Searching Repositories of Web Application Models

Alessandro Bozzon, Marco Brambilla, and Piero Fraternali

Politecnico di Milano, Dipartimento di Elettronica e Informazione
P.za L. Da Vinci, 32. I-20133 Milano, Italy
{alessandro.bozzon,marco.brambilla,piero.fraternali}@polimi.it

Abstract. Project repositories are a central asset in software development, as they preserve the technical knowledge gathered in past development activities. However, locating relevant information in a vast project repository is problematic, because it requires manually tagging projects with accurate metadata, an activity which is time consuming and prone to errors and omissions. This paper investigates the use of classical Information Retrieval techniques for easing the discovery of useful information from past projects. Differently from approaches based on textual search over the source code of applications or on querying structured metadata, we propose to index and search the *models* of applications, which are available in companies applying Model-Driven Engineering practices. We contrast alternative index structures and result presentations, and evaluate a prototype implementation on real-world experimental data.

1 Introduction

Software repositories play a central role in the technical organization of a company, as they accumulate the knowledge and best practices evolved by skilled developers over years. Besides serving the current needs of project development, they have also an archival value that can be of extreme importance in fostering reuse and the sharing of high quality design patterns. With the spreading of open source software, project repositories have overcome the boundaries of individual organizations and have assumed a social role in the diffusion of coding and design solutions. They store billions of lines of code and are used daily by thousands of developers. State-of-the-practice project repositories mostly support source code or documentation search [4,10,14]. Several solutions are available, with different degrees of sophistication in the way in which queries are expressed, the match between the query and the indexed knowledge is determined, and results are presented. *Source code search engines* (e.g., Google code, Snipplr, Koders) are helpful if the abstraction level at which development occurs is the implementation code. However, searching project repositories at the source code level clashes with the goal of Model-Driven Engineering, which advocates the use of models as the principal artefact to express solutions and design patterns. Therefore, the question arises of what tools to use to leverage the knowledge implicitly

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 1–15, 2010.

stored in repositories of models, to make them play the same role in disseminating modeling best practices and foster design with reuse as code repositories had in fostering implementation-level best practices and code reuse. Approaches to model-driven repository search have been recently explored in the fields of business process discovery [3,5,12] and UML design [9,17].

The problem of searching model repositories can be viewed from several angles: the language for expressing the user's query (natural language, keywords, structured expressions, design patterns); the granularity at which result should be presented (full project, module or package, design diagram, design pattern, individual construct); the criteria to use in computing the match between the query and the model and for ranking results; the kind of metadata to collect and incorporate in the index (manually provided or automatically extracted).

The goal of this paper is to investigate solutions for making project repositories searchable *without requiring developers to annotate artifacts for the sole purpose of search*. The key idea is to exploit the structural knowledge embedded within application models, which can be expressed at a variable degree of abstraction (from Computation Independent, to Platform Independent, to Platform Specific). We concentrate on Platform Independent Models, because they describe the application, and not the problem, and are independent of the implementation technology. The contribution of the paper can be summarized as follows: 1) we introduce the notion of *model-driven project information retrieval system*, as an application of the information retrieval (IR) techniques to project repository search; 2) we identify the relevant design dimensions and respective options: project segmentation, index structure, query language and processing, and result presentation; 3) we implement an architecture for automatic model-driven project segmentation, indexing and search, which does not require the manual annotation of models; 4) we evaluate the approach using 48 industrial projects provided by a company, encoded with a Domain Specific Language.

The paper is organized as follows: Section 2 discusses the related work; Section 3 introduces the architecture of the model-driven project information retrieval system; Section 4 classifies the main design decisions; Section 5 illustrates the implementation experience; Section 6 presents the results of a preliminary performance assessment and user evaluation; finally, Section 7 draws the conclusions and discusses the ongoing and future work.

2 Related Work

Before illustrating the proposed solution, we overview the state-of-the-art in searchable project repositories, to better highlight the current limitations and original contribution of the work.

Component Search. Retrieval of software components, intended as mere code artifacts or as annotated pieces of software, is a well established discipline. The first proposals date back to the '90s. Agora [23] is a search engine based on JavaBeans and CORBA technologies that automatically generates and indexes a worldwide database of software artifacts, classified by component model. In the context of

SOA, Dustdar *et al.* [22] propose a method for the discovery of Web services based on a Vector Space Model characterization of their properties, indexed in a search engine. The work in [15] proposes a graph-representation model of a software component library, based on analyzing actual usage relations of the components and propagating a significance score through such links. The approach proposed in [7] combines formal and semi-formal specification to describe behaviour and structure of components, so as to make the retrieval process more precise.

