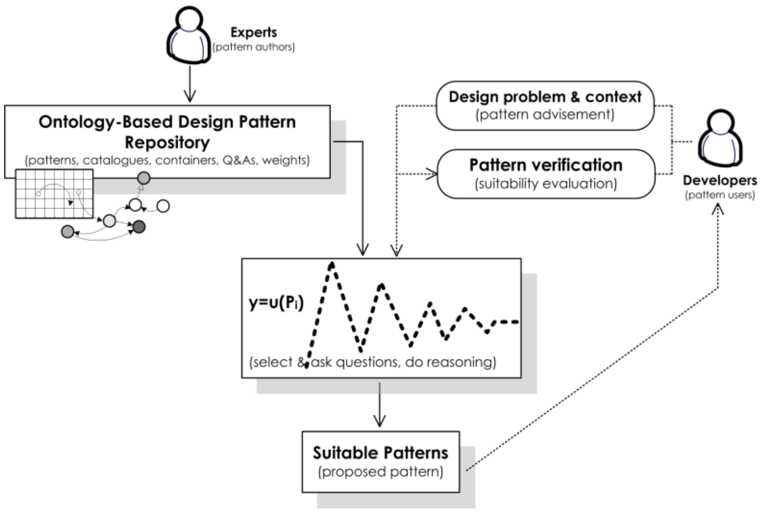

Fig. 1. The holistic view of the proposed OQBA approach

The approach presented in this paper is focused on object-oriented design patterns. To show the benefits, we experimented with the GoF design patterns catalogue, since even catalogues with few patterns proved to be problematic when identifying the most suitable pattern[5]. At the moment we cannot claim the approach to be general to all software patterns. However, it is our belief that a primary idea of question-based advisement approach could eventually be used also for other software patterns. However for that it would be necessary to formalize additional concepts and relationships that are specific to other software patterns groups.

This paper is organized as follows: An overview of related work and research is given in the second section. The third section discusses problems related to design pattern descriptions and presents our extensions to an existing ODOL ontology (Object-Oriented Design Ontology). It also introduces a new ontology named DPAO (Design Pattern Advisement Ontology). Section four gives a general description of the proposed question-based approach and presents algorithms for assessing the usability of patterns. In addition, the advisement procedure as it is implemented in the DPEX (Design Pattern Expert) tool is introduced. The description of the conducted surveys and experiment with their results are given in the fifth section. Finally, some concluding remarks and directions for future work will be presented.

2. Related Work

Kung et al. [4] presented a five-step methodology for constructing an expert system is presented that suggests design patterns to solve a design problem. The focus is on the collection and analysis of knowledge on patterns in order to formulate questions, threshold values and rules to be used by an expert shell. A prototype of the Rule-Based tool (for a subset of GoF patterns) was developed. It selects the design pattern through dialogue with the software designer to narrow down the choices. A preliminary evaluation of the proposed methodology was done on a group of ten students. The focus of the experiment was not on evaluating the efficiency of a tool in finding a solution for a given problem situation. Rather it was aimed at proving that, with a known design pattern, a tool would lead the user to a suitable suggestion or solution. Therefore, the focus was on verifying that a tool would confirm the suitability of a particular design pattern. Our approach is different: we want to suggest which design pattern might be useful in given context.

Gomes et al. [10] introduced similar approach that is based on Case-Based Reasoning (CBR) and lexical database WordNet [10]. The approach is based on the idea that a system can learn to select and to apply design patterns if it can store and reuse knowledge on pattern usage experience. An application of a specific pattern to a specific software design is represented in the form of a design pattern application (DPA) case. A DPA case describes a specific situation where a software design pattern was applied to a class diagram. Our approach does not require a developer to prepare a model first. The information needed to identify a suitable solution is gathered using a guided dialogue, where a developer provides information on problem characteristics by choosing items from the lists of available answers.

Birukuo et al. [11] proposed a multi-agent system that supports collaboration between developers in order to chose the suitable design pattern for a given problem. Suggestions are made using experiences from a group of developers who were previously faced with a similar problem. The problem for which an appropriate pattern is searched for is described using a “bag of words” approach as a sequence of terms. Having a vector that maps these terms to a vocabulary of all terms, a search for patterns can be initiated at all agent nodes. The approach assumes that the developers will provide a textual description of the problem. Their work could be seen as a complementary approach to ours – in the case that our approach would not result in a suitable recommendation, a term vector could be automatically formed and used as an input to their system.

Ontologies have already been successfully used to describe design patterns within the scope of the ”Web of Patterns“ (WoP) project [8]. The main goals of the WoP project were (1) to define an ontology with which object-oriented models could be described, (2) to describe design patterns, anti-patterns and refactorings and (3) to develop tools based on this ontology which would be useful for software engineers. The main focus of the project was on finding pattern instances in Java projects as well as on refactoring and anti-patterns. The project is not directly related to our work. However, one of the artifacts of the WoP project is the Object Oriented Software Design Ontology (ODOL), which contains a set of concepts and relationships used in all of the more important object-oriented (OO) languages. ODOL includes basic OO concepts like classes and methods, OO design concepts and high-level concepts like design patterns, pattern categories, aggregations etc. ODOL (available at [12]) is an open ontology so it is possible to extend it with concepts that are currently not present in the ontology. During our research we extended already existing ontology ODOL with proposing capabilities as described later in paper. It was extended with additional concepts and a new ontology that was needed to provide an infrastructure for the application of the question-based approach.

