Maejo Int. J. Sci. Technol. **2016**, *10*(01), 113-126

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Full Paper

Recommending design patterns using task-based conceptual features

Nasith Laosen, Nuttapon Sanyawong and Ekawit Nantajeewarawat*

School of Information, Computer and Communication Technology, Sirindhorn International Institute of Technology, Thammasat University, Thailand

* Corresponding author, e-mail: ekawit@siit.tu.ac.th

Received: 13 October 2015 / Accepted: 25 March 2016 / Published: 21 April 2016

Abstract: Task-based conceptual features (TCFs) represent human knowledge concerning intentions and/or characteristics of design tasks to which a design pattern is applicable. They provide a bridge connecting the usage of a design pattern and the characteristics of a design problem. A method for recommending appropriate design patterns based on TCFs is presented. From grammatical relations between words generated from the textual description of an input design problem, problem keywords are extracted. The obtained problem keywords are matched with clue words of each TCF in order to construct a feature vector representing the input problem. Based on the similarity between the feature vector representing the problem and the TCF-based vector representing each design pattern, design patterns are ranked and recommended. The method is evaluated on a collection of 24 input design problems. The evaluation results show that when the first-level hypernyms and hyponyms obtained from the WordNet ontology are employed for word matching and an appropriate penalty score is assigned to them, design patterns recommended in the top three ranks include the correct design patterns for all 24 problems.

Keywords: design pattern, pattern recommendation, task-based conceptual feature

INTRODUCTION

Object-oriented design patterns are proven solutions to frequently occurring software design problems [1-5]. However, even for an experienced developer, selection of an appropriate design pattern is often difficult since it requires a large amount of knowledge concerning the usage, intent and applicability of many available design patterns. To facilitate design pattern selection, Hasheminejad and Jalili [6] presented a two-phase selection method. In their proposal each design pattern is represented by a vector of words appearing in the descriptions of the pattern taken from standard design pattern textbooks. In the first phase a binary classification model is constructed for each category of patterns, e.g. creational, structural and behavioural patterns. The obtained classifiers are then used for predicting the category of a given input design problem, which is encoded as a vector of words occurring in the textual description of the problem. In the second phase the word vector representing the design problem is compared with the word vectors that represent design patterns in the predicted category by using cosine similarity. Design patterns are then recommended based on the obtained similarity scores. The use of word vectors for pattern representations, however, has two major drawbacks. First, occurrences of words are low-level features that may not clearly express the true characteristics of a design pattern. Second, the description of a pattern in a source textbook is often verbose and contains many words that are not relevant to the usage and applicability of the pattern. As a result, an obtained similarity score is often distorted by these irrelevant words.

In this article the concept of task-based conceptual feature (TCF) is introduced as a high-level feature for representing an intention and/or characteristic of a design task, and an automatic design pattern recommendation framework is proposed by exploiting two types of expert knowledge centring around TCFs. The knowledge of the first type associates a design pattern with a set of TCFs that characterise its usage. The knowledge of the second type provides a collection of clue words that partially characterises each TCF. With these two types of expert knowledge, TCFs provide a bridge connecting the usage of a design pattern and the characteristics of a given input design problem. From the pattern usage descriptions in the pattern usage hierarchy proposed in our previous work [7], 28 TCFs are extracted. We restrict our attention in this article to 13 Gang-of-Four design patterns [1] in the usage category of ‘performing a domain-specific task other than object creation’, which is the largest and most complicated category in the usage hierarchy under consideration.

This work consists of two main parts. The first part is concerned with the construction of TCFs and their clue words. TCFs are associated with a design pattern by means of a feature vector representing the pattern. The second part presents a method for automatically recommending appropriate design patterns based on TCFs. Using the WordNet ontology [8], clue words of TCFs are matched with keywords extracted from a given input design problem in order to construct a feature vector representing the problem. By computing their cosine similarity, the feature vector representing the problem is compared with that representing each design pattern. Appropriate design patterns are then ranked based on the resulting similarity scores.

RELATED WORK

In addition to the two-phase selection method proposed by Hasheminejad and Jalili [6], some studies on design pattern recommendation have been reported. Kampffmeyer and Zschaler [9] proposed a knowledge-based approach to the design pattern recommendation. A design pattern intent ontology was constructed to capture the relationship between design patterns and problem concepts that could be addressed by them. In their approach when a design problem was given, a software designer manually extracted the characteristics of the problem and employed the design pattern intent ontology to retrieve an appropriate design pattern.

Kim and Khawand [10] employed the Role-based Metamodelling Language, which is a UML-based pattern specification language, to formally specify the problem context in which a design pattern could be applied. Static pattern specifications were used for capturing the structural properties of design patterns, and interaction pattern specifications were used for capturing object interaction behaviours suggested by the patterns. Although the design pattern recommendation was not directly discussed, their proposed formal specifications provided checkpoints for assessing the conformance of a design problem to a design pattern.

Bouassida et al. [11] proposed an interactive tool-set for recommending an appropriate design pattern. Semantic correspondences (e.g. equivalences, hyponyms and compositions) among element names in an input class diagram and the names of design patterns' participants given in the Gang-of-Four book [1] were determined. Based on the obtained correspondences, hand-crafted recommendation rules were used for finding and instantiating a suitable design pattern. A user may interact with the recommendation rules to provide additional information required for determining an appropriate pattern.

