Textual and Content-Based Search in Repositories of Web Application Models

BOJANA BISLIMOVSKA,
Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
ALESSANDRO BOZZON,
Delft University Of Technology, Software And Computer Technology Department
MARCO BRAMBILLA,
Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
PIERO FRATERNALI,
Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)

Model-Driven Engineering relies on collections of models, which are the primary artefacts for software development. To enable knowledge sharing and reuse, models need to be managed within repositories, where they can be retrieved upon users queries. This paper examines two different techniques for indexing and searching model repositories, with a focus on Web development projects encoded in a Domain Specific Language. Keyword-based and content-based search (also known as query-by-example) are contrasted, with respect to the architecture of the system, the processing of models and queries, and the way in which meta-model knowledge can be exploited to improve search. A thorough experimental evaluation is conducted to examine what parameter configurations lead to better accuracy and to offer an insight in what queries are addressed best by each system.

Categories and Subject Descriptors: D.2.13 [**Reusable Software**]: Reuse models; H.3.3 [**Information Search and Retrieval**]: Search process

General Terms: Algorithms, Experimentation, Design, Performance

Additional Key Words and Phrases: Information Retrieval, Web Application, Search, Domain Specific Language

ACM Reference Format:
Bislimovska, B., Bozzon, A., Brambilla, M., Fraternali P. 2013.Textual and Content-Based Search in Repositories of Web Application Models. ACM Trans. Web V, N, Article A (January YYYY), 49 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

The increased complexity and pervasiveness of software requires raising the level of abstraction, and automating labor–intensive and error-prone tasks to increase efficiency and effectiveness in software development [Mohagheghi and Dehlen 2008].

An approach that advocates software abstraction through the use of models is Model-Driven Engineering (MDE), widely used in academia and industrial organizations

This work is partially supported by the FP7 Cubrik Integrating Project (`http://www.cubrikproject.eu/`) and by the Capacities Research for SMEs project BPM4People of the Research Executive Agency of the European Commission (`http://www.bpm4people.org/`)
Author's addresses: B. Bislimovska, M. Brambilla and P. Fraternali, Dipartimento di Elettronica, Informazionee Bioingegneria (DEIB), Politecnico di Milano. P.zza Leonardo Da Vinci 32, 20133, Milano, Italy; A. Bozzon, Web Information Systems, Delft University of Technology, Mekelweg 4, 2628 Delft, The Netherlands

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

across different domains. MDE promotes the use of models in any engineering activity as abstractions that provide a simplified or partial representation of reality, useful to accomplish a task or to reach an agreement on a topic. Model-Driven Software Engineering specifically considers software models, i.e., abstractions of the static or dynamic properties of a software system. Studies demonstrate that the benefits of MDE in industry are perceived in terms of quickly responding to change of requirements, of streamlining communication among stakeholders [Mohagheghi and Dehlen 2008] thanks to more accessible organizational knowledge [Hutchinson et al. 2011], and of improving the quality of code design and test case development [Anda et al. 2006].

The adoption of MDE in academic and business organizations resulted in an increasing number of models collections, stored in *model repositories* [France et al. 2006]. To name but a few: the MIT Process Handbook [MIT 2012] contains over 5000 business process model entries; the AtlanMod Metamodel Zoos [AtlanMod Group 2012] provides a collection of more than three hundred metamodels; the ReMODD repository [ReMoDD Team 2012; France et al. 2012] is collecting case studies, models and metamodels in different modeling languages. In the industry, several MDE tool vendors provide repositories that contain application and component models authored with their tools: examples include the WebRatio Store [WebRatio s.r.l. 2012]; the Mendix App Store [Mendix 2012], the CodeCharge Studio marketplace [YesSoftware, Inc. 2012], the Genexus marketplace [Artech Consultores S.R.L. 2012], and the Outsystems AgileNetwork component repository [Outsystems Inc. 2012].