3. Using Ontology for Describing Knowledge of Design Patterns

3.1. Towards a More Suitable Design Pattern Description

In order to develop a tool that would assist and advise the developer on the most appropriate design patterns, or a combination of patterns, for a given design problem, knowledge and experiences on design patterns should be gathered and described. Since 1994, when design patterns were introduced, many different approaches have been used to document them. In general there are three main categories of descriptions: (1) informal representations, (2) semiformal representations based on graphical notations such as UML, and (3) various formal representations which also include notations using semantic web technologies. Design patterns are traditionally described using natural language and published in printed catalogues [5]. These documents are loosely structured in a canonical form which consists of a series of fields: name, intent, applicability, structure, participants, consequences, implementation etc. Because of its loose structure, this kind of representation is less suitable for knowledge management and sharing [13]. Informal representations based on a canonical form do not enable the desired and necessary level of design pattern identification and application. For this we need more structured representation forms. This has led some researchers to devise more formal presentations, mainly by using existing mechanisms of UML or by extending UML specifications [14,15]). Design patterns are usually described using class diagrams and interaction diagrams – primarily parameterized communication diagrams. The main drawback of these approaches is an over-reliance on visual specifications with UML diagrams and limited support for the behavioural aspects of design patterns [16]. They are efficient for a basic understanding of patterns since they cover their structural elements. But they do not provide information and knowledge on high level aspects such as intent, usability and consequences. The introduction of automated support requires a formal approach to design pattern description.

3.2. How Can Ontologies and Semantic Web Technologies Help?

The semantic web enables knowledge to be expressed in a way that enables machine processing and its use in web environments by both intelligent agents and human users

[17]. It is considered to provide an efficient way to present 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 - Resource Description Framework and OWL - Ontology Web Language) it can also be considered as a long-term platform for intelligent services based on a common knowledge base [18].

One of the enabling approaches used in the semantic web is metadata. It is supported with the concept of ontologies and has a foundation in W3C standards. Ontology describes the subject domain using the notions of concepts, instances, attributes, relations and axioms. Concepts are typically organized into taxonomies through which inheritance mechanisms can be used in an ontology. Ontologies build 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 a design pattern description but also for providing additional information, relations and rules, needed to define and implement the OQBA approach:

– Ontology-based design pattern descriptions are computer readable and therefore suitable for automated (computer) processing.
– If design pattern descriptions are provided in the form of ontological definitions, they can be presented in textual or graphical form as well as transformed to presentation forms customized for developers (developer friendly representation).
– Ontology and related technologies are well established, recognized and also extendable, whereas the semantic web is becoming an enabling factor for better knowledge management and the management of a high volume of data that still has to be inter- and cross-linked.
– Ontologies enable the establishing and revising of a knowledge base on design patterns based on common, accepted standards and technologies.
– Navigation based on relationships between patterns helps with a better understanding of a pattern space [19]; consequently ontology-based descriptions are ideal because navigation-based relationship investigation capabilities are inherent to ontologies and semantic web technologies.
– Ontologies enable the exchangeability of design pattern descriptions and are extendable.
– Several knowledge sources can easily be integrated using relatively simple transformations.
– The knowledge base can be distributed (on the web or in closed networks).
– Third-party ontology (OWL)–enabled tools are available, which can extend the use of an ontology-enabled knowledge base.
– The behaviour of developed system can be improved simply by changing the ontology and/or data without changing the system itself.
– Even generic search engines can be impacted to retrieve more reasonable results by using ontology-rich data.

These were the main reasons to use the results of the WoP project and ODOL ontology as a starting point for defining a novel approach and tools that are based on ontologies and other semantic web technologies. Alongside the definition of the question-based advisement approach we have developed a new ontology, called DPAO (Design Pattern Advisement Ontology) that represents the foundation for the proposed question-based approach.

The WoP project was oriented toward design pattern instance identification and not on the selection of suitable design patterns. Consequently, ODOL ontology does not provide and/or enable one to capture all the information needed to introduce and apply the OQBA approach. We needed additional information that would direct and guide the advisement procedure as well as the process of verification so that the selected design pattern actually represents a reasonable solution for a given problem. That is why we extended the ODOL ontology with additional concepts that would enable the efficient grouping of design patterns. In addition, we developed a new ontology, named DPAO, which provides the infrastructure for two main aspects of the advisement:

– Advisement on an appropriate design pattern for a given problem situation: Developers are faced with a problem for which they presume that the solution in the form of pattern already exists, but they have no idea which pattern it is or which pattern group or catalogue they should look at.
– Advisement on the applicability of a particular design pattern for a given problem situation: Developers believe they know which are suitable design patterns within a given context, however they would like to verify their choice and/or evaluate the suitability of the identified design pattern(s).