Sahly and Sallabi [12] presented a strategic method for recommending a suitable design pattern. By using a pre-defined questionnaire, users were classified into 3 levels, i.e. novices, advanced beginners and experts. A vector space model was applied to represent textual patterns' intents and a problem description specified in terms of queries. Based on cosine similarity measurement and pre-defined similarity thresholds, candidate patterns were suggested. When a suitable pattern candidate was not found, a formal concept analysis and case-based reasoning techniques were applied to augment the currently given queries by making use of previously known queries. When a suitable pattern was still not found, the level of the user was considered. For an expert user, for example, the input problem description was posted to a group of experts for further discussion.

Palma et al. [13] developed a design pattern recommendation system in which the knowledge about design patterns, e.g. intentions and applicabilities, was transformed into textual conditions for selecting an appropriate pattern. A user characterised his/her design problem by answering questions reformulated from such textual conditions. An answer was given in the forms of 'yes', 'do not know' or 'no', with a weight indicating the degree of user's confidence. Based on the obtained answers, a pattern with the highest total weight was recommended. The proposed recommendation system was evaluated by 8 subjects and an accuracy of 50% was reported.

Issaoui et al. [14] presented a semi-automatic approach to design pattern suggestion. By applying the WordNet ontology [8], class names and method names extracted from an input class diagram were semantically compared with the names of design patterns' participants given in the Gang-of-Four book [1]. Candidate design patterns were determined based on the obtained similarity scores. Pre-defined questions specifying the intentions of each candidate pattern were then reformulated by replacing the pattern's participant names with their corresponding extracted class/method names. A user was required to specify the characteristics of his/her design problem by answering the reformulated questions. Based on the obtained answers, an appropriate pattern was recommended.

All the studies mentioned above focused their attention on the Gang-of-Four design patterns [1]. No empirical evaluation was presented by Kampffmeyer and Zschaler [9], Kim and Khawand [10], Bouassida et al. [11], Sahly and Sallabi [12] or Issaoui et al. [14].

TCFs AND THEIR CLUE WORDS

To begin with, TCFs are introduced. They are associated with design patterns in terms of feature vectors representing design patterns called DPFVs. The construction of clue words of each TCF is described.

TCFs and Feature Vectors Representing Design Patterns

A *task-based conceptual feature* (*TCF*) describes an intention and/or a characteristic of a design task. From pattern usage descriptions in the pattern usage hierarchy proposed in our previous work [7], 28 TCFs, referred to as F1-F28, are extracted. They are shown in Table 1. Table 2 associates TCFs with design patterns. It provides expert knowledge describing each design pattern in terms of the characteristics of problems to which the pattern is applicable. For example, the first row in the right-hand side of Table 2 indicates that the Strategy pattern is used for solving a design problem that is characterised by the TCFs F12, F13 and F15.

Table 1. Task-based conceptual features (TCFs)

Feature	Description
F1	Working with grammar and text parsing
F2	Separating basic tasks from specific tasks
F3	Assigning specialised tasks to different task performers
F4	Accessing external resources (hard-disk, Internet, etc.)
F5	Reducing retrieval time from external resources / Data caching
F6	Working with a sequence of tasks
F7	Working with collaborative tasks
F8	Working with multi-layer tasks
F9	Handling task-chain alternatives
F10	Changing an algorithm flow
F11	Constantly calling tasks in a pre-defined sequence
F12	Selecting an algorithm depending on an environment
F13	Working with many alternative algorithms
F14	Choosing an appropriate algorithm on object creation
F15	Determining an algorithm at run-time
F16	Performing operations that depend on states of objects
F17	Changing algorithms based on the current computation state
F18	Storing object data
F19	Object data restoration
F20	Storing an object operation
F21	Keeping a list of already-done operations
F22	Working with an undo function
F23	Dissemination of information
F24	Working with communication among objects
F25	Notification of information change
F26	One-to-many object communication
F27	Centralised object communication
F28	Working with a list of operations for information exchange

Table 2. Associating TCFs with design patterns

Design pattern	TCF	Design pattern	TCF
Interpreter	F1	Strategy	F12, F13, F15
Template method	F2, F3	State	F12, F13, F16, F17
Proxy	F4, F5	Memento	F18, F19
Decorator	F6, F7, F8, F9	Command	F20, F21, F22
Chain of responsibility	F6, F7, F9, F10	Observer	F23, F24, F25, F26
Façade	F6, F11	Mediator	F24, F27, F28
Bridge	F12, F13, F14		

Based on Table 2, a *feature vector representing a design pattern* (*DPFV*) is constructed. A DPFV for a design pattern *DP* is a sequence $[v_1, v_2, \ldots, v_{28}]$, where, for each $i \in \{1, 2, \ldots, 28\}$, v_i = '1' if the TCF F_i is associated with *DP*, and v_i = '0' otherwise. For example, in the DPFV for the Strategy pattern, only the elements v_{12}, v_{13} and v_{15} are ‘1’ and all other elements of the vector are ‘0’. DPFVs are normalised as follows: the normalised vector of a DPFV $[v_1, v_2, \ldots, v_{28}]$ is the sequence $[nv_1, nv_2, \ldots, nv_{28}]$, where, for each $i \in \{1, 2, \ldots, 28\}$, $nv_i = v_i \div (v_1 + v_2 + \ldots + v_{28})$.