Source Code Search. Several communities and on-line tools exist for sharing and retrieving code, e.g., *Google code*, *Snipplr*, *Koders*, and *Codase*[1].

In the simplest case, keyword queries are directly matched to the code and the results are the exact locations where the keyword(s) appear in the matched code snippets. Several enhancements are possible: 1) using more expressive query languages, e.g., regular expressions (in Google Codesearch) or wildcards (in Codase); restricting search to specific syntactical categories, like class names, method invocations, variable declarations, and so on (e.g., in Jexamples and Codase); restricting keyword search using a fixed set of metadata (e.g., programming language, license type, file and package names). Another dimension concerns how the relevance of the match is computed and presented to the user; the spectrum of solutions goes from the minimal approaches that simply return a list of hits without a meaningful ranking, to classical IR-style ranking based on term importance and frequency (e.g., TF/IDF), to composite scores taking into account both inherent project properties, e.g., number of matches in the source code, recency of the project, and social aspects, e.g., number of downloads, activity rates, and so on; for example, in SourceForge, one can rank results based on relevance of match, activity, date of registration, recency of last update, or on a combination calculated from such partial scores, and the system can account precisely for the rank value of each project over time.

Research works have applied IR techniques [10] and structural context techniques [14] for improving productivity and reuse of software. For example, the Sourcerer Project [4] provides an infrastructure for large-scale indexing and analysis of open source code that takes advantage of code structural information.

Model Search. Some approaches have addressed the problem of searching UML models. Early works exploited the XML format for indexing seamlessly UML models, text files, and other sources [11]. The work [13] stores UML artifacts in a central knowledge base, classifies them with WordNet terms and extracts relevant items exploiting WordNet classification and Case-Based Reasoning. The paper [17] proposes a retrieval framework allowing designers to retrieve information on UML models based on XMI representation through two query modalities: inclusion and similarity. Schemr [9] implements a novel search algorithm, based on a combination of text search and schema matching techniques, as well as a structurally-aware scoring methods, for retrieving database conceptual models with queries by example and keyword-based. Another branch of research applies

[1] Sites: `http://code.google.com`, `http://www.snipplr.com`, `http://www.koders.com`, `http://www.codase.com`

IR techniques to models and code together, for tracing the association between requirements, design artifacts, and code [24] [2].

Business Process Model Search. Several proposals have attempted to facilitate the discovery of business process models. Most of the approaches only apply graph-based comparison or XML-based querying on the business process specifications: Eyal *et al.* [5] proposed BP-QL, a visual query language for querying and discovering business processes modelled using BPEL. Lu and Sadiq [18] propose a way for comparing and retrieving business process variants. WISE [25] is a business process search engine that extracts workflow models based on keyword matching. These proposals offer a query mechanism based on the process model structure (i.e., the workflow topology) only. Other approaches adopt semantic-based reasoning and discovery: Goderis *et al.* [12] developed a framework for discovering workflows using similarity metrics that consider the activities composing the workflows and their relationships, implementing a ranking algorithm. [20] proposed a framework for flexible queries on BP models, for providing better results when too few processes are extracted. [3] proposes the BPMN-Q query language for visual semantic queries over BPMN models. Kiefer *et al.* [16] proposed the use of semantic business processes to enable the integration and inter-operability of business processes across organizational boundaries. They offer an imprecise query engine based on iSPARQL to perform the process retrieval task and to find inter-organizational matching at the boundaries between partners. Zhuge *et al.* [26] proposes an inexact matching approach based on SQL-like queries on ontology repositories. The focus is on reuse, based on a multi-valued process specialization relationship. The similarity of two workflow processes is determined by the matching degrees of their corresponding subprocesses or activities, exploiting ontological distances. The work [6] proposes a query by example approach that relies on ontological description of business processes, activities, and their relationships, which can be automatically built from the workflow models themselves.

Contribution. The approach described in this paper falls into the category of model-based search solutions, where it brings several innovations: (i) it automatically extracts the semantics from the searched conceptual models, without requiring manual metadata annotation; (ii) it supports alternative index structures and ranking functions, based on the language concepts; (iii) it is based on a model-independent framework, which can be customized to any DSL metamodel; (iv) it has been subjected to a preliminary evaluation on the relative performance and quality of alternative design dimensions.

3 IR Architecture Overview

Applying information retrieval techniques over model repositories requires an architecture for processing content, building up the required search indexes, matching the query to the indexed content, and presenting the results. Figure 1 shows the reference architecture adopted in this paper and the two main information flows: the *content processing flow* and the *query flow*.