3.3. Extending ODOL with New Concepts and Relations - Pattern Container, Related, Alternative and Composed Patterns

The existing ODOL ontology has two concepts for identifying and classifying design patterns: Pattern and PatternCatalog. This two-level hierarchy becomes insufficient when the number of design patterns stored in the catalogue starts to rise. PatternCatalog can only inlcude Patterns, not other also PatternCatalogs. For these reason we have defined a new concept: PatternContainer. As the name suggests, it should serve as a container for patterns and/or other pattern containers. PatternContainer provides a means to group an arbitrary set of patterns based on selected aspects. In addition, the same design pattern can be an element of many different groups/categories. Even more: design patterns can be related to each other while being part of different containers/catalogues. As can be seen in Figure 2, PatternContainer becomes the main class for pattern grouping. In Figure 2, the original ODOL concepts are shown as shadowed and/or written in italics whereas new concepts are in bold. The existing PatternCatalog concept then becomes just a type of PatternContainer. In order to provide efficient advisement some additional concepts were added to ODOL. They enable us to connect an existing design pattern with possible related patterns such as:

– Alternative design pattern that could represent a reasonable alternative to a particular pattern – e.g. Façade to Adapter, Visitor to Observer, Decorator to Adapter, Builder to Abstract factory.

– Related design pattern as a pattern for which it is very likely to be used in combination with a selected pattern, e.g. Chain of Responsibility together with Composite, Memento together with Command, Abstract Factory implemented with Singleton, Composite processed with Iterator.
– Composite pattern that provides higher granularity of interrelated patterns, e.g. the MVC (Model-View-Controller) pattern is a combination of patterns and incorporates Composite, Observer and Strategy.

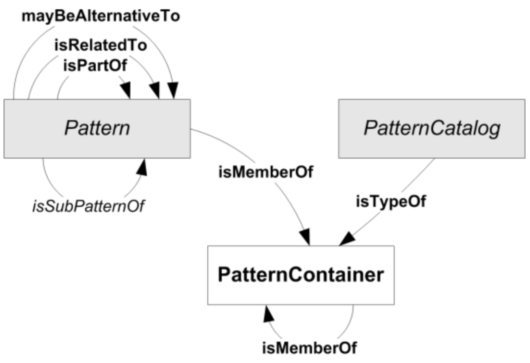

Fig. 2. Extensions (in bold) of the ODOL ontology

With these extensions, the descriptions of GoF design patterns in ODOL ontology can be updated, for instance, with definitions for the three GoF pattern categories, namely: creational, structural and behavioural patterns. Existing definitions of GoF patterns could then be expanded with categories as well as information on related and alternative patterns. ODOL ontology has been extended with a few additional relations that are of crucial importance for efficient automation and support in the process of suitable design pattern identification and selection.

3.4. Design Pattern Advisement Ontology (DPAO)

In addition to the knowledge on patterns, a precondition for successfully advising on patterns is the tool. It should be able to ask the right questions and according to the answers lead the dialogue with a developer until enough certainty is reached to propose a certain design pattern. We need an ontology that provides the basis for: (1) the selection of appropriate patterns and (2) the verification of candidates that are appropriate for a particular design problem. The developed DPAO ontology meets both goals.

Figure 3 shows the concepts that cover the aspect of selecting the most appropriate design pattern for a given design problem in a particular context. AdviceArea class represents a set of matrices (AdviceMatrix) which are connected in a graph. The advice process starts with the initial matrix. The initial matrix typically holds the knowledge necessary to make decisions about general context whereas latter matrices can also hold some context-specific knowledge. AdviceMatrix is three-dimensional structure and is constructed from questions, answers and pattern/pattern container candidates. A candidate is modeled with the class AdviceMatrixNode and can be a terminal one (AdviceMatrixLeafNode leading to particular Pattern or PatternContainer) or a link to another AdviceMatrix (AdviceMatrixNodeMatrix). The matrix content is modelled as AdviceMatrixCell, holding in CellAssessment the weight value of how relevant a candidate can be if a particular answer for a given question is selected. Since assessments can be given by more than one expert, assessments are grouped into AssesmentSets. Expert knowledge can be gathered using simple in-house developed tools. Weights, questions, answers and their relations to patterns and pattern containers are freely inserted by experts. For example data and meaning during the dialogues please see section 4.1.

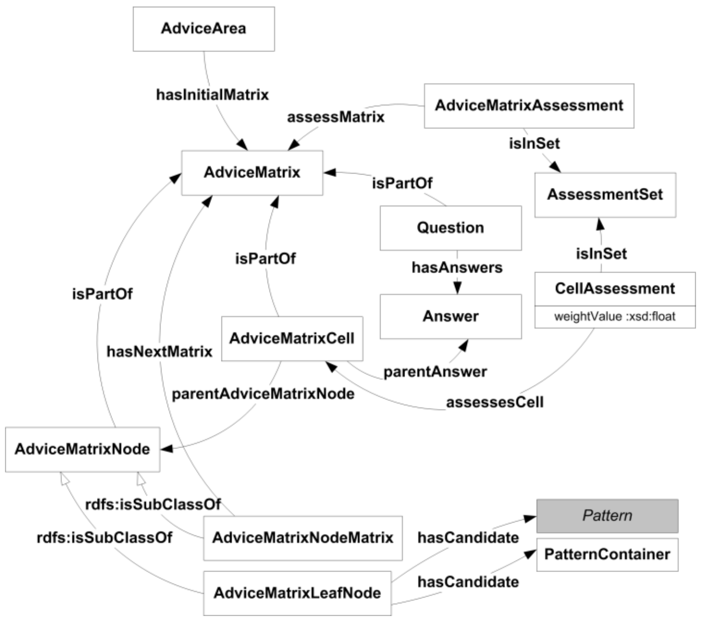

Fig. 3. Concepts in DPAO ontology enabling the selection of relevant design pattern(s)