Constructing Clue Words of TCFs

A *clue word* of a TCF is a word or a phrase that partially characterises the TCF. Four linguistic types of clue words are considered, i.e. nouns, adjective-noun pairs, verbs and verb-noun pairs. Noun and adjective-noun-pair clue words represent entities that are involved with a TCF, while verb and verb-noun-pair clue words represent intentions to perform some activities or tasks related to a TCF.

Clue words are associated with a TCF by words/phrases from candidate words/phrases collected from sentences in predetermined sections of three object-oriented design pattern textbooks [1-3]. Three sections of the first book [1], i.e. ‘Intent’, ‘Motivation’ and ‘Applicability’, two sections of the second book [2], i.e. ‘Role’ and ‘Use’, and one section of the third book [3], i.e. ‘Description’, are used as the sources of candidate words/phrases. The collected candidate words/phrases are preprocessed by (1) removing stop words and (2) determining the base forms of the remaining words by using the WordNet stemmer [8]. For example, by these preprocessing steps, a candidate phrase ‘parsing a string’ is changed into ‘parse string’; the indefinite article ‘a’ is removed and ‘parsing’ is replaced with its base form (‘parse’).

Clue words are selected from candidate words and word pairs by a human expert. Candidate nouns and adjective-noun pairs that represent important entities related to a TCF and candidate verbs and verb-noun pairs that represent certain specific intentions of a TCF are selected as clue words. For example, ‘parse’ and ‘undo’ are verb clue words of TCFs F1 and F22 respectively, and ‘select algorithm’ is a verb-noun-pair clue word of TCF F12.

Scheme 1 shows the obtained clue words of TCF F1. Table 3 and Table 4 show the number of the obtained clue words of TCFs F1-F14 and F15-F28 respectively, classified by clue-word type.

grammar (noun)	syntax tree (noun)	interpret sentence (verb, noun)	parse tree (verb, noun)
language (noun)	language expression (noun)	interpret string (verb, noun)	parse string (verb, noun)
syntax (noun)	expression (noun)	interpret text (verb, noun)	parse text (verb, noun)
text (noun)	parse (verb)	interpret language (verb, noun)	express language (verb, noun)
string (noun)			

Scheme 1. Clue words of TCF F1

Table 3. Number of clue words of TCFs F1-F14

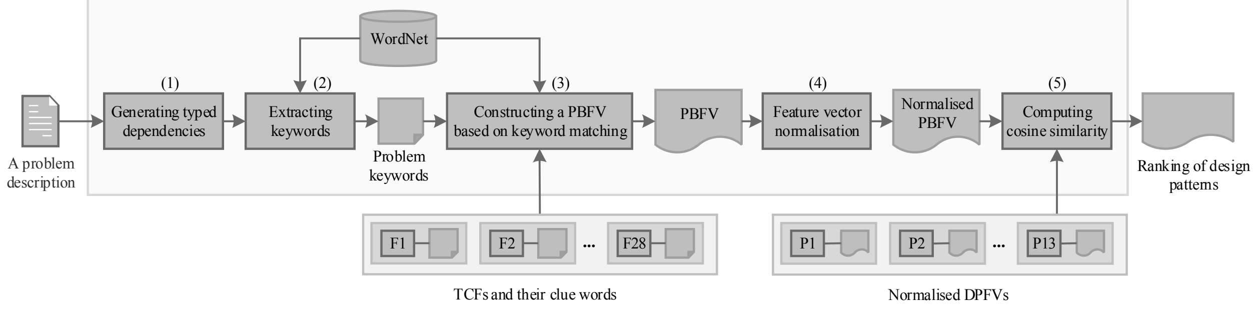
Clue-word type	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
Noun	8	8	8	2	7	6	5	8	8	8	5	4	2	4
Adjective-noun	0	56	31	3	3	8	10	8	3	2	0	2	6	0
Verb	1	0	0	0	1	0	0	0	0	0	0	0	0	0
Verb-noun	8	15	10	12	5	0	0	12	1	3	8	8	0	6

Table 4. Number of clue words of TCFs F15-F28

Clue-word type	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27	F28
Noun	5	2	7	4	4	2	7	4	1	4	2	3	5	4
Adjective-noun	2	4	2	0	0	0	1	0	0	0	0	0	1	0
Verb	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Verb-noun	16	1	5	4	6	6	10	4	7	2	9	2	2	2

METHOD FOR AUTOMATIC RECOMMENDATION OF DESIGN PATTERNS

A method for automatically recommending appropriate design patterns for a given design problem based on TCFs and their clue words is next described. As outlined in Scheme 2, the method consists of five steps: (1) generating typed dependencies (TDs) from the textual description of an input design problem, (2) extracting problem keywords from the obtained TDs, (3) matching problem keywords with clue words of each TCF in order to construct a feature vector representing the input problem called PBFV, (4) normalising the vector PBFV, and (5) computing cosine similarity scores between the normalised PBFV and normalised DPFVs for the 13 patterns under consideration. Design patterns are ranked according to the resulting similarity scores.


