Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2016), *30*, 406–423.
© Cambridge University Press 2016 0890-0604/16
doi:10.1017/S089006041600038X

Enhanced SPARQL-based design rationale retrieval

LUYE LI,[1,2] SHUMING GAO,[2] YING LIU,[3] AND XIAOLIAN QIN[2]

[1]Nanjing Research Institute of Electronics Technology, Nanjing, China
[2]State Key Laboratory of CAD and CG, Zhejiang University, Hangzhou, China
[3]Institute of Mechanical and Manufacturing Engineering, School of Engineering, Cardiff University, Cardiff, United Kingdom

(RECEIVED October 14, 2015; ACCEPTED May 31, 2016)

Abstract

Design rationale (DR) is an important category within design knowledge, and effective reuse of it depends on its successful retrieval. In this paper, an ontology-based DR retrieval approach is presented, which allows users to search by entering normal queries such as questions in natural language. First, an ontology-based semantic model of DR is developed based on the extended issue-based information system-based DR representation in order to effectively utilize the semantics embedded in DR, and a database of ontology-based DR is constructed, which supports SPARQL queries. Second, two SPARQL query generation methods are proposed. The first method generates initial SPARQL queries from natural language queries automatically using template matching, and the other generates initial SPARQL queries automatically from DR record-based queries. In addition, keyword extension and optimization is conducted to enhance the SPARQL-based retrieval. Third, a design rationale retrieval prototype system is implemented. The experimental results show the advantages of the proposed approach.

Keywords: Design Rationale Retrieval; Ontology; SPARQL Template; Text Search

1. INTRODUCTION

During the product design process, design engineers carry out various activities such as analyzing requirements, proposing and evaluating solutions, and making decisions. To complete these activities, it is often required to reuse knowledge and experience from previous proven designs. Design rationale (DR) contains most of this kind of design knowledge and experience, because it is an explanation of the product design process. DR includes all of the background knowledge, such as deliberating, reasoning, trade-off, and decision making in the design process of an artifact: information that can be valuable, or even critical, to various people who deal with the artifact (Regli et al., 2000). In recent years, more and more companies have become aware of the importance of DR retrieval, because effective reuse of DR depends on successful retrieval of relevant and useful DR information.

Research into how to capture, store, and retrieve DR has been ongoing for more than 40 years, and several tools have already emerged for DR retrieval. However, research on ontology-based DR retrieval is still in its infancy because formal query languages are required, posing two problems. First, formal query languages for ontology-based DR retrieval are difficult to use by end-users. Retrieving DR with rich semantics needs formal query languages, and formulating a query using such languages normally requires the knowledge of domain ontology as well as the syntax of the language (Kara et al., 2012). Second, formal query languages are not powerful enough for text searches. Today's formal query languages lack the sophisticated text search functionality that is required for the literals in DR ontologies. Without literals, an RDF (http://www.w3.org/TR/PR-rdf-syntax/) graph is just a set of interconnected nodes, one element out of a set of isomorphic graphs where nodes are practically meaningless (Minack et al., 2008).

SPARQL (http://www.w3.org/TR/rdf-sparql-query/) is a widely used formal query language for ontology, which can be used to retrieve and manipulate data stored in RDF format. In addition, SPARQL can powerfully express the intent of a user's query, and can manage a range of user input queries including keywords, natural language, and DR records. It is also well supported by mainstream ontology tools.

In this work, an ontology-based DR retrieval approach combining SPARQL and text search is presented, which aims to overcome the problems mentioned above. To retrieve the ontology-based DR information in an easy to use way, natural language queries are processed into SPARQL queries by template matching with domain knowledge, and DR record-based query is also processed into SPARQL query

Reprint requests to: Shuming Gao, State Key Laboratory of CAD and CG, Zhejiang University, Hangzhou, China. E-mail: smgao@cad.zju.edu.cn

by template matching or an automatic ontology-based SPARQL query generation method. To get more relevant retrieval results, keyword extension and optimization is conducted to improve and perfect the SPARQL queries. To enhance the retrieval precision, a DR ontology for DR retrieval and reuse is designed as the semantic model of the proposed extended issue-based information system (IBIS)-based DR representation. Based on this DR ontology, the captured DR records can be transformed to ontology individuals through an ontology population, and a DR database can be constructed containing the ontology information.

The rest of this paper is organized as follows. Section 2 presents the state of the art in DR representation and retrieval. Section 3 gives an overview of our approach. Section 4 details the semantic model of the extended IBIS-based DR representation. Our proposed methods of generating SPARQL queries from natural language queries and DR record-based queries are reported in Section 5 and Section 6, respectively. After that, implementation of a prototype system is given in Section 7, and Section 8 describes the experiments and evaluation results. Finally, Section 9 summarizes our proposed work and discusses future work.

2. RELATED WORKS

2.1. DR representation

A good representation schema is vital to enable effective design and reuse (Regli et al., 2000). Research on DR representation has been reported since the 1970s. Most of the DR representation approaches are argumentation-based approaches. The typical model is IBIS (Kunz & Rittel, 1970), which uses issues, positions, arguments, and the relationships between them to represent DR. Several software tools that allow engineering designers to record DR have been implemented based on IBIS. For example, Conklin and Begeman (1988) developed graphical IBIS, and Bracewell et al. (2009) implemented design rationale editor (DRed). In addition, McCall (1991) proposed the procedural hierarchy of issues model, which broadens the scope of the concept of "issue" in IBIS. Another argumentation-based model is question, option, and criteria (MacLean et al., 1991), which is a type of semiformal notation of design space analysis. Liu et al. (2010; Liang et al., 2012) proposed an issue, solution, and artifact layer model for DR representation and rationale information discovery from design archival documents. Fenves et al. (2008) presented a well-defined core product model that aims to capture product information shared throughout the whole product's life cycle. Liu and Hu (2013) proposed an intent-driven representation model to capture and formalize the DR and its evolving history to support DR reuse.

In addition, as semantic web technology has developed, several ontology-based representation schemas for DR information have been proposed. Burge and Brown (2008) developed a software-engineering-using-rationale system, which extends decision representation language with argument ontology. This argument ontology is a hierarchy of common arguments that serve as types of claims. De Medeiros and Schwabe (2008) proposed the Kuaba ontology, which extends the argumentation structure of IBIS by explicating the representation of the decisions made during design and their justifications, and the relationships between the argumentation and the generated artifacts. Based on the IBIS model, Zhang et al. (2013) have proposed an ontology-based semantic representation model for DR information, namely, the integrated issue, solution, artifact, and argument model, which introduces an ontology-based semantic representation mode to the DR representation mode of the DR representation and expands the conceptual elements of IBIS. To facilitate decision making within collaborative design, Rockwell et al. (2009) have developed a decision support ontology (DSO), which includes decision-related information including design issue, alternatives, evaluation, criteria, and preferences. It also includes decision rationale and assumptions, as well as any constraints created by the decision and the decision outcome. Although DSO, which includes more element types and relationships, can describe DR in a more explicit manner, it is not practical to capture such complex DR information, because the work required to capture this information may interrupt users' regular design work, and is likely to prevent users from capturing the DR.

2.2. DR retrieval

A good DR retrieval system should be able to provide comprehensive and precise search results for users in a convenient way. There have been several studies dedicated to DR retrieval in recent years. In general, DR retrieval works can be classified into two main categories depending on whether or not ontology is used: text-based retrieval and ontology-based retrieval.

Text-based retrieval does not use ontology, and it is easy to use. Liang et al. (2010) have proposed a DR search and retrieval system that focuses on interactive user interface design. Their system contains three basic functions: a view function, which enables engineering designers to intuitively navigate a DR repository; a search function, which supports designers in retrieving relevant DR from multiple aspects; and an analysis function, which suggests some useful DR insights. Kim et al. (2005, 2007) have presented two methods for the retrieval of DR captured using DRed. The first approach uses natural language processing techniques to annotate rationale records with nine selected semantic relationships (Kim et al., 2005). The second approach recommends relevant pieces of DR by analyzing the design task models of design reuse (Kim et al., 2007). Wang et al. also developed a keyword-based retrieval tool for DRed files (Wang et al., 2009), and then later proposed a new DR retrieval system that makes use of the implicit structures in DRed graphs (Wang et al., 2012). The general problem with text-based retrieval is that various DR records may include semantics such as types, relationships, and structure, which cannot easily be taken full advantage of using text-based retrieval, particularly for implicit semantics.

In comparison, ontology-based retrieval makes better use of the semantics embedded in DR records than text-based retrieval, and hence more comprehensive and more precise results are obtained from searches. Lim et al. (2010, 2011) have proposed an information search and retrieval framework based on a semantically annotated multifacet product family ontology, which exemplified how new product variants can be derived based on the designer's query of requirements via faceted search and retrieval of product family information. López et al. (2008) have presented NDR ontology to describe nonfunctional requirements and DR knowledge, and a multifacet search was implemented through executing SPARQL queries over the semantic catalogues of nonfunctional requirements. Zhu et al. (2010) have retrieved and inferred product data semantics with product engineering ontologies using SWRL (http://www.w3.org/Submission/SWRL/) and SQWRL (http://protege.cim3.net/cgi-bin/wiki.pl?CollectionsSQWRL). However, these approaches cannot easily be used because they require a relatively complex query language.

In our earlier work (Li et al., 2014), a DR retrieval approach using ontology-aided indexing was proposed that fully utilizes the semantics of DR in an easy to use way. However, its retrieved objects are not ontologies, and it requires specific index structure, which means that is not easily extensible. In addition, SPARQL queries are not supported. Although it is not very easy to use SPARQL as a formal query language, its expressivity for the user's query intent is more powerful than normal queries such as keyword and natural language, and it can also be used to express DR records. As a result, SPARQL-based retrieval is an effective way for design knowledge reuse.

To use SPARQL in a convenient way, Unger et al. (2012) have proposed a template-based question answering approach over RDF data, which translates questions into SPARQL queries. This approach relies on the parse of the question to produce a SPARQL template that directly mirrors the internal structure of the question, and then the template is instantiated using statistical entity identification and predicate detection. In addition, Minack et al. (2008) have presented LuceneSail, a combination of structured queries (SPARQL) with full-text search.

In summary, the research on DR retrieval is still in its infancy, and current approaches lack either semantics or convenience for users. Our work presents an ontology-based DR retrieval approach, which takes full advantage of the semantics and provides a new effective way to retrieve DR for industrial use.