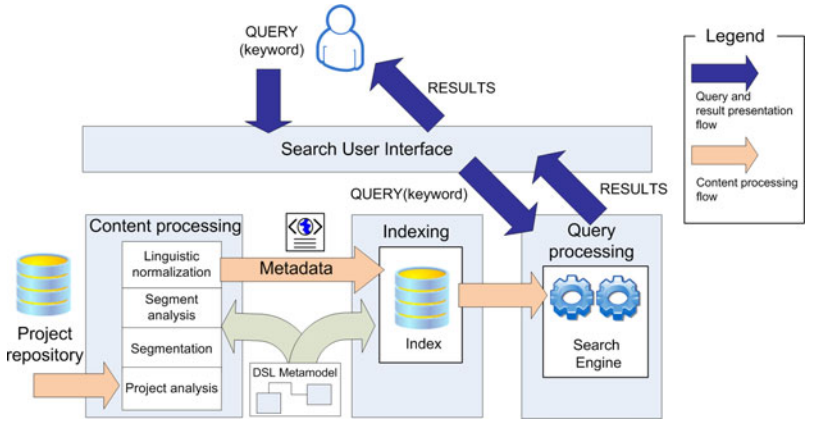


Fig. 1. Architecture of a content processing and search system

The *Content Processing Flow* extracts meaningful information from projects and uses it to create the search engine index. First, the *Content Processing* component analyzes each project by applying a sequence of steps: *project analysis* captures project-level, global metadata (e.g., title, contributors, date, and so on) useful for populating the search engine indexes; *segmentation* splits the project into smaller units better amenable to analysis, such as sub-projects or diagrams of different types; *segment analysis* mines from each segment the information used to build the index (e.g., model elements' names and types, model element relationships, designers's comments); *linguistic normalization* applies the text normalization operations typical of search engines (e.g., stop-word removal, stemming, etc. [19]) to optimize the retrieval performance. The information extracted from each project or segment thereof is physically represented as a *document*, which is fed to the *Indexing* component, for constructing the search engine indexes. Note that the metamodel of the DSL used to express the projects is used both in the Content Processing and in the Indexing components: in the former, it drives the model segmentation granularity and the information mining from the model elements; in the latter, it drives the definition of the index structure. For instance, the search engine index might be composed of several sub-indexes (called *fields*), each one devoted to a specific model element, so that a keyword query can selectively match specific model concepts.

The *query and result presentation flow* deals with the queries submitted by the user and with the production of the result set. Two main query modalities can be used: *Keyword Based*, shown in Figure 1, which simply looks for textual matches in the indexes, and *Content Based* (also known as Query by Example), not shown in in Figure 1, whereby a designer submits a model fragment as query, and the system extracts from it the relevant features (by applying the content processing flow) and matches them to the index using a given similarity criteria (e.g., text matching, semantic matching, graph matching).

4 Design Dimensions of Model-Driven Project Retrieval

The design space of a Model Driven Project IR System is characterized by multiple dimensions: the transformations applied to the models before indexing, the structure of the indexes, and the query and result presentation options.

Segmentation Granularity. An important design dimension is the **granularity** of indexable documents, which determines the atomic unit of retrieval for the user. An indexable document can correspond to:

- A whole design project: in this case, the result set of a query consists of a ranked list of projects.
- A subproject: the result set consists of ranked subprojects and each subproject should reference the project it belongs to.
- A project concept: each concept should reference its project and the concepts it relates to. The result set consists of ranked concepts, possibly of different types, from which other related concepts can be accessed.

Index structure. The structure of the index constructed from the models represents a crucial design dimension. An index structure may consists of one or more fields, and each field can be associated with an importance score (its weight). The division of the index into fields allows the matching procedure used in query processing to match in selected field, and the ranking algorithm to give different importance to matches based on the field where they occur.

The options that can be applied are the following:

- **Flat:** A simple list of terms is extracted from the models, without taking into account model concepts, relationships, and structure. The index structure is single-fielded, and stores undifferentiated bags of words. This option can be seen as a baseline, extracting the minimal amount of information from the models and disregarding any structure and semantics associated with the employed modeling language.
- **Weighted:** Terms are still extracted as flat lists, but model concepts are used in order to modify the weight of terms in the result ranking, so to give a significance boost to terms occurring in more important concepts. The index is single-fielded and stores weighted bags of words.
- **Multi-field:** Terms belonging to different model concepts are collected into separate index fields. The index is multi-fielded, and each field can be searched separately. This can be combined with the weighted approach, so as to produce a multi-field index containing weighted terms. The query language can express queries targeted to selected fields (e.g., to selected types of concepts, diagrams, etc).
- **Structured:** The model is translated into a representation that reflects the hierarchies and associations among concepts. The index model can be semi-structured (XML-based) or structured (e.g., the catalog of a relational database). Query processing can use a structured query language (e.g., SQL), coupled with functions for string matching into text data (e.g., indices for text objects).