WordNet
(1)
Generating typed dependencies
(2)
Extracting keywords
(3)
Constructing a PBFV based on keyword matching
PBFV
(4)
Feature vector normalisation
Normalised PBFV
(5)
Computing cosine similarity
A problem description
Problem keywords
Ranking of design patterns
F1
F2
...
F28
P1
P2
...
P13
TCFs and their clue words
Normalised DPFVs

Scheme 2. Overview of proposed recommendation method

Generating TDs

TDs between words in a given textual problem description are generated by using the Stanford dependency parser [15]. A TD represents the grammatical relationship between a pair of words in a sentence. From 49 grammatical relationship types generated by the dependency parser, only 9 types that potentially provide the entities and intentions associated with a problem description are selected. The 9 selected types are adjectival modifier (*amod*), passive auxiliary (*auxpass*), direct object (*dobj*), indirect object (*iobj*), noun-compound modifier (*nn*), nominal subject (*nsubj*), passive-nominal subject (*nsubjpass*), object of a preposition (*pobj*) and open-clausal complement (*xcomp*).

Scheme 3 exemplifies the TDs generated from the sentence: 'We want to parse a Roman numeral string and convert it into an Arabic number'. The TD *dobj*(parse-4, string-8), for example, indicates that the noun 'string' acts as the direct object of the verb 'parse', and they occur as the 8th and 4th words respectively in the source sentence.

nsubj(want-2, We-1)	*det*(string-8, a-5)	*cc*(parse-4, and-9)	*det*(number-15, an-13)
root(ROOT-0, want-2)	*nn*(string-8, Roman-6)	*conj*(parse-4, convert-10)	*amod*(number-15, Arabic-14)
aux(parse-4, to-3)	*nn*(string-8, numeral-7)	*dobj*(convert-10, it-11)	*pobj*(into-12, number-15)
xcomp(want-2, parse-4)	*dobj*(parse-4, string-8)	*prep*(convert-10, into-12)	

Scheme 3. TDs generated from the sentence 'We want to parse a Roman numeral string and convert it into an Arabic number' using Stanford dependency parser

Extracting Problem Keywords

From the TDs generated from the description of a given problem, problem keywords are constructed as follows (KW_N, KW_{AN}, KW_V and KW_{VN} being a set of noun keywords, a set of adjective-noun-pair keywords, a set of verb keywords and a set of verb-noun-pair keywords respectively):

1. Initially, let each of KW_N, KW_{AN}, KW_V and KW_{VN} be empty.
2. Change each word occurring in each TD into its base form by using the WordNet stemmer.
3. For any m noun-compound TDs $nn(x_1, y_1)$, $nn(x_2, y_2)$, …, $nn(x_m, y_m)$, if $x_1, x_2, …, x_m$ are the same word occurrence and $y_1, y_2, …, y_m$ are adjacent word occurrences, then add to KW_N the result of concatenating $y_1, y_2, …, y_m, x_1$ with a space being inserted after each of y_i.
4. For each TD, e.g. τ, examine its form and construct keyword(s) as follows:
   a) If τ has the form $nn(x, y)$ and has not been used for keyword construction in Step 3, then add to KW_N the result of concatenating y, a space and x.
   b) If τ has one of the forms $dobj(x, y)$, $iobj(x, y)$, $nsubj(x, y)$, $nsubjpass(x, y)$ or $pobj(x, y)$, then add y to KW_N.
   c) If τ has the form $amod(x, y)$, then add x to KW_N and add to KW_{AN} the result of concatenating y, a space and x.
   d) If τ has the form $auxpass(x, y)$, then add x to KW_V.
   e) If τ has the form $xcomp(x, y)$, then add y to KW_V.
   f) If τ has one of the forms $dobj(x, y)$, $iobj(x, y)$ or $nsubjpass(x, y)$, then add x to KW_V and add to KW_{VN} the result of concatenating x, a space and y.
5. Remove each keyword that contains a pronoun and/or a stop word.
6. Remove duplicate keywords.

From the TDs in Scheme 3, for example, the above steps generate 3 noun keywords, i.e. ‘string’, ‘number’ and ‘Roman numeral string’; one adjective-noun-pair keyword, i.e. ‘Arabic number’; 2 verb keywords, i.e. ‘convert’ and ‘parse’; and one verb-noun-pair keyword, i.e. ‘parse string’.

Construction of Feature-Vectors-Represented Problems

A *feature vector representing a problem* (*PBFV*) is created based on matching keywords extracted from the description of the problem with clue words of TCFs. A PBFV for a problem *PB* is a sequence $[w_1, w_2, …, w_{28}]$, where, for each $i \in \{1, 2, …, 28\}$, w_i is computed by the *ComputeFeatureScore* procedure given in Scheme 4, with parameter ***KW*** being a set of all keywords of *PB* and parameter ***CW*** being a set of all clue words of TCF F_i. Words that are syntactically different may represent the same concept. In order to deal with the diversity of words, the WordNet ontology [8] is used for finding synonyms, hypernyms (broader terms) and hyponyms (narrower terms) of each component of a clue word.