3. OVERVIEW OF APPROACH

In this paper, we propose an approach for ontology-based DR retrieval shown in Figure 1. This approach supports normal user input query such as natural language query. As shown in Figure 1, DR records are stored as files according to the proposed DR representation. The DR ontology contains classes and properties, and the DR instance ontologies contain individuals and the relationships between individuals. The proposed DR retrieval approach aims to improve precision, recall, and convenience of the retrieval. Specifically, ontology is used to represent DR information in order to utilize more semantic information and obtain a higher retrieval precision. To get a higher retrieval recall, SPARQL query is enriched by extending and optimizing the keywords. To make the search more convenient for normal users, key-

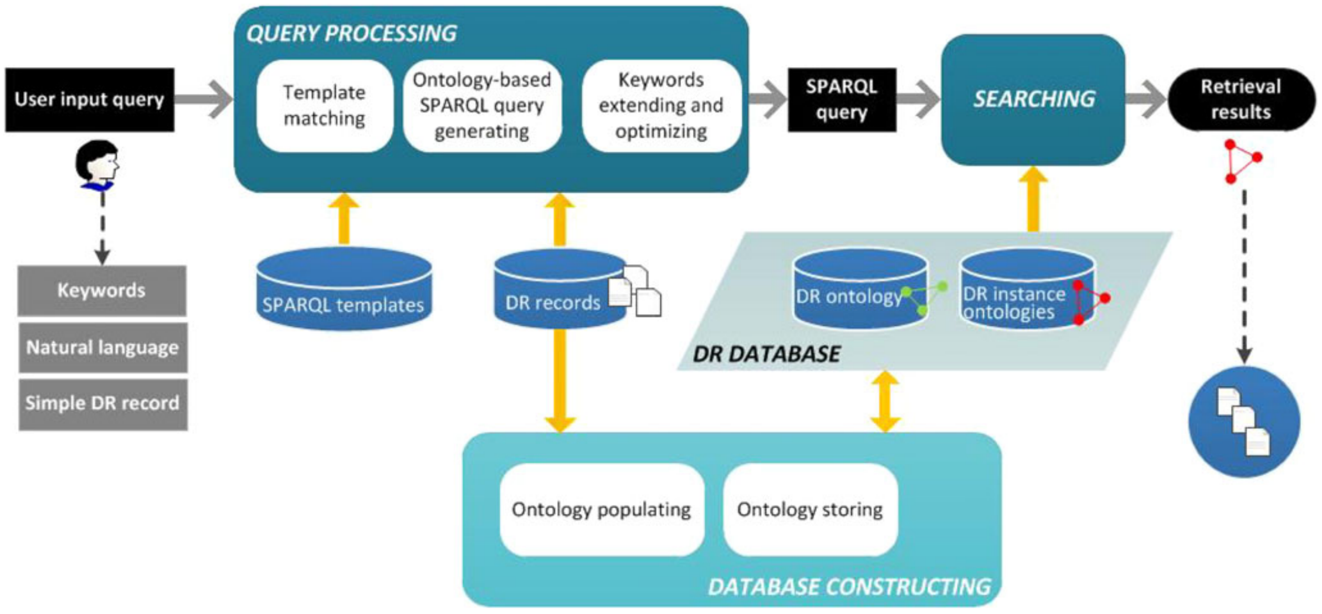


Fig. 1. Framework for ontology-based design rationale retrieval.

word-based query, natural language query, and DR record-based query are provided.

As shown in Figure 1, our approach contains three main components: query processing, database constructing, and searching. Using the SPARQL query and the DR database, the searching part just needs to execute the query and then obtain the results. We will now give a brief description of each of the remaining two components.

3.1. Query processing

Query processing starts when a user inputs a query and ends with the output of a SPARQL query. User input queries can include keywords, natural languages, and DR records. For further details on the use of these queries, please refer to Li et al. (2014). This processing uses domain knowledge of engineering design and sophisticated features of information retrieval (IR) techniques such as stemming and synonym expansion. There are three key steps to this processing: a SPARQL template is selected automatically according to the user input query; for a DR record-based query that does not correspond to an existing template, a corresponding SPARQL query will be automatically generated based on ontology; and a Lucene (http://lucene.apache.org/)-based keyword extension and optimization method is performed to fill the template with extended and optimized keywords. Stemming and synonym expansion are used to extend individual keywords into several keywords.

3.2. Database constructing

Database constructing handles every DR record and translates it into an ontology-based representation, which is then stored in the DR database. This prior processing minimizes the search operation, providing fluent human–computer interaction. This component contains three main steps: DR records are populated to corresponding instance ontologies based on DR ontology; instance ontologies are enriched using ontology reasoning; and all DR ontologies (including the DR ontology and the DR instance ontologies) are stored in a database that supports SPARQL query.

4. ONTOLOGY-BASED DR REPRESENTATION FOR DR RETRIEVAL

An ontology is an explicit specification of a conceptualization (Gruber, 1993), which formally represents knowledge as a set of concepts within a domain, and the relationships between pairs of concepts. In order to effectively make use of the semantics embedded in DR, a corresponding ontology that contains the domain knowledge of DR can be designed to help representing DR. In this work, we develop a DR ontology based on an extended IBIS-based DR representation for DR retrieval, and the DR ontology is essentially the semantic model of the proposed DR representation that mainly adopts the relevant concepts defined in the IBIS model (Kunz & Rittel, 1970), issue, solution, and artifact layer model model (Liu et al., 2010), DSO (Rockwell et al., 2009), DO (Štorga et al., 2010), and the work about knowledge needs of designers (Ahmed & Wallace, 2004). Before describing our ontology, the extended IBIS-based DR representation is briefly introduced.

4.1. DR representation for knowledge retrieval and reuse

In order to effectively support the retrieval and reuse of design knowledge, a DR representation should generally have the following characteristics: expressive enough to represent the design knowledge generated in the design process; formal enough to support computation; and easy to be captured (Qin et al., 2012). However, existing DR representations are not good enough in these aspects. Traditional DR representations do not have sufficient expression capabilities, and most of them are not formal enough to be understood by a computer.

Ahmed and Wallace (2004) performed a comprehensive analysis of the discourse between novice designers and experienced designers and identified 11 main types of knowledge needs including *how does it work*, *why*, *what issues to consider*, *when to consider issues*, and *design process*. IBIS (Kunz & Rittel, 1970) is a traditional DR representation that starts with *issues*, and each *issue* is followed by one or more *solutions* that respond to the issue. *Arguments* either support or object to a solution. The dashed line box in Figure 2 shows the relationships between three elements in IBIS. In addition, it can express much of the first 3 knowledge needs; moreover, we find that ***Requirement*** can answer *why* and *when*, ***Function*** describes *how*, and ***Artifact*** is highly related to *design process*.

Based on the analysis above, we propose an extended IBIS-based DR representation. As shown in Figure 2, the extended DR elements are ***Requirement***, ***Artifact***, and ***Function***. Design requirements are specifications of some conditions that the product needs to meet, which include *functional requirement* and *nonfunctional requirement. Functional requirement* can drive the design and lead to some issues; hence, what it relates to is ***Issue***; meanwhile, *nonfunctional requirement* plays the role of design constraint, and what it relates to is ***Argument***. ***Artifact*** and ***Function*** provide supplementary information that helps the designer to better understand the design knowledge better as supplements. *Functions* can be found from *issues*; an *artifact* contains *issues*, and a *solution* decides what an *artifact* is like. In this paper, we introduce the DR representation briefly and propose some concepts and relationships for designing DR ontology.

4.2. DR ontology

Based on the proposed DR representation, a DR ontology is designed to support the indexing of DR retrieval. The main class hierarchy of the designed DR ontology is shown in Figure 3. We create the main concepts according to the extended IBIS-based DR representation, which are subclasses

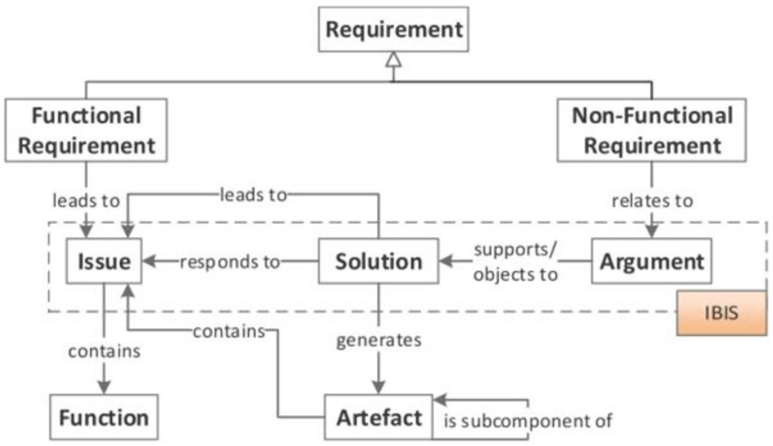


Fig. 2. Extended issue-based information system-based design rationale representation.

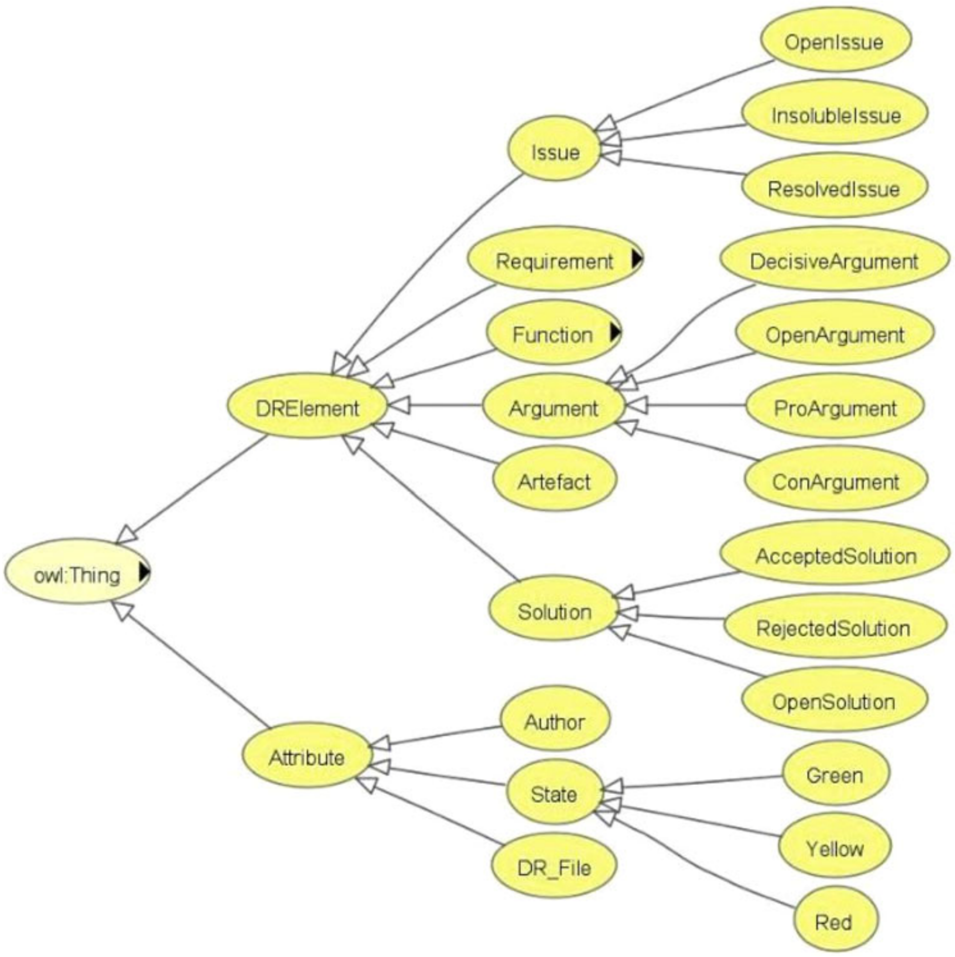


Fig. 3. Main class hierarchy of design rationale ontology.