Moving from flat to structured index structures augments the fidelity at which the model structure is reflected into the index structure, at the price of a more complex extraction and indexing phase and of a more articulated query language.

Query Language and Result Presentation. An IR system can offer different query and result visualization options. In the context of software model retrieval, the modalities that can be envisioned are:

- **Keyword-based search:** The user provides a set of keywords. The system returns results ranked according to their relevance to the input keywords.
- **Document-based search:** The user provides a document (e.g., a specification of a new project). The system analyzes the document, extracts the most significant words and submits them as a query. Results are returned as before.
- **Search by example:** The user provides a model as a query. The model is analyzed in the same way as the projects in the repository, which produces a document to be used as a query[2]. The match is done between the query and the project document and results are ranked by similarity.
- **Faceted search:** The user can explore the repository using *facets* (i.e., property-value pairs) extracted from the indexed documents, or he can pose a query and then refine its results by applying restrictions based on the facets present in the result set.
- **Snippet visualization:** Each item in the result set can be associated with an informative visualization, where the matching points are highlighted in graphical or textual form.

The abovementioned functionalities can compose a complex query process, in which the user applies an initial query and subsequently navigates and/or refines the results in an exploratory fashion.

Use Cases and Experiments. Although several combinations could be assembled from the above dimensions, we evaluate two representative configurations, compared to a baseline one. As reported in Section 6, the following scenarios have been tested: Experiment A (baseline): keyword search on whole projects; Experiment B: retrieval of subprojects and concepts with a flat index structure; Experiment C: retrieval of subprojects and concepts with a weighted index structure. Experiment B and C represent two alternative ways of structuring the index, both viable for responding to designer's query targeted at relevant subproject retrieval and design pattern reuse. Table 1 summarizes the design options and their coverage in the evaluated scenarios. In this work we focus on *flat* and *weighted* index structures. *Mulit-field* and *Structured* indexes will addressed in the future work.

[2] Here the term *document* means by extension any representation of the model useful for matching, which can be a bag of words, a feature vector, a graph, and so on.

Table 1. Summary of the design options and their relevance in the experiments

Option	Description	A	B	C
Segmentation Granularity				
Project	entire project	X		
Subproject	subproject		X	X
Single Concept	arbitrary model concepts		X	X
Index structure				
Flat	flat lists of words	X		
Weighted	words weighted by the model concepts they belong to			X
Multi-field	words belonging to each model concept in separate fields			
Structured	XML representation reflecting hierarchies and associations			
Query language and result presentation				
Keyword-based	query by keywords	X	X	X
Document-based	query through a document			
By example	query through a model (content-based)			
Faceted	query refined through specific dimensions	X	X	X
Snippets	visualization and exploration of result previews	X	X	X

5 Implementation Experience

To verify our approach we developed a prototype system that, given a meta-model and a repository of models conforming to such meta-model: 1) configures a general purpose search engine according to selected dimensions, 2) exploits metamodel-aware extraction rules to analyze models and populate the index with information extracted from them; 3) provides a visual interface to perform queries and inspect results.

The experiments adopt WebML as a Domain Specific Language [8]. The WebML metamodel [21] specifies the constructs for expressing of the data, business logics, hypertext interface, and presentation of a Web application. In WebML, content objects are modeled using Entity-Relationship or UML class diagrams. Upon the same content model, it is possible to define different *application models* (called *site views*), targeted to different user roles or access devices. Figure 2 (a) depicts an excerpt of the WebML meta-model describing the siteview construct. A site view is internally structured into *areas*, which in turn may contain *pages*. Pages comprise *content units*, i.e., components for publishing content, and units are connected to each other through *links*, which carry parameters and allow the user to navigate the hypertext. WebML also allows specifying *operations* implementing arbitrary business logic (e.g., to manipulate content instances), and *Web service* invocation and publishing. Finally, the language comprises a notion of *module*, which denotes a reusable model pattern. Figure 2 (b) presents a sample Web application that implements a product catalog, where users can see a list of product and select one for getting more details. Figure 2 (c) shows the corresponding XML project file encoding the model.