The *ComputeFeatureScore* procedure uses parameter *NL* to limit the number of levels of hypernyms/hyponyms to be considered. As a level increases, the meanings of terms in that level are typically more different from an original word. In order to reflect the actual degree of relationship between keywords and clue words, a penalty score *PS* is taken as another parameter, and a linearly increasing value of *PS* is assigned to every level of hypernyms/hyponyms. In our experiment four values of *NL* were considered, i.e. 0, 1, 2 and 3. When $NL = 0$, neither hypernyms nor hyponyms were used and *PS* was set to zero. For each non-zero value of *NL*, varying values of *PS* such that $1 - (PS \times NL) > 0$ were used. For example, when $NL = 2$, each value in the set $\{0, 0.1, 0.2, 0.3, 0.4\}$ was used for *PS*.

```
procedure:  ComputeFeatureScore(KW, CW, NL, PS)
input:      KW: set of keywords of given problem    NL: maximum number of hypernym/hyponym levels
            CW: set of clue words of TCF           PS: penalty score
output:     FS: feature score
begin
    FS := 0
    for each keyword KW ∈ KW
        KW.isScored := false
    for each pair (KP, CP) ∈ KW × CW
        if ((KP.isScored = false) and (KP.type ∈ {'adjective-noun', 'verb-noun'}) and (KP.type = CP.type)) then
            begin
                S := PhraseMatch(KP, CP, NL, PS)
                if (S ≠ 0) then
                    begin
                        FS := FS + S
                        KP.isScored := true
                        if ((KP.type = 'adjective-noun') and (KP.nounWord ∈ KW)) then
                            remove KP.nounWord from KW
                        if ((KP.type = 'verb-noun') and (KP.verbWord ∈ KW)) then
                            remove KP.verbWord from KW
                    end
            end
    for each pair (K, C) ∈ KW × CW
        if ((K.isScored = false) and (K.type ∈ {'noun', 'verb'}) and (K.type = C.type)) then
            begin
                S := WordMatch(K, C, NL, PS)
                if (S ≠ 0) then
                    begin
                        FS := FS + S
                        K.isScored := true
                    end
            end
    return FS
end
```

Scheme 4. *ComputeFeatureScore* procedure

The *ComputeFeatureScore* procedure uses the *PhraseMatch* and *WordMatch* procedures for matching keywords with clue words. When matching is successful, an integer greater than zero is produced as a matching score. The *ComputeFeatureScore* procedure ensures that every keyword in ***KW*** is matched successfully with at most one clue word in ***CW***. Moreover, the noun component of an adjective-noun pair that has already been matched successfully will not be taken for further matching. Similarly, the verb component of a successfully matched verb-noun pair will not be matched again with any verb clue word.

The *PhraseMatch* procedure is given in Scheme 5. It is used for matching an adjective-noun-pair or verb-noun-pair keyword *KP* with a clue word *CP* of the same type. The output matching score is obtained by accumulating the scores resulting from matching their corresponding word components, which are calculated using the *WordMatch* procedure (Scheme 6). Among the four types of keywords, the verb-noun-pair keyword is most informative for expressing the intention of a given problem, and one extra point is added to the accumulated matching score for this keyword type.

```
procedure:  PhraseMatch(KP, CP, NL, PS)
input:      KP: problem keyword           NL: maximum number of hypernym/hyponym levels
            CP: clue word                 PS: penalty score
output:     MS: matching score
begin
    MS := 0
    if (KP.type = CP.type) then
        begin
            S1 := WordMatch(KP.firstWord, CP.firstWord, NL, PS)
            S2 := WordMatch(KP.secondWord, CP.secondWord, NL, PS)
            if ((S1 ≠ 0) and (S2 ≠ 0)) then
                begin
                    MS := S1 + S2
                    if (KP.type = 'verb-noun') then MS := MS + 1
                end
        end
    return MS
end
```

Scheme 5. *PhraseMatch* procedure

```
procedure:  WordMatch(K, C, NL, PS)
input:      K: component of problem keyword    NL: maximum number of hypernym/hyponym levels
            C: component of clue word          PS: penalty score
output:     MS: matching score
begin
    MS := 0
    if (K ∈ Synonyms(C)) then MS := 1
    else begin
        i := 1
        while ((i ≤ NL) and (MS = 0))
            begin
                if (K ∈ Hypernyms(C, i)) or (K ∈ Hyponyms(C, i)) then MS := 1 – (PS × i)
                i := i + 1
            end
    end
    return MS
end
```

Scheme 6. *WordMatch* procedure

For matching a single word component K of a keyword with a single word component C of a clue word, the *WordMatch* procedure in Scheme 6 is used, where *Synonyms*(C), *Hypernyms*(C, i) and *Hyponyms*(C, i) are the sets of synonyms, i^{th}-level broader terms and i^{th}-level narrower terms respectively of C, which are obtained from WordNet ontology using MIT Java WordNet Interface API [16]. When K is matched with a term in *Synonyms*(C), the matching score of 1 is produced. When K is matched with a term in *Hypernyms*(C, i) or *Hyponyms*(C, i), where $i \leq NL$, the resulting matching score is $1 – (PS \times i)$.