of the class ***DRElement***. Then we refine the existing classes into subclasses. For ***Issue***, ***Solution***, and ***Argument***, we refine them into several types according to their states; for ***Function***, the function taxonomy of Hirtz et al. (2002), which contains 39 concepts, is introduced as its subclasses, such as *Branch, Channel*, *Convert*, and *Support*; and for ***Requirement***, the requirements list of Pahl et al. (2007), which contains 119 concepts, is referenced to enrich its subclasses, such as *Assembly, Costs, Ergonomics*, and *Forces*. Moreover, we add some other concepts according to the basic information of DR records such as author, filename, and so on, and they are classified as the subclasses of ***Attribute*** by referring to the work of Štorga et al. (2010). Finally, we add relationships between these concepts; for example, the relationship *support* is added as an object property for concepts ***Argument*** and ***Solution***. As a result, an ontology containing 184 concepts, 26 object properties, and 6 datatype properties in DR domain is created, and for the details of our work about DR ontology, please refer to Li et al. (2014).

It is worth noting that this ontology is specifically created for DR retrieval, and it is the semantic model of the proposed DR representation model. However, it is not as comprehensive as some previous ones, such as those developed by Rockwell et al. (2009), Ahmed and Wallace (2005), and Štorga et al. (2010). However, in order to improve the DR ontology, some classes such as ***ResolvedIssue*** and ***InsolubleIssue*** are specified to be disjoint, so that an individual (or object) cannot be an instance of more than one of these classes. In addition, some properties like *hasProArg* and *support* are specified to be inverse properties of each other. Because DR ontology contains two parts: ***DRElement*** (a simple structure of DR representation) and ***Attribute*** (DR retrieval specific), it is suitable for users who need to represent or retrieve DR.

5. AUTOMATIC TRANSLATION OF NATURAL LANGUAGE QUERY TO SPARQL QUERY

Formal query languages are required to retrieve ontology information. However, they are too complex to be used for normal users. Most people are used to using keyword-based or natural language-based query because this is how we speak. Considering that keywords, which have little semantic information, cannot make full use of ontology database, natural language queries are chosen as the user input for our retrieval approach, and finally turned into SPARQL queries.

To translate natural language queries into SPARQL queries, domain knowledge of engineering design and IR techniques such as keyword expansion are adopted. The overall process of SPARQL query generation is shown in Figure 4. Four SPARQL templates are predefined based on knowledge query requirements in engineering design, and four corresponding question types are also defined. Then, the natural language question is classified as one of the four types using natural language processing techniques, and a SPARQL template is matched accordingly. Moreover, a Lucene-based keywords extending method is performed to enhance the text searching ability of SPARQL. Finally, a complete SPARQL query is generated by combining the SPARQL template with the extended keywords.

5.1. Predefinition of SPARQL templates according to DR retrieval requirements

A natural language question gives us additional information on the type of information that is expected as an answer

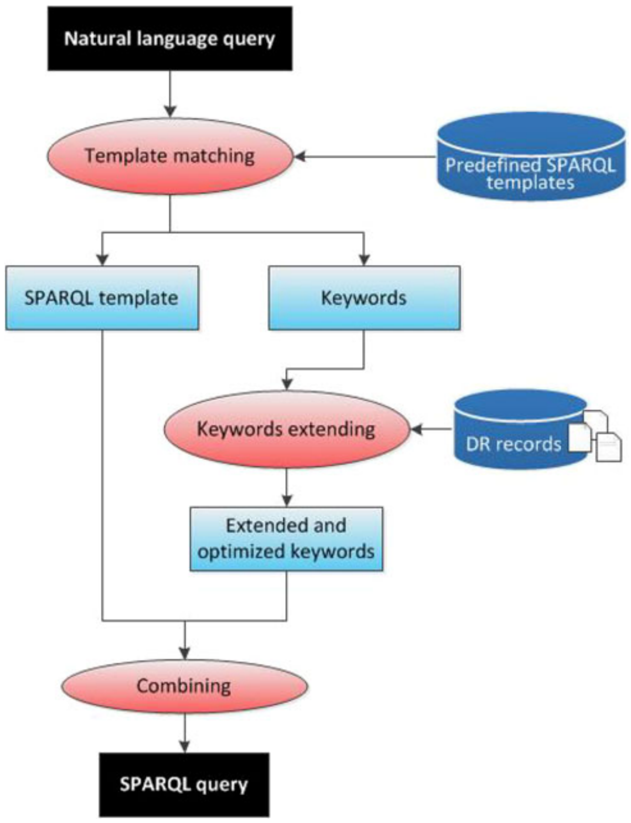


Fig. 4. Process of SPARQL query generation from natural language query.

(Kolomiyets & Moens, 2011). In general, there are many potential answer types for domain-independent knowledge that make it hard to classify natural language questions. However, there are a limited number of types of knowledge needs in engineering design domain, which makes it feasible to predefine SPARQL templates according to domain-specific knowledge needs.

During engineering design, a vast amount of knowledge that has many different types is generated, and the question arises as to which parts of it are the most concerned and the most needed by designers. Ahmed et al. (2004) identified 11 main kinds of knowledge needs after a comprehensive analysis. Of these 11 types, DR can fulfill 4 needs: *How does it work, Why, What issues to consider*, and *When to consider issues*. The remaining 7 types of knowledge needs cannot currently be directly represented by DR. Therefore, only these 4 types of knowledge needs are considered for DR retrieval, which are listed in Table 1.

For domain-independent knowledge needs, it is hard to define complete SPARQL templates, because there are too many types of knowledge needs that contain different relationships. However, it is a completely different situation for some specific knowledge needs. According to the knowledge needs in the engineering design domain that can be represented by DR, there are four corresponding SPARQL templates defined, which are also shown in

Table 1. *SPARQL templates for natural language queries and corresponding knowledge needs*

Names	SPARQL Templates	Descriptors	Knowledge Needs
T1	SELECT ?solutionNar WHERE{ ?solution DR:respondTo ?issue . ?solution DR:narration ?solutionNar . ?issue DR:narration ?issueNar FILTER regex(?issueNar, “keyword”) }	“how”	How it works
T2	SELECT ?argNar WHERE{ ?argument DR:argumentFor ?solution . ?argument DR:narration ?argNar . ?solution DR:narration ?solutionNar FILTER regex(?solutionNar, “keyword”) }	“why”	Why
T3	SELECT ?issueNar WHERE{ ?issue rdf:type DR:Issue . ?issue DR:narration ?issueNar . FILTER regex(?issueNar, “keyword”) }	“what”	What issues to consider
T4	SELECT ?objectNar WHERE{ ?issue DR:causedBy ?object . ?object DR:narration ?objectNar . ?issue DR:narration ?issueNar FILTER regex(?issueNar, “keyword”) }	“when”	When to consider issues
T5	SELECT ?objectNar WHERE{ ?object DR:narration ?objectNar. FILTER regex(?objectNar, “keyword”) }	—	—

Note: T1–5, Templates 1–5.

Table 1. In these templates, we use the following abbreviations as the prefixes:

- DR for <http://www.owl-ontologies.com/DesignRationale.owl\#>
- rdf for <http://www.w3.org/1999/02/22-rdf-syntax-ns\#>

All of the four different types of knowledge query requirements can be fulfilled using DR information. Specifically, for *How does it work*, the issues are related to the keywords (meaning that they contain either the keywords or synonyms of the keywords), and the results will be the solutions of these issues; for *Why*, the solutions are related to the keywords, and the results are the arguments of the solutions; for *What issues to consider*, the issues that are directly related to the keywords will be shown as results; for *When to consider issues*, issues are related to the keywords, and the results can be obtained by the requirements or solutions that lead to the issue.

Although the aforementioned four knowledge needs can meet most of the designers' requirements for DR, there do exist some other knowledge needs. For completeness of our retrieval system, a fifth template is created to handle all of the remaining knowledge needs, which is also shown in Table 1. This template will find all of the DR nodes that contain specific keywords, which is akin to keyword-based retrieval.

5.2. Template-based generation of initial SPARQL query

SPARQL template matching is a key task in translation of a natural language query into a SPARQL query, which automatically selects a SPARQL template by analyzing the natural language query in order to obtain the query requirement.

The designers' requirements for design knowledge that were identified by Ahmed et al. (2004) are used to adopt a question-answering strategy to deal with natural language query; that is, for each query, the expected answer type is first identified, and the keywords involved in the query are also extracted for later use. Generally, the expected answer type is identified using interrogative words, and the expected answer type exactly corresponds to one of the knowledge needs mentioned in Section 4.1. Therefore, the interrogative words *how*, *why*, *what*, and *when* correspond to the four SPARQL templates respectively.

The specific steps of the SPARQL template-matching process are as follows:

1. *Parse the natural language query.* The Stanford Parser (http://nlp.stanford.edu/software/index.shtml) is used to parse the natural language query to achieve POS tagging of each word.

2. *Identify the expected answer type and select the template.* Each word tagged by WRB is compared with the four interrogative words *how*, *why*, *what*, and *when*, which correspond to the four SPARQL templates in Table 1. If one of the four interrogative words is in the query, the corresponding SPARQL templates will be selected automatically; otherwise, the fifth template T5 will be selected. It is noteworthy that usage of template T5 here ensures that the retrieval system is robust to all natural language queries.
3. *Extract the keywords.* The verbs, nouns, and adjectives of the natural language query are tagged using labels VB, VBD, VBG, VBN, JJ, JJR, JJS, or NN and are extracted as the keywords Q_{init} for later use.