5.1 Content Processing

Content processing has been implemented according to the schema of Figure 1. The *segmentation* and *text-extraction* steps are implemented in a generic and model-independent way; they are configurable by means of model transformation

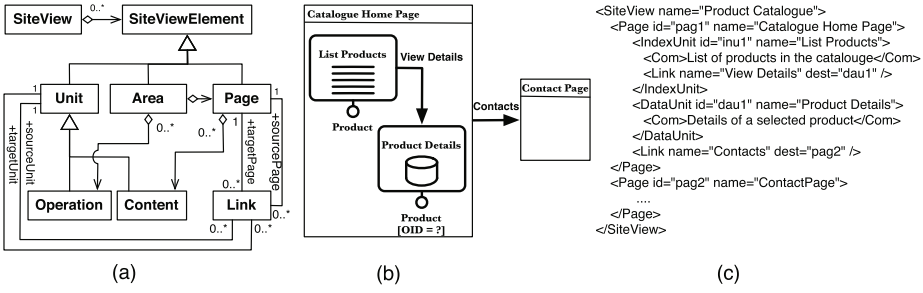


Fig. 2. a) Excerpt of the WebML metamodel. b) Example of WebML model. c) XML representation of the WebML model depicted in b).

rules encoded in XSLT, so to be adapted to the DSL of choice. These rules simply match each metamodel concept and decide what information to extract for populating the index. An auxiliary *Repository Analysis* component has been added to the architecture, which performs the offline analysis of the entire project collection to compute statistics for fine-tuning the retrieval and ranking performance: 1) a list of *Stop Domain Concepts*, i.e., words very common in the project repository (e.g., the name of meta-model concepts or terms that are part of the organization vocabulary and culture); 2) the *Weight* assigned to each model concept; presently, concepts weights are computed automatically based on the relative frequency of model concepts in the entire collection, but can be adjusted manually by the search engine administrator to promote or demote individual concepts.

Content processing has been implemented by extending the text processing and analysis components provided by Apache Lucene[3], an open-source search engine platform.

5.2 Index Structure and Ranking Function Design

Different index structures have been defined to cope with the evaluation scenarios: Experiment A and B have a flat index structure; Scenario C instead employs a weighed multi-field index structure, with the following fields: `id | projectID | projectName | documentType | text`, where the documentType field can have the values: DataModel, SiteView, Area, ServiceView, and Module, which represent the fundamental modularization constructs of WebML.

The relevance of the match is also computed differently in the scenarios. Experiment A uses the pure TF/IDF measure for textual match relevance [19], ignoring model structure; experiment B also uses traditional TF/IDF relevance, but due to segmentation the matching is performed separately on the different model concepts; finally, Experiment C exploits the model concepts also in the ranking function, which is defined as follows:

$$score(q,d) = \sum_{t \in q} \sqrt{tf(t,d)} \cdot idf(t)^2 \cdot mtw(m,t) \cdot dw(d) \qquad (1)$$

[3] `http://lucene.apache.org/java/docs/`

where:

- $tf(t,d)$ is the *term frequency*, i.e., the number of times the term t appears in the document d;
- $idf(t)$ is the *inverse document frequency* of t, i.e., a value calculated as $1 + \log \frac{|D|}{freq(t,d)+1}$ that measures the informative potential carried by the term in the collection;
- $mtw(m,t)$ is the *Model Term Weight* of a term t, i.e., a metamodel-specific boosting value that depends on the concepts m containing the term t. For instance, in the running example, a designer can decide that the weight of terms t in a *module* should be double than all the other constructs. Then he will set $mtw(module,t)$ to 2.0 and mtw of all the others concepts to 1.0;
- $dw(d)$ is the *Document Weight*, i.e., a model-specific boosting value conveying the importance of a given project segment (i.e., project, sub-project, or concept, depending on the chosen segmentation policy) in the index.

The scoring function is implemented by extending the APIs of Lucene, which support both the Boolean and the Vector Space IR Models, thus enabling complex Boolean expressions on multiple fields.

5.3 Query and Result Presentation

Figure 3 shows the UI of the search system, implemented as a rich Web application based on the YUI Javascript library. The interface lets users express Boolean keyword queries, e.g., expressions like *"validate AND credit AND NOT card"*. Upon query submission, the interface provides a paginated list of matching items (A); each item is characterized by a a type (e.g., model elements like data model, area, module, etc.) and by the associated project metadata (e.g. project name, creation date, authors, etc.). The *Inspect Match* link opens a *Snippet* window (B) that shows all the query matches in the current project/subproject/construct, allowing users to determine which model fragments are useful to their information need. Users can also drill down in the result list by applying faceted navigation. The available facets are shown in the left-hand side of Figure 3. Each facet contains automatically extracted property values of the retrieved results. The selection of a facet value triggers a refinement of the current result set, which is restricted only to the entries associated to the facet value.