Reuse and sharing of software requires the ability of effectively retrieving artifacts that meet the user's need, which is the goal of *software search systems*. Besides the software repositories inside organizations, several on-line tools exemplify the state-of-the-art in sharing and retrieving code, e.g., *Google code*, *Snipplr*, *Koders*, and *Codase*[1]. In the simplest case, the user submits keywords, which are matched to the code, and receives as a response the programs that contain the search terms. Advanced systems offer more powerful functionality: 1) expressive query languages, e.g., regular expressions (in Google Codesearch) or wildcards (in Codase); search over syntactical categories, like class names, method invocations, and variables (e.g., in Jexamples and Codase); result restriction based on metadata (e.g., programming language, license type, file and package names). In the simplest systems the result set is a plain list of unranked hits, but more sophisticated interfaces offer classical IR-style ranking based on term importance and frequency or even composite scores compounding number of matches in the source code, recency of the project, number of downloads, activity rates, and so on; for example, in SourceForge users can receive results ranked by any combination of relevance of match, activity, date of registration, and recency of last update.

Model repositories are not yet as well developed and widespread as source code repositories. The latter enable code-level reuse and thus a reduction of development time and costs, and may improve software quality, as novice programmers can learn from the code produced by more experienced ones. The same advantages could be achieved in MDE, if repositories allowed for the efficient retrieval of models relevant to the user's needs. Most industrial repositories offer rather elementary interfaces, where users can only search by matching keywords against the model's description or explore the available content via taxonomical navigation and facets. More powerful approaches may foster early stage model reuse and promote the dissemination of modeling best practices across projects and development teams: for example, a developer wishing to implement a given application requirement may find in the company's repository models that solve similar tasks and reuse them entirely or some design

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

pattern embedded therein. Model search approaches should exploit in more depth the main difference between source code and models, that is, *the high level, structural, and often visual nature of a model representation*. Ideally, an MDE developer should be able not only to search models via keywords, but also to sketch the idea he has in mind in his favorite Domain Specific Language and retrieve all models that contain a similar design, properly ranked according to their relevance to the query. Therefore, similarity search techniques are essential to allow developers' needs be formulated in the same language in which solutions are expressed.

Model search is most useful during the initial phases of application development: the translation of software requirements into design artifacts and the transformation of coarse design models into detailed models. In the former case, requirements can be expressed concisely as keywords and used to find relevant models; in the latter case, coarse design models can be used for retrieving more detailed ones.

With the notable exception of Business Process Models repositories, where research has investigated similarity measures for the specific syntax and semantics of process models [Mendling et al. 2007; Niemann et al. 2012; Qiao et al. 2011; Dijkman et al. 2009], content-based search and multimodal search (e.g., keyword *plus* content based search) are still not the state-of-the-practice for model repositories, which sets the background for the research reported in this paper.

1.1. Motivating Example

To better motivate the need for search in model repositories, let us consider the scenario depicted in Figure 1: *Alice* is a developer in a company adopting MDE for the design and implementation of Web-based information systems. *Alice* is currently working on the development of a novel customer management system and she has to address the requirement of allowing authentication of users through the OAuth[2] protocol. Let's assume her company already had several experiences in the development of system exploiting open authentication techniques; therefore, the model repository contains some project where this specific functionality has been designed already. *Alice* might be or not be aware of such previous work, but the reuse of existing models or the adherence to modeling patterns used in previous successful projects will facilitate her job and improve uniformity of modeling style across the company. The goal of a model search system is to assist *Alice* in the retrieval of existing, similar solutions, and thus allow reuse and knowledge sharing.

Alice can express her information need with the textual query "authenticate user oauth". The search system looks up the repository and returns a list of results at the appropriate granularity, as shown in Figure 1. The result set comprises concise previews of the retrieved model fragments; for a better understanding, results are ordered according to their relevance to the query and the parts of the model that match the query are highlighted. *Alice* might want to zoom in and visualize one of the results to better inspect the matching parts, or open the fragment in its original context. If she finds something useful for her current task, she might import the matching parts or the entire model in her workspace.