5.3. Generation of final SPARQL query using keyword extension and optimization

Due to the limited ability of SPARQL in text search (e.g., SPARQL only supports exact string matching), different word forms and synonyms of a keyword will be ignored by a SPARQL-based retrieval, which will lower the retrieval recall dramatically. To resolve this problem, some sophisticated features of IR such as stemming and synonym expansion are utilized. Specifically, Lucene is adopted to realize the keyword extension.

The overall process of keyword extension includes the following four steps, which are shown in Figure 5 above the dotted line.

1. Read the WordNet (http://wordnet.princeton.edu/) index I_w first, and then add a stem field for each doc of the index to produce a new WordNet index $I_{w'}$, which enables synonyms to be searched using stems.
2. Filter the keyword set Q_{init} acquired in Section 4.2 using a stopword filter. This eliminates most of the unimportant words to generate a keyword set Q.
3. The set of extended keywords Q_i will be formed after repeating step (a) and step (b) for each q_i Q $(i = 1, 2, \ldots, n)$.
    a. Use a snowball filter to extract the stem of q_i, and then obtain the synonyms of q_i through index $I_{w'}$.
    b. Get stems of all words from S_i, and search all the original DR records with these stems to obtain the corresponding keywords Q_i in raw text.
4. Merge every extended keyword set Q_i $(i = 1, 2, \ldots, n)$ to form a new set Q_{ext}, which is the set of final extended keywords of Q_{init}.

The example at the bottom of Figure 5 shows the keyword extension process for the word "provide." The set of extended keywords includes five different words. Without this keyword extension process, users would not find the results corresponding to "provided" and "providing," due to the limited text search ability of SPARQL, not to mention "offer" and "supplying." It should be noted that the extended keywords Q_{ext} may not always contain the initial keywords Q_{init} because all of the words in Q_{ext} appear in raw text while some of the words in Q_{init} may not, which will be proven in Section 7 using test case 3.

To obtain a complete SPARQL query with the ability of text search, the SPARQL template and the extended keywords should be combined. The Boolean OR has been used to handle the extended keywords, and then add the set into the template. Meanwhile, some additional triples have been added to the template to get more detailed information about the answer, such as the type, state, and filename. The example shown below is a complete SPARQL query generated from the natural language query "How to provide force," which has chosen the first template in Table 1 and added the extended keywords "provide force" into the template. Due to the extended keywords, the example SPARQL query will find issues that contain the extended keywords (provide, provided, offer, providing, supplying, force, etc.), and then return the solutions of the issues as results. However, without the extended keywords, the retrieved results would only contain the solutions of issues containing the original keywords (provide and force). It is noteworthy that there are much more than three extended keywords of the query; we have only shown "provide," "supplying," and "force" here because of limited space.

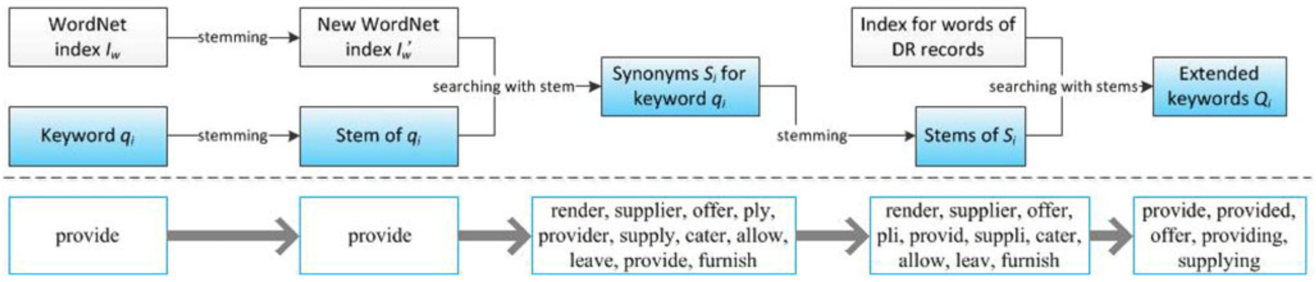

WordNet index I_w
stemming
New WordNet index I_w'
searching with stem
Synonyms S_i for keyword q_i
Index for words of DR records
searching with stems
Extended keywords Q_i
Keyword q_i
stemming
Stem of q_i
stemming
Stems of S_i
provide
provide
render, supplier, offer, ply, provider, supply, cater, allow, leave, provide, furnish
render, supplier, offer, pli, provid, suppli, cater, allow, leav, furnish
provide, provided, offer, providing, supplying

Fig. 5. Main process of keyword extension.

```
SELECT DISTINCT *
WHERE{ ?solution DR:respondToI ?issue .
   ?solution DR:narration ?solutionNar .
   ?issue DR:narration ?issueNar .
   ?solution rdf:type ?type .
   ?solution DR:hasState ?color.
   ?solution DR:belongToAFile ?drFile.
   ?drFile DR:hasFileName ?fileName
   FILTER (regex(?issueNar, "provide") || regex(?issueNar,
     "supplying") ||
   regex(?issueNar, "force"))
}
```

6. AUTOMATIC TRANSLATION OF DR RECORD-BASED QUERY TO SPARQL QUERY

Compared with keywords and natural language, DR records are more structured with more abundant DR semantics, which can help designers better express their knowledge requirements. With this consideration, DR record-based query is supported in our DR retrieval system as an accurate query mode. This query mode is imperative for an integrated DR capture and retrieval system. For example, when a designer creates a new DR file during his design work, for each issue that is inserted, the designer wants to determine whether solutions exist in response to this issue in the DR database. At this point, the DR record-based query can be very helpful to the user. The purpose of this section is to translate DR record-based query to SPARQL query automatically.

Figure 6 shows the overall translation process from DR record-based query to SPARQL query. An attempt is made to match the query with one of the SPARQL templates. If the match succeeds, the initial SPARQL query is directly generated, which will be discussed in more detail in Section 6.2. If the match fails, the initial SPARQL query is automatically generated based on ontology, which will be explained in Section 6.1. In addition, a Lucene-based keyword extension and optimization method is used to enhance the text searching ability of SPARQL. Finally, a complete SPARQL query is generated by combining the initial SPARQL query with the extended and optimized keywords.

6.1. Ontology-based SPARQL query generation

Because there is an indefinite number of DR nodes and relationships within a DR record-based query, it is not feasible to predefine complete SPARQL templates for the DR record-

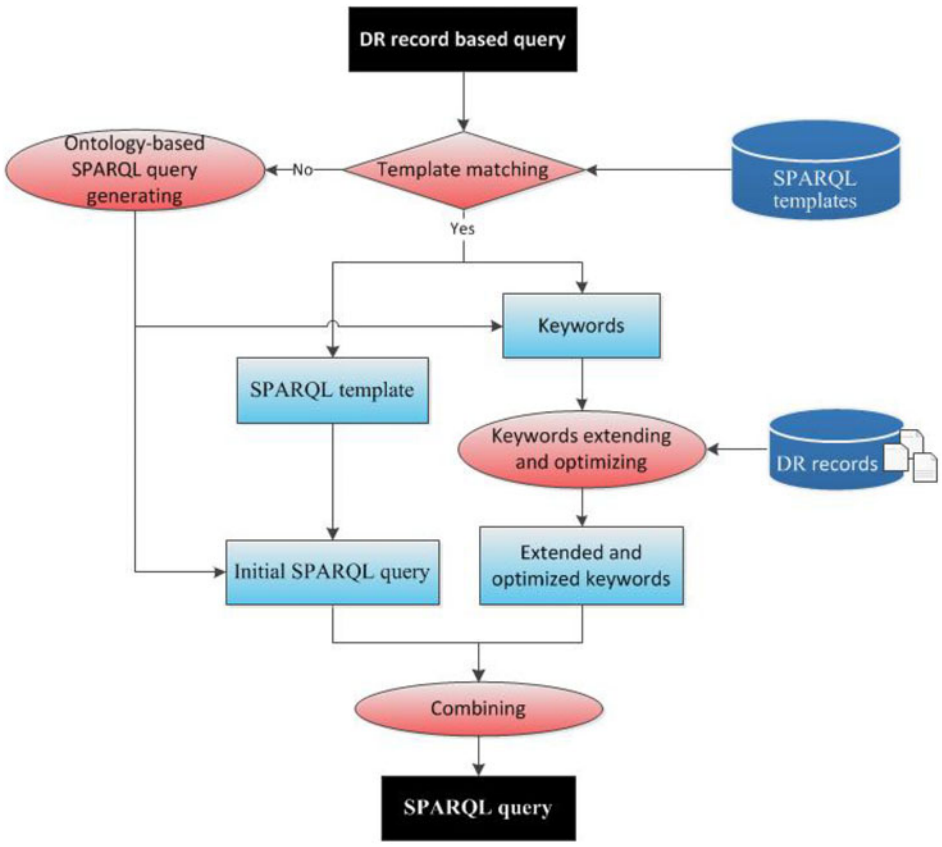


Fig. 6. Translation process of design rationale record-based query to SPARQL query.

based query to match. To ensure that all of the DR record-based queries can be translated into SPARQL queries, we propose an ontology-based translation method, which specifically includes following seven steps:

1. *Parse the DR record-based query:* Extract the DR node types and the relationships between nodes to obtain the keywords Q_{init} for each DR node.
2. *Generate SPARQL statements based on the DR node type:* Determine the concept of DR ontology according to the DR node type, and then generate the corresponding SPARQL statements. In DR ontology, all of the concepts that correspond to particular node types are subclasses of ***DRElement***. If the type of the *node* is "TYPE," the corresponding SPARQL statement is "?node rdf:type DR:TYPE."
3. *Generate SPARQL statements based on the relationships between DR nodes:* Determine the object property of DR ontology according to the relationship between DR nodes. The types of relationships that exist include *respondTo*, *argumentFor*, *causedBy*, and *relatedArgument*. If the relationship between *node1* and *node2* is "RELATIONSHIP," then the corresponding SPARQL statement is "?node1 DR:RELATIONSHIP ?node2."
4. *Generate SPARQL statements based on the text content of the DR node:* If a DR node contains some text information, then the corresponding SPARQL statements are "?node DR:hasNarration ?content . FILTER regex(?-content, 'keyword')," where "content" and "keyword" are consistent with "node" through a certain number.
5. *Generate the initial SPARQL query by combining the generated SPARQL statements:* Add a SPARQL statement "SELECT DISTINCT * WHERE{}," with the SPARQL statements generated above within the braces.
6. *Extend and optimize the keywords:* For each DR node, process the initial keywords Q_{init} that was created in step (1) using the keyword extension and optimization algorithm described in Section 4.3, and then get the updated keywords Q_{ext}.
7. *Generate the final SPARQL query:* Use the extended keywords Q_{ext} to replace "keyword" in the initial SPARQL query, and use Boolean OR to handle multiple keywords. In addition, in order to display the results more intuitively, add some SPARQL statements on the state of the DR node and filename, and so on.