6 Evaluation

Evaluation has addressed the space and time efficiency of the system and the *designers' perception* of the results quality and of the usefulness of the proposed tool.

Experimental settings and dataset. A sample project repository has been provided by Web Models [1], the company that develops WebRatio, an MDD tool for WebML modeling and automatic generation of Web applications. The repository contains *48* large-size, real-word projects spanning several applications domains (e.g., trouble ticketing, human resource management, multimedia search engines, Web portals, etc.). Projects have been developed both in

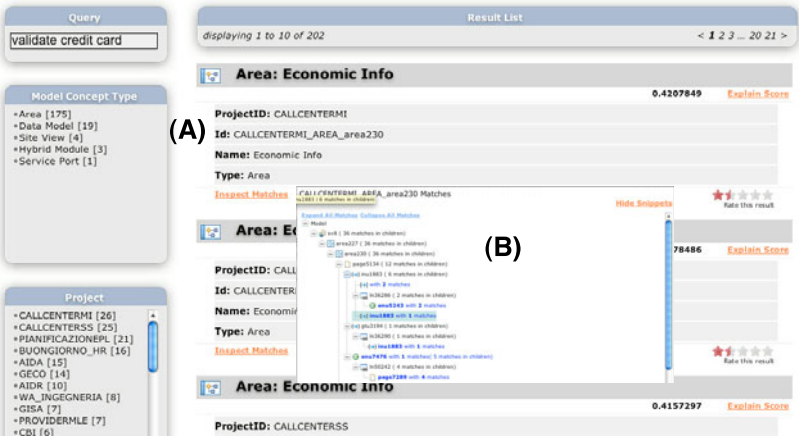


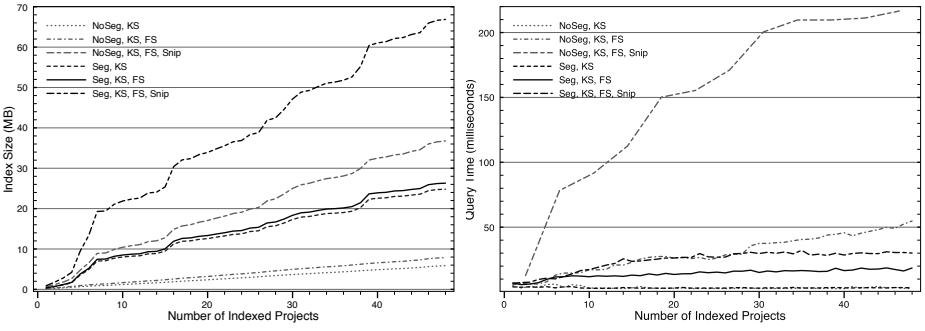
Fig. 3. Project repository search UI: result list(A) and result exploration(B)

Italian and English, and comprise about *250* different modeling concepts (resulting from WebML standard constructs and user-defined modeling elements), *3,800* data model entities (with about *35,000* attributes and *3,800* relationships), *138* site views with about *10,000* pages and *470,000* units, and *20* Web services. Each WebML project is encoded as an XML project file conforming with the WebML DTD, and the overall repository takes around 85MB of disk space. The experiments were conducted on a machine equipped with a 2GHz Dual-Core AMD Opteron and 2GB of RAM. We used Apache Solr 1.4 (`http://lucene.apache.org/solr/`) as Lucene-based search engine framework.

Performance Evaluation. Performance experiments measured index size and query response time, varying the number of indexed projects. Two scenarios were used: the baseline, in which whole projects are indexed as documents with no segmentation, and a segmentation scenario in which selected model concepts are indexed as separate documents. For both scenarios, three alternative configurations have been evaluated (keyword search, faceted search, and snippet browsing) so to exercise all *query language and result presentation* options reported in Table 1.

Figure 4 (a) shows the results of evaluating index size. For each of the six system configurations, all project in the repository are progressively indexed, disabling any compression mechanism in the search engine index. Size grows almost linearly with the number of projects in all configurations, thus showing good scalability. As expected, the basic *keyword search* scenario has the least space requirements in both the non-segmented and segmented configurations (about 10 times smaller than the repository size)[4]. The addition of *Faceted Search*, doubles the index size, because the original text of facet properties must

[4] Higher values at the beginning of the curves can be explained by low efficiency with small index sizes.



(a) Index Size (t) Query Time

Fig. 4. Index size (a) and response time (b) of six system configurations (*NoSeg*: No Segmentation, *Seg*: Segmentation, *KS*: Keyword Search, *FS*: Faceted Search, *Snip*: Snippet)

be stored in the index to enable its visualization in the UI. *Snippet Visualization* is the most expensive option, especially when models are segmented and thus the number of indexed documents grows.