To illustrate phrase matching and word matching, suppose that:

- ***KW*** contains a verb-noun-pair keyword 'decide algorithm', referred to as *KP*, and this keyword is obtained from the direct-object TD *dobj*('decide', 'algorithm').
- ***CW*** contains a verb-noun-pair clue word 'select algorithm', referred to as *CP*.
- The parameters *NL* and *PS* are set to 2 and 0.4 respectively.

From the WordNet ontology, *Hypernyms*('select', 1) contains the word 'decide' and *Synonyms*('algorithm') contains the word 'algorithm' itself. The output score resulting from matching *KP* with *CP* using *PhraseMatch* is calculated as follows: (1) by calling *WordMatch*('decide', 'select', 2, 0.4), the score 1 – (0.4 × 1) = 0.6 is obtained, (2) by calling *WordMatch*('algorithm', 'algorithm', 2, 0.4), the score 1 is obtained, and (3) since *KP* is a verb-noun-pair keyword, the final matching score computed by *PhraseMatch*(*KP*, *CP*, 2, 0.4) is (0.6 + 1) + 1 = 2.6. Note that since *KP* is constructed from a TD of type *dobj*, its verb component, i.e. 'decide', is also stored as a verb keyword in ***KW*** by our keyword extraction process (cf. Step 4f of the procedure for extracting problem keywords). After *KP* is successfully matched, the *ComputeFeatureScore* procedure also removes the verb keyword 'decide' from ***KW***.

Normalising Feature Vectors and Computing Cosine Similarity

In order to determine its similarity to a DPFV for each design pattern without biased values, the obtained PBFV is also normalised. The normalised vector of a PBFV [w_1, w_2, …, w_{28}] is the vector [nw_1, nw_2, …, nw_{28}], where, for each $i \in \{1, 2, …, 28\}$, $nw_i = w_i \div (w_1 + w_2 + … + w_{28})$. The cosine similarity score between the PBFV representing an input problem and the DPFV for each design pattern is computed by

$$\left[\sum_{i=1}^{n}(NPBFV_i \times NDPFV_i)\right] \div \left[\sqrt{\sum_{i=1}^{n}(NPBFV_i)^2} \times \sqrt{\sum_{i=1}^{n}(NDPFV_i)^2}\right],$$

where *NPBFV* is the normalised vector of PBFV, *NDPFV* is the normalised vector of DPFV, and *n* is the total number of TCFs. The design patterns are then ranked and recommended according to the resulting similarity scores.

EXPERIMENTAL RESULTS

In order to evaluate the proposed method, we collected 24 design problems, referred to as Q1-Q24, from three design pattern books authored by Kuchana [3], Cooper [4] and Lasater [5], and from the Internet. Scheme 7 shows an example of design problems taken from the third book [5], according to which the Strategy pattern should be applied to this problem. An experiment was conducted using 20 combinations of the *NL* and *PS* parameters (the maximum number of levels of hypernyms/hyponyms and the penalty score), i.e. 1, 10, 5 and 4 combinations with *NL* being 0, 1, 2 and 3 respectively (cf. the description of *ComputeFeatureScore* procedure).

We have a set of classes, each of which has arithmetic code that returns a value based on the type of arithmetic we wish to perform. Each class is slightly different, but all take a common input of a value and a variance. We have a class for addition. We have a class that performs subtraction on the value and variance. We allow a user to decide an algorithm to use at any given time. Right now we have if-then-else code to do this, but it is not very efficient. If we want to add a class to perform multiplication and a class to perform division, we would have to modify the if-then-else code for each algorithm. We need a better and more flexible way to add classes and manage the algorithmic grouping. We need to use a common interface that ties all these classes together as well.

Scheme 7. Example of design problems

From the experimental results, the combination in which *NL* = 1 and *PS* = 0.8 yielded the most accurate recommendations. Table 5 shows the recommended patterns in the top three ranks obtained by applying this combination to the problems Q1-Q24. The correct recommended patterns are emphasised using bold alphabets. The column 'Actual answer' gives the correct pattern for the problem under consideration, and the columns '1st Rank', '2nd Rank' and '3rd Rank' show the patterns with highest, second highest and third highest similarity scores respectively, obtained by the proposed method. For example, the second row of Table 5 indicates that the State pattern has the highest similarity score (0.78), and the Bridge and Strategy patterns have the second highest similarity score (0.15). Since there are two recommended patterns in the 2nd rank, the 3rd rank is empty. From the 24 problems used in the evaluation, 14 actual answers were obtained as recommended patterns in the 1st rank, 8 actual answers as recommended patterns in the 2nd rank, and 2 actual answers as recommended patterns in the 3rd rank.