After executing the seven steps above, a corresponding SPARQL query will be generated for a DR record-based query. Figure 7 shows an example of the translation process from a DR record-based query to a SPARQL query, where the seven numbers correspond to each of the seven steps above.

6.2. Template-based SPARQL query generation

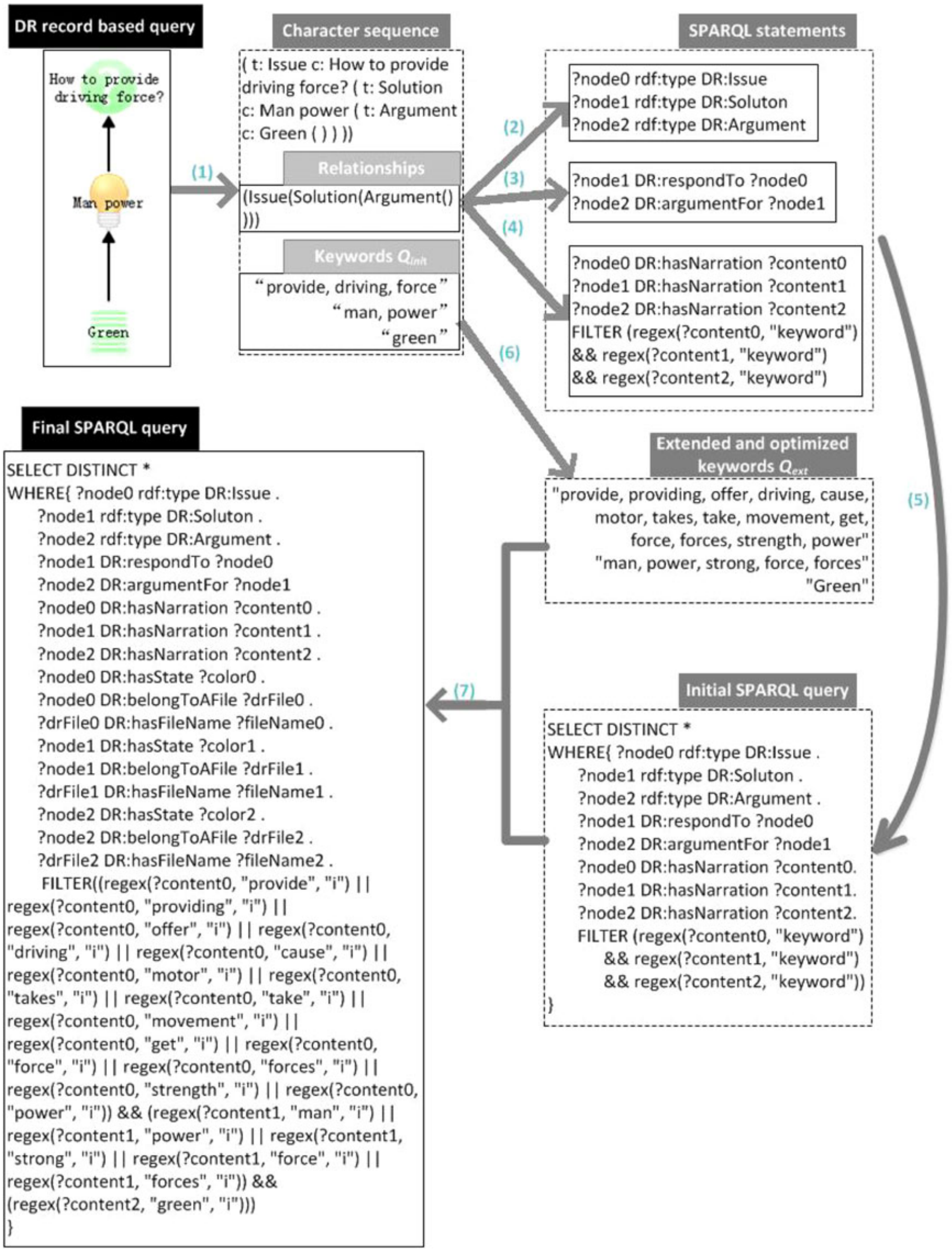
SPARQL template-based translation matches a DR record-based query with one of the SPARQL templates in a template library, then extends and optimizes the keywords, thus generating the final SPARQL query. Compared with the ontology-based translating method, SPARQL template-based translating is more effective.

Compared with natural language query, DR record-based query contains more complex DR node types and relationships between DR nodes, and expresses users' query requirements more effectively. However, the uncertainty of DR nodes and relationships mean that it is not feasible to predefine complete SPARQL templates for all DR record-based queries. Therefore, the SPARQL template and its corresponding descriptor for a DR record-based query are extracted when the ontology-based SPARQL query generation method described in Section 5.1 is used.

The SPARQL template corresponding to a DR record-based query expresses the DR node types and the relationships between nodes. Therefore, its descriptor should include as much information as possible to node types and the relationships between nodes, and it should be relatively simple to enhance the efficiency of the template matching. Based on the above analyses, the descriptor for the SPARQL template corresponding to the DR record-based query can be defined as a string formed by the depth-first traversal-ordered DR node types. Specifically, there are three points that should be noted. First, for each node of a DR record, its descriptor is of the form (type(children)), where "type" denotes DR the node type, and "children" denotes all descendant nodes of current DR node. If there is no "children," the current node is a leaf node. The types of DR node include **Is**sue, **So**lution, **Ar**gument, and **F**unctional**R**equirement, where the bold letters show the descriptor that is used as an abbreviation of the DR node type. Second, for the query that is for similar results, its descriptor can be represented as a type tree. An example of the descriptor extraction process is shown in Figure 8. Third, for the query that is for wanted results, its descriptor can be represented as a type tree adding "?" after the type abbreviation that corresponds to the blank node. An example of the descriptor extraction process is shown in Figure 9.

The following steps are used for template-based SPARQL query generation from the DR record-based query:

1. *Extract the query's descriptor:* The descriptor generation method described above can be used to extract the corresponding descriptor for the DR record-based query that is input by a user. It is worth noting that this step is the same as the first step in Section 6.1 except for keyword extraction.
2. *Search for a SPARQL template with the descriptor:* Use the descriptor to match to a SPARQL template. If the matching succeeds, choose the corresponding SPARQL template as the initial SPARQL query.
3. *Extract the keywords:* For each DR node, extract the verbs, nouns, and adjectives (words that are tagged by the Stanford Parser as VB, VBD, VBG, JJ, JJR, or



```
( t: Issue c: How to provide
driving force? ( t: Solution
c: Man power ( t: Argument
c: Green ( ) ) ))
```

```
(Issue(Solution(Argument()
)))
```

```
"provide, driving, force"
"man, power"
"green"
```

```
?node0 rdf:type DR:Issue
?node1 rdf:type DR:Soluton
?node2 rdf:type DR:Argument
```

```
?node1 DR:respondTo ?node0
?node2 DR:argumentFor ?node1
```

```
?node0 DR:hasNarration ?content0
?node1 DR:hasNarration ?content1
?node2 DR:hasNarration ?content2
FILTER (regex(?content0, "keyword")
&& regex(?content1, "keyword")
&& regex(?content2, "keyword")
```

```
"provide, providing, offer, driving, cause,
motor, takes, take, movement, get,
force, forces, strength, power"
"man, power, strong, force, forces"
"Green"
```

```
SELECT DISTINCT *
WHERE{ ?node0 rdf:type DR:Issue .
    ?node1 rdf:type DR:Soluton .
    ?node2 rdf:type DR:Argument .
    ?node1 DR:respondTo ?node0
    ?node2 DR:argumentFor ?node1
    ?node0 DR:hasNarration ?content0.
    ?node1 DR:hasNarration ?content1.
    ?node2 DR:hasNarration ?content2.
    FILTER (regex(?content0, "keyword")
        && regex(?content1, "keyword")
        && regex(?content2, "keyword"))
}
```

```
SELECT DISTINCT *
WHERE{ ?node0 rdf:type DR:Issue .
    ?node1 rdf:type DR:Soluton .
    ?node2 rdf:type DR:Argument .
    ?node1 DR:respondTo ?node0
    ?node2 DR:argumentFor ?node1
    ?node0 DR:hasNarration ?content0 .
    ?node1 DR:hasNarration ?content1 .
    ?node2 DR:hasNarration ?content2 .
    ?node0 DR:hasState ?color0 .
    ?node0 DR:belongToAFile ?drFile0 .
    ?drFile0 DR:hasFileName ?fileName0 .
    ?node1 DR:hasState ?color1 .
    ?node1 DR:belongToAFile ?drFile1 .
    ?drFile1 DR:hasFileName ?fileName1 .
    ?node2 DR:hasState ?color2 .
    ?node2 DR:belongToAFile ?drFile2 .
    ?drFile2 DR:hasFileName ?fileName2 .
     FILTER((regex(?content0, "provide", "i") ||
regex(?content0, "providing", "i") ||
regex(?content0, "offer", "i") || regex(?content0,
"driving", "i") || regex(?content0, "cause", "i") ||
regex(?content0, "motor", "i") || regex(?content0,
"takes", "i") || regex(?content0, "take", "i") ||
regex(?content0, "movement", "i") ||
regex(?content0, "get", "i") || regex(?content0,
"force", "i") || regex(?content0, "forces", "i") ||
regex(?content0, "strength", "i") || regex(?content0,
"power", "i")) && (regex(?content1, "man", "i") ||
regex(?content1, "power", "i") || regex(?content1,
"strong", "i") || regex(?content1, "force", "i") ||
regex(?content1, "forces", "i")) &&
(regex(?content2, "green", "i")))
}
```

Fig. 7. Example of the translation process from design rationale record-based query to SPARQL query.

NN) from the text information as the initial keywords Q_{init}.

4. *Extend and optimize the keywords:* For each DR node, process the intial keywords Q_{init} from step 1 using the keyword extension and optimization algorithm described in Section 4.3, and then obtain the updated keywords Q_{ext}.
5. *Generate the final SPARQL query:* Use the extended keywords Q_{ext} to replace "keyword" in the initial SPARQL query, and use Boolean OR to multiple keywords. In addition, in order to display the results more intuitively, add some SPARQL statements relating to the state of DR node and the file name, and so forth.