The second aspect, supported by the new approach, is to verify the relevance or possibility to use a pattern in a known problem situation. In this case the developer has already selected a solution (pattern) to be used and would just like to verify if the provided pattern is relevant or there are more relevant patterns. This aspect is covered by the ontology concepts depicted in Figure 4.

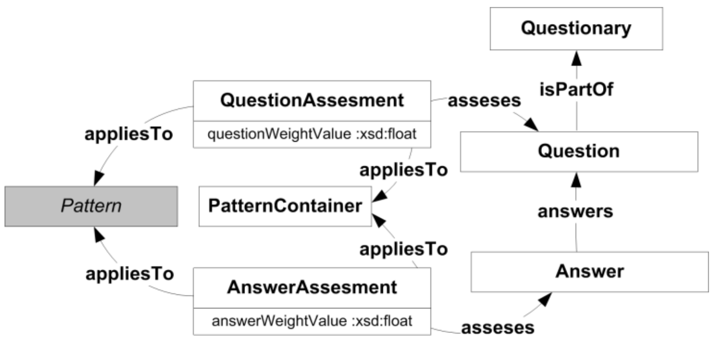

Fig. 4. Concepts in the DPAO ontology enabling relevance verification of a particular design pattern

Obviously, the DPAO ontology enables us to connect some question-answer pairs to a candidate, i.e. pattern or pattern container. Most of the introduced concepts in Figure 4 are known from prior figures. Questionary is attached via Question instances to a particular candidate. The weight of the question is modelled with the questionWeightValue held as a numeric value in QuestionAssessment, and respectively in the answerWeightValue attribute in AnswerAssessment for every possible answer to the question. The details of expert-inserted weights in terms of meaning and usage is described in details in section 4.1. Experts use user-friendly tool for inserting questions, answers and weights, as well possibility to review and alter already existing data in the ontology.

4. The Advisement Procedure

Utilizing knowledge on design patterns as well as a set of question/answers pairs that indicate the applicability of design patterns, we can apply the advisement procedure. In general the question-based advisement procedure consists of five phases:

- **Phase 1: Reducing a set of possible solutions to a subset of pattern containers**
  Using knowledge in the Ontology-Based Design Pattern Repository, especially advice matrices, and based on user's answers, the overall intention of applying a design pattern should be revealed, e.g. is it related to a specific development phase, technology, domain or any other criteria used to form pattern containers in the repository. The procedure is directed by initial matrices and corresponding question/answer pairs, using algorithms and the usability function presented in chapters 4.1 to 4.3. As a result we get a set of containers that might include a suitable solution.
- **Phase 2: Identifying the most suitable pattern container**
  An additional narrowing down is necessary in order to identify the atomic container with concrete design pattern candidates. The selection of an appropriate container is also based on advice matrices and algorithms, presented in the sections that follow. As a result, we get a single pattern container. e.g. - after the GoF container is

selected in the first phase, in the second phase we would identify one of the GoF pattern containers, namely behavioural, creational or structural patterns.

- **Phase 3: Selecting the most suitable pattern in a given design pattern container**
  In this phase, we identify the most appropriate design pattern within the container found in the previous phase. To achieve this we select questions that are common to as many patterns in the container as possible. For this we use groups of questions and answers for relevant patterns. The result is advice based on the most relevant design pattern and eventually related alternative solutions (design patterns).
- **Phase 4: Verification of the proposed design pattern**
  In this phase, the verification is done to see if any of alternative patterns can be equally or even more relevant. At this stage all questions and trade-offs, connected with a selected pattern, are taken into consideration especially those that can identify negative effects. The same is done for eventual alternative patterns. If the alternative pattern demonstrates less negative consequences it is applied instead of the initially proposed design pattern. The aim of this phase is also to eliminate possible mistakes, caused if a developer misunderstands a question in the advisement procedures.
- **Phase 5: Identifying related design patterns**
  We want to identify design patterns that might be useful in combination with the already selected pattern. A list of all design patterns that are related to the selected verified design pattern is constructed first. These patterns are then evaluated using the same procedure as applied in phase 4. Thus, composite design patterns can also be identified and applied.