Figure 4 (b) shows the query response time with a varying number of indexed projects. We selected about 400 2-terms and 3-terms keyword queries, randomly generated starting from the most informative terms indexed within the repository. Each query has been executed 20 times and execution times have been averaged. Under every configuration, query time is abundantly sub-second, and curves indicate a sub-linear growth with the number of indexed projects, thus showing good scalability. Notice that the addition of *Faceted Search* and *Snippet Visualization* impacts on performances also for query latency time; differently from the previous evaluation, though, the most affected scenario is the one where no project segmentation is applied, thus introducing a penalty caused by the greater amount of information that needs to be retrieved for each query result item.

Preliminary User Evaluation. A preliminary user study has been conducted with 5 expert WebML designers to assess 1) the perceived retrieval quality of alternative configurations and 2) the usability and ease-of-use of the system.

The perceived retrieval quality has been tested with the three system configurations listed in Table 1, using a a set of ten queries manually crafted by the company managers responsible of the Web applications present in the repository. Designers could access the three system configurations and vote the appropriateness of each one of the top-10 items in the result set; they were asked to take into account both the element relevance with respect to the query and its rank in the result set. Votes ranged from 1 (highly inappropriate) to 5 (highly appropriate). Figure 5 shows the distribution of the 500 votes assigned to each system configuration. Experiment B and C got more votes in high (4-5) range

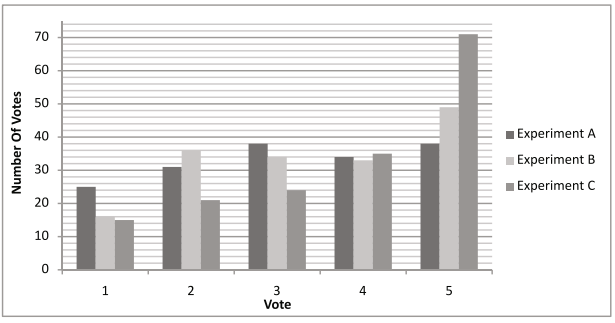


Fig. 5. Average vote distribution for the test queries

Table 2. User Evaluation: questionnaire items, average rates and variance

Item	Avg.	Var.
Features		
Keyword Search	3.6	0.24
Search Result Ranking	3.2	0.16
Faceted Search	3.8	0.16
Match Highlighting	3.6	0.24
Application		
Help reducing maintenance costs?	3.2	0.56
Help improving the quality of the delivered applications?	3.0	0.4
Help understanding the model assets in the company?	4.4	0.24
Help providing better estimates for future application costs?	2.8	0.56
Wrap-up		
Overall evaluation of the system	4.0	0.4
Would you use the system in your activities?	3.0	1.2

of the scale with respect to the baseline. Also, results were considered better in configuration C, which assigned weights to the terms based on the model concept they belonged to. The penalty of Configuration A can be explained by the lower number of the indexed documents (48), which may brings up in the top ten more irrelevant projects, having a relatively low score match with the query.

The study also included a questionnaire to collect the user opinion on various aspects of the system. Table 2 reports the questions and the collected results, in a 1 to 5 range. The results show that the system, although very prototypical, has been deemed quite useful for model maintenance and reuse, while its direct role in improving the quality of the produced applications is not perceivable. One could notice a certain distance between the overall judged quality and the likelihood of adoption, which is possibly due to three aspects: 1) such a tool would probably be more valuable in case of a very large corpus of projects; 2) UI usability is greatly hampered by the impossibility at present of displaying the matches in the WebRatio diagram editor; and 3) developers are not familiar with search over a model repository and would probably need some experience before accepting it in their everyday work life.

While our evaluation is insufficient to entail statistical relevance, we believe that the proposed user study suffices to support the motivations of our work and provides useful feedbacks to drive future research efforts.

7 Conclusions

In this paper we presented an approach and a system prototype for searches over model repositories. We analyzed the typical usage scenarios and requirements and we validated our claims with an implementation over a repository of Web application models specified in WebML. The experimental evidence show that the system scales well both in size and query response time; a preliminary user evaluation showed that the ranking of results based on a priori knowledge on the metamodel elements is gives better results with respect to the baseline solution of flat indexing of the text content of a project. Ongoing and future work includes the implementation of content-based search, the integration of the search interface in the diagram editing GUI of WebRatio, for visually highlighting the matches in the projects, the capture of the user feedback on the top-ranking results, to automatically learn how to fine-tune the model weights and improve precision and recall, a larger scale study on the scalability and effectiveness of the retrieval system, not limited to WebML models but also to UML artifacts, and the definition of benchmark criteria for model-driven repository search.