Table 5. Recommended patterns in the top three ranks when *NL* = 1 and *PS* = 0.8 for 24 design problems (Q1-Q24)

Question	Actual answer	Recommended pattern		
		1st Rank	2nd Rank	3rd Rank
Q1	Strategy	Template M. (0.48)	**Strategy** (0.43)	Decorator (0.40)
Q2	State	**State** (0.78)	Bridge, Strategy (0.15)	
Q3	Decorator	Observer (0.54)	Mediator (0.53)	**Decorator** (0.46)
Q4	CoR	Observer (0.58)	**CoR** (0.53)	Decorator (0.46)
Q5	Command	**Command** (0.64)	Decorator (0.48)	CoR (0.39)
Q6	Observer	**Observer** (0.95)	Mediator (0.29)	State (0.19)
Q7	Proxy	**Proxy** (0.73)	Memento (0.33)	Mediator (0.33)
Q8	Template M.	**Template M.** (0.62)	Decorator, CoR (0.32)	
Q9	Façade	**Façade** (0.59)	Memento (0.41)	Decorator, CoR (0.35)
Q10	Bridge	**Bridge**, Strategy (0.54)		Template M. (0.42)
Q11	Mediator	Observer (0.85)	**Mediator** (0.54)	Proxy, Memento (0.18)
Q12	Memento	**Memento** (0.52)	Command (0.50)	Proxy (0.25)
Q13	Interpreter	**Interpreter** (0.97)	Decorator, CoR (0.15)	
Q14	Interpreter	Mediator (0.67)	**Interpreter** (0.58)	Observer (0.41)
Q15	Template M.	**Template M.** (0.89)	Façade (0.23)	Strategy (0.18)
Q16	Decorator	**Decorator** (0.42)	Command (0.38)	Proxy (0.20)
Q17	CoR	Decorator (0.59)	**CoR** (0.51)	Command (0.36)
Q18	Façade	Decorator (0.65)	CoR (0.58)	**Façade** (0.44)
Q19	Strategy	Decorator (0.54)	**Strategy**, Bridge (0.49)	
Q20	State	**State** (0.50)	Mediator (0.31)	Façade (0.21)
Q21	Memento	**Memento** (0.69)	Proxy (0.27)	Observer (0.15)
Q22	Command	**Command** (0.77)	Mediator (0.19)	Decorator, CoR, State, Observer (0.13)
Q23	Observer	State (0.41)	**Observer** (0.33)	Proxy (0.29)
Q24	Mediator	Observer (0.69)	**Mediator** (0.59)	Strategy (0.22)

Table 6 shows the accuracy of all combinations of *NL* and *PS*. For example, the row in which *NL* = 1 and *PS* = 0.8 shows that:

- The actual answers of 58.33% (14/24) of the problems are recommended in the 1st rank.
- The actual answers of 91.67% (22/24) of the problems belong to the top two ranks (1st rank or 2nd rank).
- The actual answers of 100.00% (24/24) of the problems belong to the top three ranks (1st rank, 2nd rank or 3rd rank).

Table 6. Overall experimental results

Parameter		**Percentage of correct answers**				
NL	***PS***	**Top (1st) rank**	**2nd rank**	**3rd rank**	**4th rank**	**5th rank**
0	0	58.33% (14/24)	79.17% (19/24)	91.67% (22/24)	91.67% (22/24)	91.67% (22/24)
1	0	54.17% (13/24)	75.00% (18/24)	87.50% (21/24)	**100.00% (24/24)**	100.00% (24/24)
1	0.1	54.17% (13/24)	75.00% (18/24)	87.50% (21/24)	**100.00% (24/24)**	100.00% (24/24)
1	0.2	54.17% (13/24)	75.00% (18/24)	91.67% (22/24)	**100.00% (24/24)**	100.00% (24/24)
1	0.3	58.33% (14/24)	79.17% (19/24)	91.67% (22/24)	**100.00% (24/24)**	100.00% (24/24)
1	0.4	58.33% (14/24)	83.33% (20/24)	95.83% (23/24)	**100.00% (24/24)**	100.00%(24/24)
1	0.5	58.33% (14/24)	87.50% (21/24)	**100.00% (24/24)**	100.00% (24/24)	100.00% (24/24)
1	0.6	58.33% (14/24)	87.50% (21/24)	**100.00% (24/24)**	100.00% (24/24)	100.00% (24/24)
1	0.7	54.17% (13/24)	91.67% (22/24)	**100.00% (24/24)**	100.00% (24/24)	100.00% (24/24)
1	0.8	58.33% (14/24)	91.67% (22/24)	**100.00% (24/24)**	100.00% (24/24)	100.00% (24/24)
1	0.9	58.33% (14/24)	87.50% (21/24)	**100.00% (24/24)**	100.00% (24/24)	100.00% (24/24)
2	0	50.00% (12/24)	62.50% (15/24)	70.83% (17/24)	79.17% (19/24)	91.67% (22/24)
2	0.1	50.00% (12/24)	62.50% (15/24)	70.83% (17/24)	83.33% (20/24)	91.67% (22/24)
2	0.2	50.00% (12/24)	66.67% (16/24)	75.00% (18/24)	91.67% (22/24)	91.67% (22/24)
2	0.3	50.00% (12/24)	66.67% (16/24)	91.67% (22/24)	95.83% (23/24)	95.83% (23/24)
2	0.4	54.17% (13/24)	79.17% (19/24)	91.67% (22/24)	95.83% (23/24)	95.83% (23/24)
3	0	33.33% (8/24)	54.17% (13/24)	70.83% (17/24)	79.17% (19/24)	79.17% (19/24)
3	0.1	45.83% (11/24)	58.33% (14/24)	70.83% (17/24)	83.33% (20/24)	83.33% (20/24)
3	0.2	45.83% (11/24)	66.67% (16/24)	70.83% (17/24)	83.33% (20/24)	83.33% (20/24)
3	0.3	50.00% (12/24)	62.50% (15/24)	87.50% (21/24)	91.67% (22/24)	95.83% (23/24)
Average		**52.71%**	**74.58%**	**87.29%**	**93.75%**	**95.00%**