4.1. Calculating the Pattern / Pattern Container Usability

Our aim is to identify and suggest the most appropriate design pattern that could be used by a user in the situation described. For this purpose the usability u(P) of each design pattern container P (in phases 1 and 2) or design pattern P (in phase 3) is calculated as:

$$u(P) = \frac{\sum_{q_i \in Q^A} w_q(P,q_i) \cdot w_a(P,q_i,k)}{\sum_{q_i \in Q^A} w_q(P,q_i)} \qquad \textbf{(1)}$$

where

P	is a design pattern or pattern container (category),
$w_q(P,q_i)$	is the weight of a question q_i regarding the given P
$w_a(P,q_i,k)$	is the weight of the obtained answer k for the question q_i regarding the given P
Q^A	is the set of already answered questions

As we can see from the (Eq. 1), the same function is used to determine the usability of a design pattern or a pattern container. This is possible because the same mechanism of questions/answers matrices are used throughout the advisement process.

In the following text, we will use the term “design pattern” to explain the algorithm; please note that the same procedure is also used to identify and select the pattern containers.

The usability u(P) of each design pattern P is 0 when no question has been answered. After a user answers a question q_i, the usability u(P) of each pattern P changes regarding the values of the weight for this question ($w_a(P,q_i)$) and weight for the given answer ($w_a(P,q_i,k)$); the weights are pre-determined by experts. A gain for a pattern P, obtained by answering the question q_i with answer k is calculated as:

$$gain(P,q_i,k) = \frac{w_q(P,q_i) \cdot w_a(p,q_i,k)}{\sum_{q_j \in \{Q^A + q_i\}} w_q(P,q_j)} \qquad \textbf{(2)}$$

To clarify this concept, let us consider the following example: there are three possible patterns P_1, P_2, and P_3, from which we want to select the most appropriate one. From the advisement matrix the question q_n is chosen as the first question to be asked with the weight value $w_q(P,q_n)=0.3$. There are two possible answers (k=1 or k=2) available for the question q_n with the following weight values:

– When choosing the first answer k=1, the weights are $w_a(P_1,q_n,1)=-0.3$, $w_a(P_2,q_n,1)=0.0$, $w_a(P_3,q_n,1)=0.8$.
– When choosing the second answer k=2, the weights are $w_a(P_1,q_n,2)=0.5$, $w_a(P_2,q_n,2)=0.7$, $w_a(P_3,q_n,2)=0.0$.

Let us now presume that the user chooses the second answer (k=2). Using (Eq. 2) the gains for patterns P_1 through P_3 are the following (note that no questions have been answered yet): $gain(P_1,q_n,2)=0.5$, $gain(P_2,q_n,2)=0.7$, $gain(P_3,q_n,2)=0.0$.

For the chosen question q_n and the given answer k=2, the pattern P_2 gains the most and is temporarily (after answering only one question q_n) considered to be the most appropriate design pattern. Naturally, the design pattern having the highest usability score u(P) (Eq. 1) is ultimately selected after obtaining enough answers to the given questions.

The weight values for questions range from 0 to 1, meaning:

– If the weight value for a question is 0, the answer will have no effect on the selection of design patterns (unimportant question),
– If the weight value for a question is 1, the answer will have the maximum effect on the selection of the design pattern (the most important question), and
– Weights between 0 and 1 determine the importance of the question – the higher the weight value, the more important the question.

The weight values for the answers also range from -1 to 1, meaning:

– If the weight value of an answer is less than zero, choosing this answer will include some penalty for a design pattern,
– If the weight value of the answer is 0, choosing this answer will not change the usability of the design pattern, and if the weight value of the answer is greater than zero, choosing this answer will add something to the usability of the design pattern.

4.2. Dynamic Question Selection

Since the weights for questions and answers are pre-determined by experts, the order of choosing questions and/or assessment matrices is determined by the selected strategy and goals of the advisement process. If we want to identify and suggest an appropriate design pattern as soon as possible (with a minimum number of questions), a question should be chosen that maximizes the difference in usability functions for the most appropriate design pattern Pfirst and the second most appropriate design pattern Psecond. This can be determined by calculating specific gains for each of the patterns and all the remaining (unanswered) questions. As, naturally, we do not know which answer will be given by a user, we have decided to use the min-max algorithm, well known from game playing, for selecting the next question; it has been implemented in the prototype tool, used to obtain the results in the experiment performed (see section 5). The algorithm is presented in Figure 5.

```
findTheMostRelevantQuestion(Matrix m)
create a list L of all unasked questions from a matrix m
if notEmpty(L)
  foreach design pattern Pi
    foreach question q in list L
     calculate gain(Pi,q,k) for each possible ans. k according to (Eq. 2)
     determine minimal gain(Pi,q,k)
    determine maximal of all mininal gains
    select question q where the mininal gain(Pi,q,k) is maximal
else
  find next matrix or solution
```

Fig. 5. OQBA pseudo code algorithm for choosing the most relevant question

4.3. The Basic Design Pattern Selection Algorithm

The proposed ontology and question-based design pattern advisement approach is based on the proposition that the developers can adequately describe their specific situations following an interactive question/answer session. From their answers, enough information should be obtained to automatically identify and suggest appropriate design patterns. Based on the set of already answered questions, a level of usability is calculated that determines the appropriateness of each specific design pattern or pattern container. This set of answered questions is also used in combination with the remaining questions to determine the next most relevant question, until the usability of the most appropriate design pattern is dominant over all the other patterns. The overall algorithm that can be used for both the solicitation of pattern containers (phases 1 and 2) and for the selection of design patterns (phase 3) is presented in