References

1. Acerbis, R., Bongio, A., Brambilla, M., Butti, S.: Webratio 5: An eclipse-based case tool for engineering web applications. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 501–505. Springer, Heidelberg (2007)
2. Antoniol, G., Canfora, G., de Lucia, A., Casazza, G.: Information retrieval models for recovering traceability links between code and documentation. In: IEEE International Conference on Software Maintenance, p. 40 (2000)
3. Awad, A., Polyvyanyy, A., Weske, M.: Semantic querying of business process models. In: Enterprise Distributed Object Computing Conference (EDOC), pp. 85–94 (2008)
4. Bajracharya, S., Ossher, J., Lopes, C.: Sourcerer: An internet-scale software repository. In: ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation. SUITE '09, pp. 1–4 (May 2009)
5. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: VLDB, pp. 343–354. ACM, New York (2006)
6. Belhajjame, K., Brambilla, M.: Ontology-based description and discovery of business processes. In: Interval Mathematics. LNBIP, vol. 29. Springer, Heidelberg (2009)
7. Ben Khalifa, H., Khayati, O., Ghezala, H.: A behavioral and structural components retrieval technique for software reuse. In: Advanced Software Engineering and Its Applications. ASEA 2008, pp. 134–137 (December 2008)
8. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers Inc., San Francisco (2002)

9. Chen, K., Madhavan, J., Halevy, A.: Exploring schema repositories with schemr. In: SIGMOD '09: Proc. of the 35th SIGMOD Int. Conf. on Management of data, New York, NY, USA, pp. 1095–1098. ACM, New York (2009)
10. Frakes, W.B., Nejmeh, B.A.: Software reuse through information retrieval. SIGIR Forum 21(1-2), 30–36 (1987)
11. Gibb, F., McCartan, C., O'Donnell, R., Sweeney, N., Leon, R.: The integration of information retrieval techniques within a software reuse environment. Journal of Information Science 26(4), 211–226 (2000)
12. Goderis, A., Li, P., Goble, C.A.: Workflow discovery: the problem, a case study from e-science and a graph-based solution. In: ICWS, pp. 312–319. IEEE Computer Society, Los Alamitos (2006)
13. Gomes, P., Pereira, F.C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J.L., BentoI, C.: Using wordnet for case-based retrieval of uml models. AI Communications 17(1), 13–23 (2004)
14. Holmes, R., Murphy, G.C.: Using structural context to recommend source code examples. In: ICSE '05: Proceedings of the 27th international conference on Software engineering, pp. 117–125. ACM, New York (2005)
15. Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., Kusumoto, S.: Ranking significance of software components based on use relations. IEEE Transactions on Software Engineering 31(3), 213–225 (2005)
16. Kiefer, C., Bernstein, A., Lee, H.J., Klein, M., Stocker, M.: Semantic process retrieval with iSPARQL. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 609–623. Springer, Heidelberg (2007)
17. Llorens, J., Fuentes, J.M., Morato, J.: Uml retrieval and reuse using xmi. In: IASTED Software Engineering. Acta Press (2004)
18. Lu, R., Sadiq, S.: Managing process variants as an information resource. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 426–431. Springer, Heidelberg (2006)
19. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (July 2008)
20. Markovic, I., Pereira, A.C., Stojanovic, N.: A framework for querying in business process modelling. In: Multikonferenz Wirtschaftsinformatik (February 2008)
21. Moreno, N., Fraternali, P., Vallecillo, A.: Webml modelling in uml. IET Software 1(3), 67–80 (2007)
22. Platzer, C., Dustdar, S.: A vector space search engine forweb services. In: ECOWS '05: Proceedings of the Third European Conference on Web Services, Washington, DC, USA, pp. 62–71. IEEE Computer Society, Los Alamitos (2005)
23. Seacord, R.C., Hissam, S.A., Wallnau, K.C.: Agora: A search engine for software components. IEEE Internet Computing 2(6), 62–70 (1998)
24. Settimi, R., Cleland-Huang, J., Ben Khadra, O., Mody, J., Lukasik, W., DePalma, C.: Supporting software evolution through dynamically retrieving traces to uml artifacts. In: Proceedings of 7th International Workshop on Principles of Software Evolution, pp. 49–54 (2004)
25. Shao, Q., Sun, P., Chen, Y.: Wise: A workflow information search engine. In: IEEE 25th International Conference on Data Engineering. ICDE '09, pp. 1491–1494 (2009)
26. Zhuge, H.: A process matching approach for flexible workflow process reuse. Information & Software Technology 44(8), 445–450 (2002)