Table 6 also reveals that only the combinations in which *NL* = 1 yield all actual answers within the top five recommended patterns. More specifically, the combinations in which *NL* = 1 and $0 \leq PS \leq 0.4$ give all actual answers within the top four recommended patterns, and the combinations in which *NL* = 1 and $0.5 \leq PS \leq 0.9$ give all actual answers within the top three recommended patterns.

CONCLUSIONS

Task-based conceptual features (TCFs) have been introduced as intermediate artifacts for representing expert knowledge concerning the usage of each design pattern. By matching clue words of TCFs with problem keywords extracted from the typed dependencies generated from the description of an input design problem, the similarity between the input problem and each design pattern is estimated. Appropriate design patterns are ranked according the obtained similarity scores. An evaluation using 24 design problems has shown that when the first-level hypernyms and hyponyms are used for keyword matching and a penalty score is between 0.5-0.9, the correct design pattern for each problem is one of the patterns recommended in the top three ranks.

ACKNOWLEDGEMENTS

This work was partially supported by (1) the National Research University Project of Thailand, Office of the Higher Education Commission and (2) the Centre of Excellence in Intelligent Informatics, Speech and Language Technology and Service Innovation (CILS), Thammasat University.

REFERENCES

1. E. Gamma, R. Helm, R. Johnson and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, Boston (MA), **1994**.
2. J. Bishop, “C# 3.0 Design Patterns”, O’Reilly Media, Sebastopol (CA), **2007**.
3. P. Kuchana, “Software Architecture Design Patterns in Java”, Auerbach Publications, Boca Raton (FL), **2004**.
4. J. W. Cooper, “Java Design Patterns: A Tutorial”, Addison-Wesley, Boston (MA), **2000**.
5. C. G. Lasater, “Design Patterns”, Jones and Bartlett Learning, Burlington (MA), **2006**.
6. S. M. H. Hasheminejad and S. Jalili, “Design patterns selection: An automatic two-phase method”, *J. Syst. Softw.*, **2012**, *85*, 408-424.
7. N. Sanyawong and E. Nantajeewarawat, “Design pattern recommendation based on a pattern usage hierarchy”, Proceedings of 18th International Computer Science and Engineering Conference, **2014**, Khon Kaen, Thailand, pp.134-139.
8. C. Fellbaum, “WordNet: An Electronic Lexical Database”, MIT Press, Cambridge (MA), **1998**.
9. H. Kampffmeyer and S. Zschaler, “Finding the pattern you need: The design pattern intent ontology”, in “Model Driven Engineering Languages and Systems” (Ed. G. Engels, B. Opdyke, D. C. Schmidt and F. Weil), Springer, Berlin, **2007**, Ch.15.
10. D.-K. Kim and C. E. Khawand, “An approach to precisely specifying the problem domain of design patterns”, *J. Vis. Lang. Comput.*, **2007**, *18*, 560-591.
11. N. Bouassida, A. Kouas and H. Ben-Abdallah, “A design pattern recommendation approach”, Proceedings of 2nd IEEE International Conference on Software Engineering and Service Science, **2011**, Beijing, China, pp.590-593.
12. E. M. Sahly and O. M. Sallabi, “Design pattern selection: A solution strategy method”, Proceedings of International Conference on Computer Systems and Industrial Informatics, **2012**, Sharjah, UAE, pp.1-6.
13. F. Palma, H. Farzin, Y. G. Guéhéneuc and N. Moha, “Recommendation system for design patterns in software development: An DPR overview”, Proceedings of 3rd International Workshop on Recommendation Systems for Software Engineering, **2012**, Zurich, Switzerland, pp.1-5.

14. I. Issaoui, N. Bouassida and H. Ben-Abdallah, "A new approach for interactive design pattern recommendation", *Lect. Notes Softw. Eng.*, **2015**, *3*, 173-178.
15. M.-C. de Marneffe, B. MacCartney and C. D. Manning, "Generating typed dependency parses from phrase structure parses", Proceedings of 5th International Conference on Language Resources and Evaluation, **2006**, Genoa, Italy, pp.449-454.
16. M. A. Finlayson, "Java libraries for accessing the Princeton WordNet: Comparison and evaluation", Proceedings of 7th International Global WordNet Conference, **2014**, Tartu, Estonia, pp.78-85.