Figure 6. The process of identifying appropriate candidate solutions (pattern containers or design patterns) is presented. The integral part of the algorithm (line 5) is also the procedure for finding the most relevant question described above.

```
QBA algorithm(AdviceArea aa)
1  for a given AdviceArea aa
2    ma = getMatrix(aa)
3    Qa = {}   ; set of already answered questions
4    do
5      q = findMostRelevantQuestion(aa)
6      a = getAnswer(q)
7      Qa = Qa + q
8      foreach pattern P in aa
9        calculate u(P) using (Eq. 1)
10       calculate remaining gain(P) using (Eq. 2)
11       ;the remaining gain is the sum of max possible
12       ;gains of all the remaining questions until
13       ;remaining gain (P) < u(Pfirst)-u(Psecond)
14       ;if the dominant pattern cannot be changed
15       ;with the remaining questions
16   if isContainer(Pfirst)
17     ma=nextMatrix(Pfirst)
18   else
19     displaySolution(Pfirst)
```

Fig. 6. OQBA pseudo code algorithm for choosing the most appropriate pattern container/design pattern

5. Concept Verification: The Experiment

In order to evaluate the efficiency of the proposed OQBA approach, a controlled experiment was designed and conducted. Two surveys were also taken – one before and one after the experiment. This section presents methodology, results and discussion on results.

5.1. Experiment Introduction and Methodology

The experiment was aimed at confirming or rejecting the following propositions:

– (P1) The OQBA approach contributes to more successful design pattern identification in comparison to manual identification.
– (P2) Less experienced developers benefit the most from using the OQBA approach in identifying the appropriate design patterns.

The experiment was carried out in four phases: in the first phase, participants had to answer a few questions concerning their development background and level of expertise (see Table 2). In the second phase, they were given the opportunity to solve a set of design problems manually, without the DPEX tool (but they were allowed to use other instruments such as books and established search facilities). In the third phase, the DPEX tool was provided to help them solve the same design problems. The fourth phase was aimed at gathering participants' opinions and reflections on the tool. For this purpose a post-experiment survey was conducted.

Two groups of developers were involved in the experiment. The basic profile of the participants and differences between groups are summarized in Table 2. The results of this self-assessment on GoF patterns knowledge is given in the last row in Table 2.

Table 2. Group profiles – used in experiment

Group no.	I	II
No. of participants (profile)	10 (software engineers / developers)	11 (students)
Formal training in GoF design patterns	None	All – one year ago
Active in production software development	less than 3 years: 0 3-6 years: 5 7-9 years: 3 10 years or more: 2	less than 3 years:4 3-6 years:6 7-9 years:1
Knowledge of GoF design patterns (self-assessment)	Weak: 1 Good: 5 Very good: 2 Excellent: 2	Weak: 4 Good: 4 Very good: 3 Excellent: none
How often they use design patterns	Never: 2 Rarely: 7 Often: 1	Never: 3 Rarely: 7 Often: 1

Each participant was given the descriptions for nineteen problem situations in which a particular GoF design pattern might represent a suitable solution. The participants were asked to propose the most appropriate design pattern to solve each of the given design problems. Design problems were chosen to represent real-life situations, not academic ones.

When preparing problem descriptions we intentionally selected problem situations with higher complexity and as we will describe later, some were even too complex. We wanted to evaluate the approach for situations that are similar to those that developers are faced with during their daily work.

Using the tool, they needed on average 40 minutes to complete all 19 tasks whereas an average completion time without the tool was 65 minutes (55 minutes for the fastest participant, 80 minutes for the slowest). Undoubtedly this difference is not caused by the tool, but due to the fact that they were familiar with the problems since they had been solving them in the previous stage of the experiment.

5.2. Results of The Experiment

We can observe that using DPEX tool, on average developers in the first group identified the appropriate design pattern in 58% of all cases. The approach and tool helped achieve more than 50 percent more correct solutions to design problems (40% without the tool).

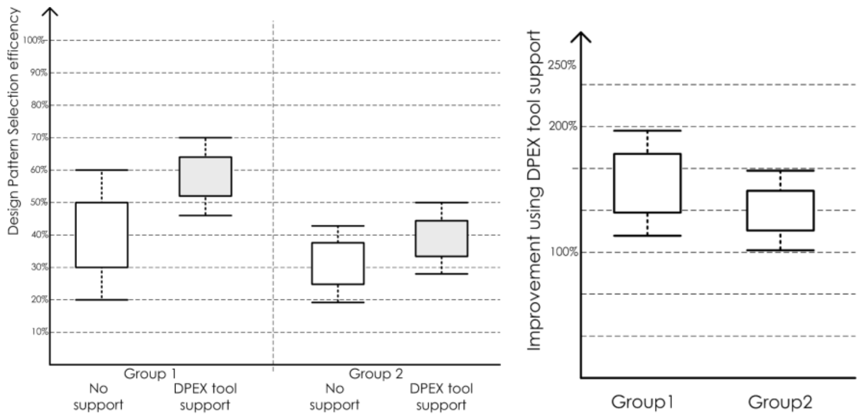

Fig. 7. Efficiency and improvement factors on solving problems with and without the DPEX tool – Group 1 and Group 2

Using the paired t-test we determined that for Group I the difference between the number of correct solutions found with and without the DPEX tool 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 tool to be less efficient than searching for an appropriate solution without the tool.