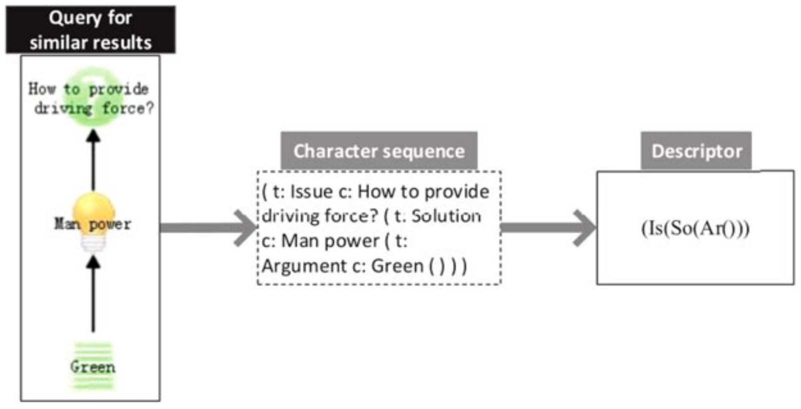

Query for similar results
How to provide driving force?
Man power
Green
Character sequence
(t: Issue c: How to provide driving force? (t. Solution c: Man power (t: Argument c: Green ( )))
Descriptor
(Is(So(Ar()))

Fig. 8. Example of descriptor extracting process on query for similar results.

7. IMPLEMENTATION

The proposed DR retrieval approach has been implemented in a multimodule prototype system. The core module that realizes the retrieval function is developed using Java, and the user interface module is developed using Qt 4.7.3, which is integrated with our capture tool (Li et al., 2013). Moreover, Jena 2.10.1 is used for handling the OWL files, Jena TDB for constructing database of ontology-based DR, and Lucene 3.6.2 for the keyword extension.

The DR records utilized in this study are captured using our DR capture tool, which has been developed based on the extended IBIS-based DR representation. The three IBIS elements are given a "traffic light" status (yellow means open status; green means resolved issue, accepted solution, or a pro; and red means insoluble issue, rejected solution, or a con), which refers to Dred (Bracewell et al., 2009). Currently, our DR records for retrieval consist of 106 DR files, which are captured by experienced engineering designers. There are a total of 1473 DR nodes connected with 1152 edges, after reasoning with 18 SWRL rules, an additional 4419 node types, and 978 relationships.

7.1. SPARQL engine

In order to cope with the growing amount of DR information and fully exploit the semantics of it, an ontology-based method to construct the DR database is proposed. The key points are how to translate DR records captured by engineering designers into DR instance ontologies, and how to store the large amount of DR instance ontologies and the proposed DR ontology. DR records correspond to separate DR instance ontologies, and retrieving over these ontologies needs to manage them as whole object; therefore, we need to store these instance ontologies in a database.

The basic process for the DR database constructing is illustrated in Figure 10. DR records are instantiated into corresponding DR instance ontologies based on DR ontology. Then, ontology reasoning is conducted to enrich the instance ontologies. Finally, DR ontology and related instances are stored into a database. Meanwhile, the RDF query language SPARQL is supported for searching the DR database.

7.1.1. Generating of DR instance ontologies through ontology populating and ontology reasoning

DR ontology is used to automatically translate DR records into DR instance ontologies through ontology population.

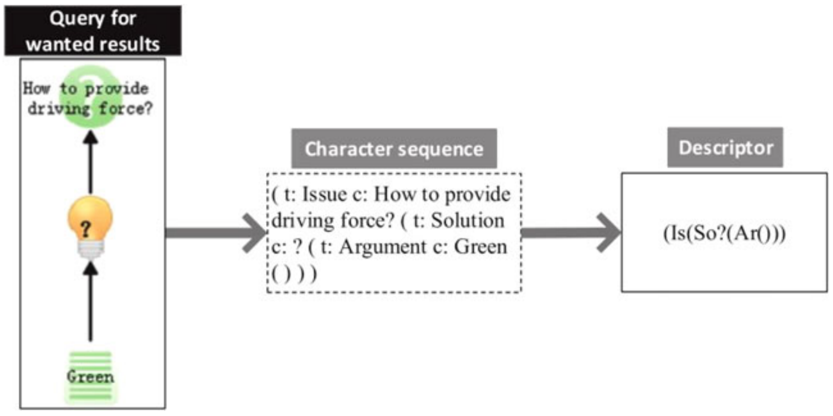

Query for wanted results
How to provide driving force?
?
Green
Charactor sequence
(t: Issue c: How to provide driving force? (t: Solution c: ? (t: Argument c: Green ( )))
Descriptor
(Is(So?(Ar()))

Fig. 9. Example of descriptor extracting process on query for wanted results.

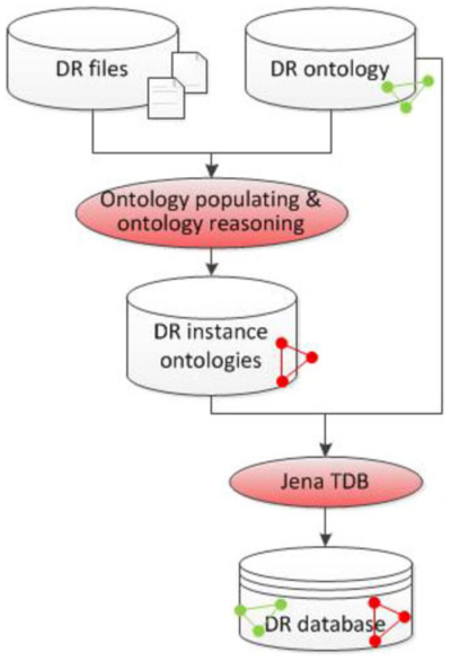


Fig. 10. Process of design rationale database constructing.

Ontology population is a knowledge acquisition activity, which transforms or maps unstructured, semistructured, and structured data into instance data. In addition, ontology reasoning is conducted to enrich the instance ontologies.

In this work, the DR instance ontologies generation process includes the following five steps, of which four steps are for ontology population and the final step is for ontology reasoning.

1. *Create ontology individuals for DR nodes:* An ontology individual is created for each DR node, and which class it belongs to depends on the DR node's type.
2. *Create data type properties for DR nodes:* Information inside the DR node such as text and state is added as the ontology individual's properties.
3. *Create object properties between DR nodes:* Once all DR nodes have been processed using the two steps above, the relationships between the DR nodes are added to the instance ontology as object properties of the individuals.
4. *Create ontology individuals and properties for the DR file:* Ontology population is not restricted to the DR nodes. DR files also contain basic information, including authors, creation date, and modification date, which together with the filenames are also added to the instance ontology by creating OWL individuals and properties.
5. *Infer semantic information using SWRL rules:* SWRL rules are adopted to obtain as much of the inferred semantic information as possible, which can be used to effectively improve the performance of the DR retrieval. Some examples are given in Table 2 to illustrate the benefits of SWRL rules. Rule 1 implies that if an ***Issue*** node's state is ***Yellow***, its node type could be ***OpenIssue***, and this rule is able to infer more specific type for an ontology individual according to DR node's state. Rule 2 means that if an accepted solution has both pros and cons at the same time, and there is only one supporting argument, then we can infer that the supporting argument is very important. Both Rule 3 and Rule 4 add a relationship between ontology individuals. Specifically, Rule 3 infers that the relationship between a ***ProArgument*** node and a ***Solution*** node is *support*; and Rule 4 implies that if an issue *i1* has a solution *s*, and *s* leads to another issue *i2*, then it should be assumed that *i2* affects *i1*.

Table 2. *Examples of SWRL rules*

1. Issue(?i) ∧ hasState(?i, Yellow) → OpenIssue(?i)
2. AcceptedSolution(?s) ∧ hasConArgNo(?s, ?no1) ∧ greaterThan(?no1, 0) ∧ hasProArg(?s, ?a) ∧ hasProArgNo(?s, ?no2) ∧ isEqualTo(?no2, 1) → DecisiveArgument(?a)
3. ProArgument(?a) ∧ Solution(?s) ∧ hasArgument(?s, ?a) → support(?a, ?s)
4. Solution(?s) ∧ Issue(?i1) ∧ Issue(?i2) ∧ repondTo(?s, ?i1) ∧ leadTo(?s,?i2) → affect(?i2, ?i1)

For the details of our work on translating DR files into ontologies, ontology population, and ontology reasoning, please refer to Li et al. (2014).

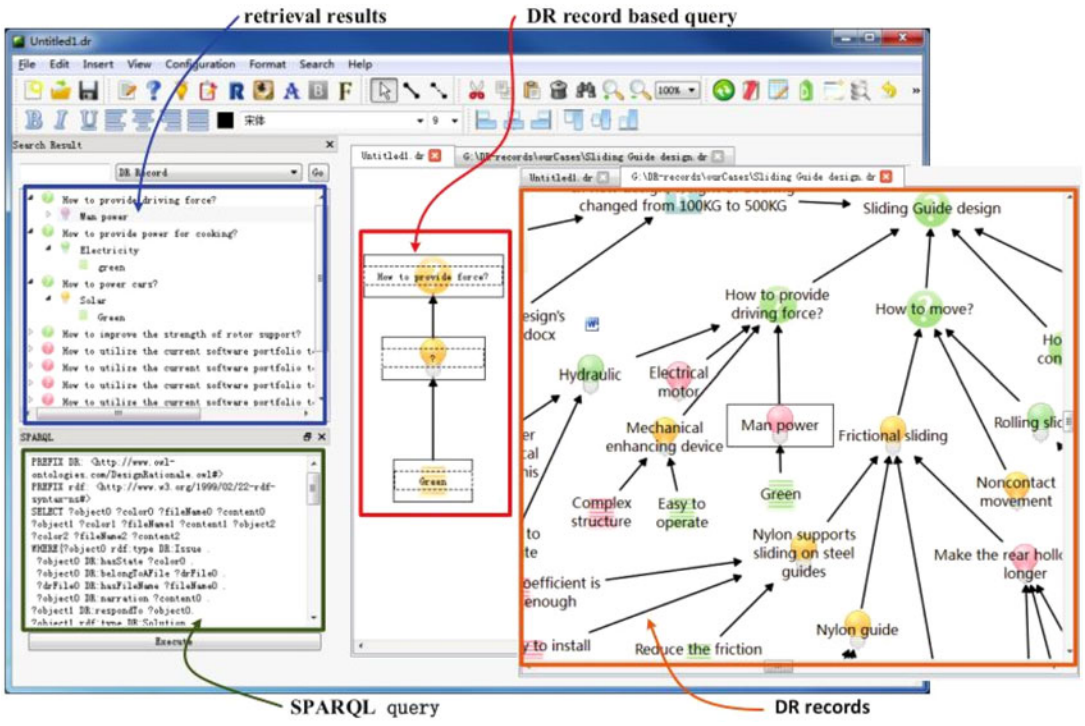
7.1.2. *Creation of DR database supporting SPARQL query*

SPARQL is the most widely used query language for OWL ontology. However, when there are large scales of ontology information, it is not possible to load all information into memory at the same time, so indexing should be performed before searching. The key point is that when users search with indexes, SPARQL query should be supported as well. Fortunately, Jena (http://jena.apache.org/) provides a component TDB for RDF storage and query specifically to support SPARQL.