Only in four cases did the developer not accept and agree with the solution advised by the tool. In general the tool increased the efficiency of less experienced developers, and also of experts. The information on standard deviation for successfully solved problems with and without the tool confirms that by using the tool, less experienced developers manage to achieve results that are closer to those demonstrated by experts (Figure 7). After all, this was one of our main goals: to bridge the gap between the pattern expert community and the typical pattern user.

The use of the tool also resulted in improvements in the second experimental student group. The average improvement factor was 1.33. A paired t-test for Group 2 confirmed that there is a significant difference between the results achieved with and without the tool (P=0.001575).

The post-experiment survey showed that three developers believed it was easier to search for a solution using a tool, while four participants were convinced that it was easier to work without the tool. Four developers found the tool easy to use and beneficial to them.

The efficiency of both groups have improved significantly using the DPEX tool (group 1 by factor 1.63 and group 2 by factor 1.33). We can observe that the improvement factor for Group 1 is surprisingly higher. The discussion of the results follows in next section.

5.3. Discussion and Future Work

As shown by the experiment, most developers were able to solve more design problems by using DPEX tool. This shows that our approach assure better achievements regarding design pattern use. Let us discuss the propositions.

– P1: The OQBA approach contributes to a more successful design pattern identification in comparison to manual identification. P1 was confirmed since results confirmed that the use of tool and implemented approach improved the achievements of developers. We can detect the improvements in both groups.
– P2: Less experienced developers benefit the most from using the OQBA approach in identifying the appropriate design patterns. Proposition is not confirmed with the experiment. Group I average improvement factor is 1.63, while average improvement factor for Group II is 1.33.

It is a surprise that improvement factor is higher in Group I (experts). The reason might be in experiences, that professional developers have in terms of understanding questions and giving appropriate answers. However, interesting observation could be identified while comparing standard deviations for both groups. Decreased standard deviation show that the design pattern selection efficiency is becoming more equal using the DPEX tool. Less-experienced developers are obviously becoming more similar to experienced developers.

Additional analysis will be necessary (pre and post-experiment survey, log files) in order to improve the advisement on selecting a suitable pattern. It is also our aim to upgrade the approach with learning capabilities to ask personalized questions.

As future work, we plan to develop a holistic methodology for design pattern selection including automatic question forming based on the analysis of paths (recorded in logs) taken by developers interacting with the system during the question-answer session. We also plan to develop/use ontology based reasoning and techniques in order to automatically create relevant questions and answers based on information, concepts and relationships stored in the ontology. Nevertheless, the proposed Ontology and Question Based Design Patterns Advisement Approach might contribute to our common goal: to bridge the identified gap between pattern experts and the typical pattern user from the software community. For that purpose we also plan to join initiatives such as [23] aimed at resolving this challenge by networking, sharing ideas and joining resources.

6. Conclusion

The potential of using patterns has not yet been fully realized. Many challenges and issues still remain to be solved. Finding a suitable design pattern for a given situation obviously represents a great challenge for a typical developer. Tools assisting in this process have become of the utmost importance [11]. There are some facilities and approaches based on pattern review, browsing and full-text search. They might be helpful for the pattern expert community, but not for less-experienced developers, who are unaware as to which patterns exist in their work domains. The novel ontology and question-based approach presented in this paper was aimed at improving the use of design patterns. The proposed OQBA approach and corresponding DPEX tool assist software developers in choosing design patterns suitable for a given problem. The main contributions of the conducted research were:

– Definition of the Question and Ontology-Based Design Pattern Advisement approach.
– Extension of existing ODOL ontology with concepts needed to capture additional knowledge on patterns.
– Definition of the DPAO ontology that is used to provide knowledge for applying the OQBA approach.

Using the defined ontological concepts, information and knowledge on design patterns could be shared and automatically analyzed. We also provided the semantic based description of GoF patterns not available in the WoP project. The DPEX tool was developed and a controlled experiment was conducted. The main aim of the experiment was to explore if the DPEX tool-enabled and OQBA approach for selecting an appropriate pattern would do better than a manual selection. The results of the controlled experiment show that the developers were essentially able to solve more problem cases when the tool was available. The proposed approach also provided promising results with regard to the use of design patterns in the community of less-experienced developers.

Introducing the concepts and technologies of the semantic web into the field of design-pattern research creates new possibilities for making design patterns more approachable to software engineers. One could envisage a global knowledge database on design patterns, which would be specified using the ODOL ontology and other derived ontologies, such as DPAO ontology. This knowledge base could then be accessible through some form of a service-oriented architecture and available to various software development tools. This would allow the integration of facilities of a global knowledge base into Web-based knowledge platform on design patterns as defined in [4]. At the time of publishing the results we are extending and improving both approach and tools. Because of the platforms ability to insert new knowledge easily we are also introducing it to new fields, which include selecting design patterns in other domains (e.g. Service Oriented Architecture) or as a helper in expert systems for selecting resources (e.g. e-services).

References

1. L. Rising, Understanding the Power of Abstraction in Patterns, IEEE Software, 24 (4) (2007) 46-51.
2. Schmidt, D.C., Using Design Patterns to Develop Reusable Object-Oriented Communication Software, Communications of the ACM, 38 (10) (1995) 65-74.
3. Microsoft Patterns & Practices Group, http://msdn.microsoft.com/practices

4. D.Manolescu, W. Kozaczynski, A. Miller, J. Hogg, The Growing Divide in the Patterns World, IEEE Software, 24 (4) (2007), 61-67.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1998.
6. L. Rising, The Pattern Almanac 2000, Addison Wesley, 2000.
7. F. J. Budinsky, M. A. Finnie, J. M. Vlissides, P. S. Yu, Automatic Code Generation from Design Patterns, IBM Systems Journal, 36 (2)(1996) 151-171.
8. J. Dietrich, C. Elgar, The Web of Patterns Project, http://www-ist.massey.ac.nz/wop
9. A. H. Eden, A. Yehudai, J. Gil, Precise Specification and Automatic Application of Design Patterns, 12th IEEE International Conference on Automated Software Engineering, IEEE Press, 1997, Proceedings, pp. 143-152.
10. 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.
11. A. Birukuo, E. Blanzieri, P. Giorgini, Choosing the Right Design Pattern: The Implicit Culture Approach, Arturo Hinojosa, A Cognitive Model of Design Pattern Selection. Technical Report DIT-06-007, Informatica e Telecomunicazioni, University of Trento, 2006.
12. ODOL- Object-Oriented Design Ontology, http://svn.sourceforge.net/viewvc/webofpatterns/wop-patterndefinitions/20060324/ wop.owl
13. J. M. Rosengard, M. F. Ursu, Ontological Representations of Software Patterns, Lecture Notes in Computer Science (Proceedings of the of KES’04), 3215 (2004), pp. 31-37.
14. M. Fontoura, C. Lucena , Extending UML to Improve the Representation of Design Patterns, Journal of Object-Oriented Programming, 13(11) (2001) 12-19.
15. G. Sunyé, A.L. Guennec, J-M Jézéquel, Design Patterns Application in UML, Lecture Notes in Computer Science (ECOOP'2000 proceedings), 1850 (2000), pp. 44-62.
16. R. B. France, D. K. Kim, S. Ghosh and E. Song. A UML-Based Pattern Specification Technique, IEEE Transactions on Software Engineering, 30(3)(2004), 193-206.
17. T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American, 284 (5) (2001) 28-37.
18. W3C Semantic Web Activity, http://www.w3.org/2001/sw/
19. M. Safiz, P. Adamczyk, R.E. Johnson, Organizing Security Patterns, IEEE Software, 24 (4) 52-60.
20. M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003.
21. C. Alexander et al, A Pattern Language, Oxford University Press, New York, 1977.
22. Kung, D. C., H. Bhambhani, R. Shah, and G. Pancholi. 2003. An expert system for suggesting design patterns: a methodology and a prototype. In Software Engineering With Computational Intelligence, ed. T. M. Khoshgoftaar. Kluwer Int. Lazovik, A., M. Aiello, and M. Papazoglou. 2006.
23. M. Weiss, A. Birukou, P. Giorgini, EuroPLoP 2007 Focus Group on Pattern Repositories, http://www.patternforge.net/wiki/index.php?title=EuroPLoP 2007 Focus Group.

Luka Pavlič received his Ph.D. degree in computer science in 2009 from the University of Maribor, Slovenia. He is currently a senior researcher with the Institute of Informatics, FERI, at the University of Maribor. His main research interests include all aspects of IS development, reuse in software engineering, object orientation, information system architecture, Java, UML and XML-related technologies, semantic technologies, intelligent systems, big data and nosql. He has appeared as an author and co-author in several peer-reviewed scientific journals. He has also presented his work at a number of international conferences. In addition, he has participated in many national and international research projects.

Vili Podgorelec is a professor of computer science at the University of Maribor, Slovenia. His main research interests include intelligent systems, semantic technologies, and medical informatics. He has been participating in many international research projects and is author of several journal papers on computational intelligence, software engineering and medical informatics. Dr. Podgorelec has worked as a visiting professor and researcher at several universities around the world, including University of Osaka, Japan, University of Nantes, France, University of La Laguna, Spain, University of Madeira, Portugal, and University of Applied Sciences Seinäjoko, Finland. He received several international awards and grants for his research activities.

Marjan Heričko is a full professor at the Institute of Informatics. He is the head of the Information systems laboratory and Deputy Head of the Institute of informatics. He received his PhD in Computer Science from University of Maribor in 1998. His main research interests include all aspects of information systems development, software and service engineering, agile methods, process frameworks, software metrics, functional size measurement, SOA, component-based development, object-orientation, software reuse and software patterns. Dr. Heričko has been a project or work co-ordinator in several applied projects, project or work co-ordinator in several international research projects and committee member and chair of several international conferences.

Received: August 24, 2013; Accepted: January 25, 2014