TDB is a component of Jena for RDF storage and query. It supports the full range of Jena application programming interfaces. TDB can be used for high-performance RDF storage on a single machine.

7.2. Graphical user interface

Figure 11 shows an example of a DR retrieval with a DR record-based query. The red rectangle represents the DR record-based query that is input by the user, and the blue rectangle shows the search results that are returned to the user. The bottom left panel shows the SPARQL query that is generated from the DR record-based query, and the DR record appears



```
How to provide driving force?
    Man power
How to provide power for cooking?
    Electricity
        green
How to power cars?
    Solar
        Green
How to improve the strength of rotor support?
How to utilize the current software portfolio t
How to utilize the current software portfolio t
How to utilize the current software portfolio t
How to utilize the current software portfolio t
```

```
PREFIX DR: <http://www.owl-
ontologies.com/DesignRationale.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>
SELECT ?object0 ?color0 ?fileName0 ?content0
?object1 ?color1 ?fileName1 ?content1 ?object2
?color2 ?fileName2 ?content2
WHERE{?object0 rdf:type DR:Issue .
 ?object0 DR:hasState ?color0 .
 ?object0 DR:belongToAFile ?drFile0 .
 ?drFile0 DR:hasFileName ?fileName0 .
 ?object0 DR:narration ?content0 .
?object1 DR:respondTo ?object0.
?object1 rdf:type DR:Solution .
```

Fig. 11. User interface of the design rationale capture and retrieval system.

on the right panel. When a single line of the results is double-clicked, the corresponding DR record will be shown in the right panel, and the focus will move to the corresponding DR node. It is worth noting that users can directly input SPARQL queries into the SPARQL panel, as well as showing the automatically generated SPARQL query. The example in Figure 11 also shows the overall process for our proposed retrieval process. The user input query (red rectangle) is translated into a SPARQL query (green rectangle), and then the SPARQL query is executed to get the results (blue rectangle).

8. EXPERIMENTS AND EVALUATIONS

The DR retrieval approach proposed in this paper takes advantage of both ontology and SPARQL. Ontology plays a key role in improving the recall and precision of DR retrieval, which is evaluated in Section 8.1. SPARQL is a commonly used method of retrieving DR, and our proposed SPARQL that is enhanced with a text search method greatly improves the retrieval recall, which is evaluated in Section 8.2. The metrics most commonly used to assess the effectiveness of IR systems are precision and recall, which are defined in this work as follows:

$$\textit{precision} = \textit{the number of relevant DR records retrieved}/\textit{the number of retrieved DR records}.$$

$$\textit{recall} = \textit{the number of relevant DR records retrieved}/\textit{the number of relevant DR records in the DR database}.$$

8.1. Evaluation of ontology-based DR retrieval

The retrieval recall and precision of three different methods is tested in order to evaluate the benefits of ontology in our retrieval approach. Method 1 is keyword-based retrieval for the original DR files, Method 2 is our proposed enhanced SPARQL-based retrieval without ontology reasoning, and Method 3 is our proposed enhanced SPARQL-based retrieval with ontology reasoning.

In order to demonstrate the effectiveness and usefulness of the proposed approach in industrial practice, more than half of the test queries shown in Table 3 are taken from real cases in engineering design. Specifically, three of the queries (Q1, Q4, and Q5) relate to the design of a maintenance tool for assembling a gas turbine journal bearing. The overall function of this tool is to move a bearing to the right place, which can be broken down into four subfunctions: providing driving force, moving, connecting, and lifting. The queries are mainly related to providing driving force and moving. Figure 11 shows some of the DR about this maintenance tool. In addition, Q7 relates to the design of a rotor's cooling system. There are three alternatives for this design issue: single-stage internal air cooling, two-stage internal air cooling, or external cooler. The purpose of Q7 is to search for the pros of using an external cooler.

The retrieval results are also shown in Table 3. The first three queries should be considered, which are all keyword-based queries. There is an obvious increase in the precision values as more semantic information is captured in the index. Taking Q1 as an example, it can be seen that when Method 1

Table 3. *Retrieval results corresponding to three different retrieval methods*

Test Queries	Keyword (Method 1)		Ontology Without Reasoning (Method 2)		Ontology (Method 3)	
	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.
Q1: provide force (**ResolvedIssue**)	0.034	1	0.2	1	0.333	1
Q2: sort (**OpenSolution**)	0.087	1	0.118	1	0.25	1
Q3: install (**ConArgument**)	0.294	1	0.556	1	1	1
Q4: How to provide force?	0.009	0.105	0.462	0.947	0.288	1
Q5: How to move?	0.007	0.111	0.12	0.333	0.25	1
Q6: Why not choose merge sort?	0	0	0.067	0.333	0.176	1
Q7: Why choose external cooler?	0.039	1	0.033	0.40	0.192	1

is used, all types of DR nodes that contain "provide force" are returned as results. When using Method 2, which contains some basic types of DR nodes such as ***Issue***, the range of results is limited to the ***Issue*** nodes. Furthermore, when using Method 3, which contains some more specific types of DR nodes such as ***ResolvedIssue***, the range of results is further limited. Therefore, the precision gradually increased with each method. It can also be seen in Table 3 that the recall values are increased from Method 2 to Method 3. The reason for this is that implicit relationships are uncovered by reasoning with the semantic rules in Table 2.

Our evaluation results show that Method 3 has the best retrieval performance, followed by Method 2, with Method 1 having the poorest performance. This is because Method 2 utilizes more semantic information than Method 1, and Method 3 uses much more inferred semantic information than either of the other two methods. This proves that the proposed DR retrieval approach is better than the traditional keyword-based retrieval method, and ontology reasoning plays an important role in improving the retrieval performance.

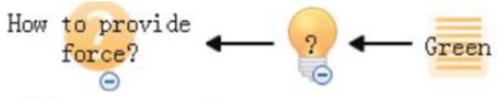
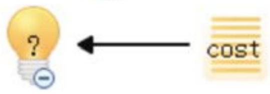
8.2. Evaluation of enhanced SPARQL-based retrieval

In order to evaluate the benefits of our proposed SPARQL-based retrieval approach, the retrieval recall and precision is tested for three different methods (Method 1 is keyword-based retrieval, Method 2 is SPARQL-based retrieval, and Method 3 is our proposed method, namely, SPARQL-based retrieval with text search). Section 8.1 evaluates the benefits of processing DR knowledge base with ontology. Based on this evaluation, this section evaluates the benefits of processing queries with SPARQL with text search.

The experiment is performed with a number of test cases. Table 4 provides the user input queries for the nine cases, which includes seven natural language queries and two DR record-based queries. Based on the results obtained using the three methods, comparisons of recall and precision were undertaken and are shown in Figure 12 and Figure 13, respectively. It is noteworthy that Method 1 cannot be used for Case 8 and Case 9.

As shown in these figures, searches using SPARQL with text search has the best performance in terms of recall and precision. Specifically, Figure 12 shows that the recall of Method 3 is much higher than the recall of Method 1 for the first five test cases because keyword-based query cannot fully express what users really want. However, the recall of the last two cases shows that Method 1 and Method 3 can get the same correct results for *what* questions, for there is no semantic relationship involved. Meanwhile, the recall of Method 3 is clearly higher than the recall of Method 2 in most cases due to the keywords extending. As shown in Figure 13, the precision of Method 2 and Method 3 is much larger than the precision of Method 1 due to the semantic restriction in SPARQL. In addition, the precision of Method 2 is larger than the precision of Method 3 for most cases, because the keyword extension may introduce some incorrect words. However, the keyword extension may make the retrieval precision higher occasionally, such as the precision comparison of Case 3, in which "carry" can be found by "car" using SPARQL, but the extended keywords do not include "car" because the exact word "car" does not appear in the raw text of the DR files.

Table 4. *User input queries for nine test cases*

Test Cases	User Input Queries
Case 1	How to provide force?
Case 2	How can a car be powered for?
Case 3	Why is the solar suited for powering a car?
Case 4	Why do we use gasoline as energy?
Case 5	When to consider "merge"?
Case 6	What issues to consider about "force"?
Case 7	What issues to consider about "offer"?
Case 8	
Case 9	

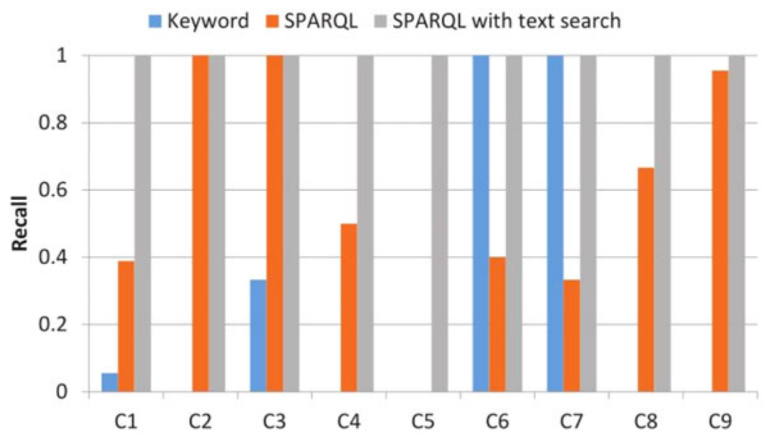


Fig. 12. Comparison of the retrieval recall for the three methods.

In summary, the evaluation results show that SPARQL-based retrieval makes a significant improvement over keyword-based retrieval in both recall and precision. Moreover, SPARQL query combined with keyword extension and optimization clearly enhances the recall compared with SPARQL query alone. Finally, although keyword extension may lower the retrieval precision, it can be ignored by identifying methods in the future that can be used to control the quality of the extended keywords.

9. CONCLUSIONS AND FUTURE WORKS

In this paper, an ontology-based DR retrieval approach combining SPARQL and text search has been presented. This work makes the following contributions:

1. A template-based SPARQL query generation method has been proposed, which translates user input queries to SPARQL queries automatically by matching to pre-defined templates, and allows normal users to benefit from SPARQL-based retrieval in a convenient way.
2. An ontology-based SPARQL query generation method has been proposed for DR record-based queries that cannot be translated by the template-based method, which enables normal users to more conveniently express more complex retrieval intentions.
3. A Lucene-based keywords extension and optimization method has also been proposed, which combines SPARQL with text search, thus enhancing the retrieval recall. A database of ontology-based DR has been constructed, which stores DR in a semantic way and supports structured query languages, enabling more accurate results to be searched.

In the future, several works will be done to improve the DR retrieval approach presented in this paper:

1. The synonym expansion is currently based on WordNet, which is not very accurate for a specific domain.

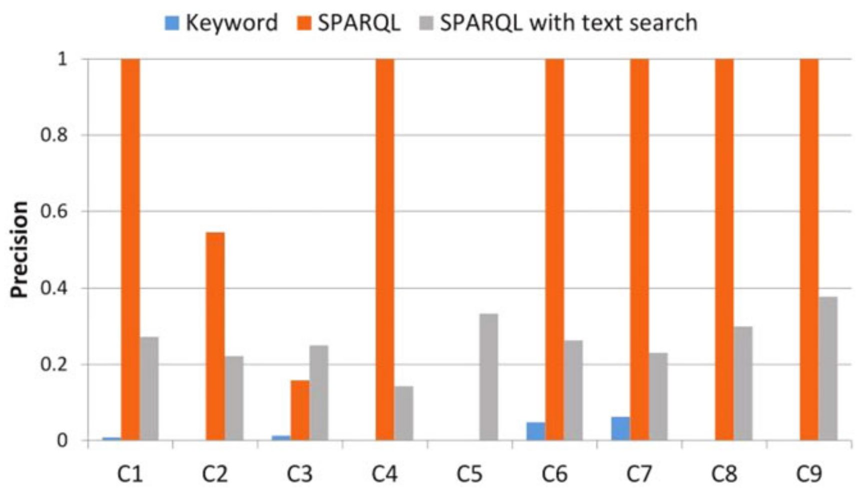


Fig. 13. Comparison of the retrieval precision for the three methods.

To further improve the effect of synonym expansion, it may be possible to replace WordNet with domain knowledge of engineering design such as functional basis.

2. Deeper research will be conducted on database and ontology. A larger DR database will be constructed to further prove the effectiveness of our retrieval approach.
3. Ambiguity of parsing natural language will be handled to improve the accuracy of translation from natural language query into SPARQL query.

ACKNOWLEDGMENTS

The authors are very grateful for the financial support from the National Science Foundation of China (61572432) and National 863 High Technology Plan (2013AA041301).

REFERENCES

Ahmed, S., & Wallace, K.M. (2004). Understanding the knowledge needs of novice designers in the aerospace industry. *Design Studies 25(2)*, 155–173.

Bracewell, R.H., Wallace, K.M., Moss, M., & Knott, D. (2009). Capturing design rationale. *Computer-Aided Design 41(3)*, 173–186.

Burge, J.E., & Brown, D.C. (2008). Software engineering using RATionale. *Journal of Systems and Software 81(3)*, 395–413.

Conklin, J., & Begeman, M.L. (1988). gIBIS: a hypertext tool for exploratory policy discussion. *Design Studies 12(4)*, 303–331.

De Medeiros, A.P., & Schwabe, D. (2008). Kuaba approach: integrating formal semantics and design rationale representation to support design reuse. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 22(4)*, 399–419.

Fenves, S.J., Foufou, S., Bock, C., & Sriram, R.D. (2008). A core model for product data. *Journal of Computing and Information Science in Engineering 8(1)*, 014501.

Gruber, T.R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition 5(2)*, 199–220.

Hirtz, J., Stone, R.B., McAdams, D.A., Szykman, S., & Wood, K.L. (2002). A functional basis for engineering design: reconciling and evolving previous efforts. *Research in Engineering Design 13(2)*, 65–82.

Kara, S., Alan, Ö., Sabuncu, O., Akpınar, S., Cicekli, N.K., & Alpaslan, F.N. (2012). An ontology-based retrieval system using semantic indexing. *Information Systems 37(4)*, 294–305.

Kim, S., Bracewell, R.H., & Wallace, K.M. (2005). A framework for design rationale retrieval. *Proc. Int. Conf. Engineering Design, ICED'05*. Melbourne, Australia: Design Society.

Kim, S., Bracewell, R.H., & Wallace, K. M. (2007). Improving design reuse using context. *Proc. Int. Conf. Engineering Design, ICED'07*. Paris: Design Society.

Kolomiyets, O., & Moens, M.-F. (2011). A survey on question answering technology from an information retrieval perspective. *Information Sciences 181(24)*, 5412–5434.

Kunz, W., & Rittel, H.W.J. (1970). *Issues as Elements of Information Systems*. Berkeley, CA: University of California at Berkeley.

Li, L., Qin, F., & Gao, S. (2013). An extended design rationale representation for supporting retrieval and reuse of design knowledge. *Journal of Computer-Aided Design & Computer Graphics (Chinese Journal) 25(10)*, 1514–1522.

Li, L., Qin, F., Gao, S., & Liu, Y. (2014). An approach for design rationale retrieval using ontology-aided indexing. *Journal of Engineering Design 25(7–9)*, 259–279.

Liang, Y., Liu, Y., Kwong, C.K., & Lee, K.B. (2012). Learning the "Whys": discovering design rationale using text mining—an algorithm perspective. *Computer-Aided Design 44(10)*, 916–930.

Liang, Y., Lu, W.F., Liu, Y., & Lim, S.C.J. (2010). Interactive interface design for design rationale search and retrieval. *Proc. ASME 2010 IDETC & CIE Conf.*, Montreal.

Lim, S.C.J., Liu, Y., & Lee, W.B. (2010). Multi-facet product information search and retrieval using semantically annotated product family ontology. *Information Processing & Management 46(4)*, 479–493.

Lim, S.C.J., Liu, Y., & Lee, W.B. (2011). A methodology for building a semantically annotated multi-facet ontology for product family modelling. *Advanced Engineering Informatics 25(2)*, 147–161.

Liu, J., & Hu, X. (2013). A reuse oriented representation model for capturing and formalizing the evolving design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 27(4)*, 401–413.

Liu, Y., Liang, Y., Kwong, C.K., & Lee, W.B. (2010). A new design rationale representation model for rationale mining. *Journal of Computing and Information Science in Engineering 10(3)*, 031009.

López, C., Cysneiros, L.M., & Astudillo, H. (2008). NDR ontology: sharing and reusing NFR and design rationale knowledge. *Proc. 1st Int. Workshop on Managing Requirements Knowledge, MARK'08*. Barcelona: IEEE.

MacLean, A., Young, R.M., Bellotti, V.M.E., & Moran, T.P. (1991). Questions, options, and criteria: elements of design space analysis. *Human-Computer Interaction 6(3)*, 201–250.

McCall, R.J. (1991). PHI: a conceptual foundation for design hypermedia. *ACM Transactions on Office Information Systems 6(1)*, 30–41.

Minack, E., Sauermann, L., Grimnes, G., Fluit, C., & Broekstra, J. (2008). *The Sesame LuceneSail: RDF queries with full-text search*. Technical Report 1, NEPOMUK Consortium.

Pahl, G., Wallace, K., & Blessing, L. (2007). *Engineering Design: A Systematic Approach*, Vol. 157. Berlin: Springer.

Qin, F., Li, L., & Gao, S. (2012). A survey of design rationale. *Journal of Computer-Aided Design & Computer Graphics (Chinese Journal) 24(10)*, 1283–1293.

Regli, W.C., Hu, X., Atwood, M., & Sun, W. (2000). A survey of design rationale systems: approaches, representation, capture and retrieval. *Engineering With Computers 16(3–4)*, 209–235.

Rockwell, J., Grosse, I.R., Krishnamurty, S., & Wileden, J.C. (2009). A decision support ontology for collaborative decision making in engineering design. *Proc. 2009 Int. Symp. Collaborative Technologies and Systems*, pp. 1–9, Baltimore, MD, May 18–22.

Štorga, M., Andreasen, M.M., & Marjanović, D. (2010). The design ontology: foundation for the design knowledge exchange and management. *Journal of Engineering Design 21(4)*, 427–454.

Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.-C., Gerber, D., & Ciminao, P. (2012). Template-based question answering over RDF data. *Proc. 21st Int. Conf. World Wide Web*, pp. 639–648. New York: ACM.

Wang, H., Johonson, A., & Bracewell, R.H. (2009). Supporting design rationale retrieval for design knowledge reuse. *Proc. Int. Conf. Engineering Design, ICET'09*. Stanford, CA: Design Society.

Wang, H., Johnson, A.L., & Bracewell, R.H. (2012). The retrieval of structured design rationale for the re-use of design knowledge with an integrated representation. *Advanced Engineering Informatics 26(2)*, 251–266.

Zhang, Y., Luo, X., Li, J., & Buis, J.J. (2013). A semantic representation model for design rationale of products. *Advanced Engineering Informatics 27(1)*, 13–26.

Zhu, L., Jayaram, U., Jayaram, S., & Kim, O. (2010). Querying and reasoning with product engineering ontologies—moving past modeling. *Proc. ASME 2010 IDETC&CIE Conf.*, Montreal.

Luye Li obtained his BS in computer science and technology from Wuhan University and his PhD in computer science and technology from Zhejiang University. His research is primarily in design rationale representation and design rationale retrieval. Dr. Li is also interested in geometry modeling and computer-aided design/computer-aided engineering integration.

Shuming Gao is a Professor in the State Key Lab of CAD&CG at Zhejiang University. He received his PhD from the Applied Mathematics Department of Zhejiang University and was a Visiting Scholar and a Visiting Professor in the Design Automation Lab of Arizona State University in 1996 and 2001, respectively. In 2002 he won the fund of the Trans-Century Training Programme Foundation for Talents by the Education Ministry of China. His research interest includes product modeling, CAX integration, Internet-based collaborative design, virtual reality in design and manufacturing, computer-aided design, concurrent engineering, and microelectromechanical systems.

Ying Liu is currently an Associate Professor (Senior Lecturer) in the Institute of Mechanical and Manufacturing Engineering at the School of Engineering in Cardiff University. He obtained his Bachelor's and Master's degrees in mechanical engineering from Chongqing University and MS and PhD from the Innovation in Manufacturing Systems and Technology program under the Singapore MIT Alliance at Nanyang Technological University and National University of Singapore, respectively. His research interests focus primarily on design informatics, manufacturing informatics, intelligent manufacturing, design methodology and process, product design, and information and communications technology in design and manufacturing. He is an Associate Editor of the *Journal of Industrial and Production Engineering* and is on the Editorial Board of *Advanced Engineering Informatics*.

Xiaolian Qin obtained his BS in computer science and technology from Jilin University and is currently pursuing his MS in computer science and technology at the State Key Lab of CAD&CG of Zhejiang University. His research is primarily in geometry modeling, and his other interests are in design rationale.