1.2. Goals and Contributions

The goal of this paper is to study the implications of building search systems for software models expressed according to Domain Specific Languages, so to *increase the reuse of modeling artifacts and promote the discovery of existing design patterns and the application of modeling best practices from previous projects*. We study two different scenarios of model search: *keyword-based search*, which proceeds in continuity

[2]`http://www.oauth.net`

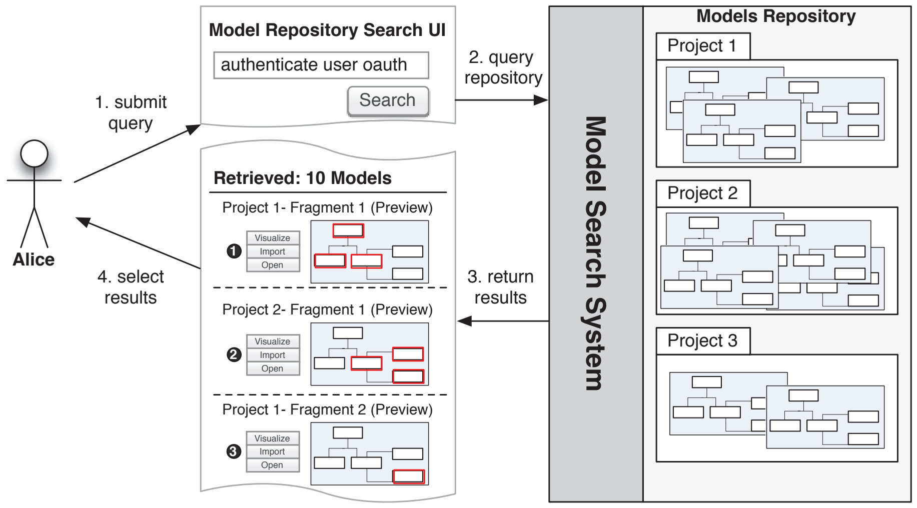


Fig. 1: Example of interaction between a developer and a search system for model repositories: the user submits her query (1), which is applied upon the repository (2); in turn, the repository returns the matching project fragments (3), among which the user can select the one that fits better the new requirements (4).

with classical Information Retrieval approaches and source code search techniques; and *content-based search*, which introduces the query-by-example paradigm into model search. The illustrated research aims at addressing the following questions:

Q.1 How can we search model repositories in order to unlock their hidden value and allow efficient reuse of models?
Q.2 How can we adapt text- and content-based search techniques to model repositories, so to exploit metamodel knowledge and improve the quality of results?
Q.3 How do text- and content-based search compare in terms of retrieval performance under different technical configurations of their characteristic parameters?
Q.4 How do users perceive the quality of results retrieved with text- and content-based search?

To address the above questions, the paper overviews the requirements of model search and investigates keyword-based and content-based techniques for search of model repositories. Keyword-based and content-based search techniques are extended with the injection of metamodel knowledge in the search process, to test its effect on retrieval performance. Two approaches (text- and content-based) are implemented, configured and technically evaluated on a real world collection of 341 industrial models, with a panel of 10 queries. Models in the experimental repository are encoded in the Web Modeling Language (WebML) [Ceri et al. 2003], a DSL for Web applications.

Performance of text- and content-based search are evaluated with two distinct experiments: a technical evaluation based on a gold standard defined by experts, which evaluates the quality of the matches, response time, and space occupation; and two user studies, which engaged 25 MDE practitioners in the subjective evaluation of utility and quality of search results. Different variants of the technical configurations of the two systems have been evaluated and compared. The user studies examine the relationship between the performance of the systems and the user-perceived utility of retrieved results.

The contributions of the paper can be summarized as follows:

C.1 We extend state-of-the-art methods for keyword-based search in order to incorporate metamodel-specific information. We show that augmenting the IR index with metamodel knowledge leads to a performance improvement with respect to conventional, metamodel-agnostic text-based IR techniques.
C.2 We implement content-based search by means of graph matching; to do so, we extend standard techniques for sub-graph isomorphism (the A-star algorithm) by considering a formulation of the matching score function that takes into account metamodel-specific information. We also investigate how the locality of the match between the query and the project graph affects performance.
C.3 We compare keyword-based and content-based search systems with respect to retrieval accuracy (precision and recall), ranking accuracy, and stability of the results across different queries, using a gold dataset created by experts.
C.4 We report the results of a user study that assesses how model-driven practitioners appreciate keyword-based and content-based search.

1.3. Outline

The rest of the paper is organized as follows: Section 2 presents the fundamentals of search over model repositories and thus responds to question [*Q.1*]; Section 3 focuses on the case study of search over Web application model repositories, thus addressing question [*Q.2*]; in particular, Section 3.1 introduces WebML, the DSL used as a case study; Section 3.2 discusses the architecture and configuration of the keyword-based search system; Section 3.3 focuses on the content-based search system; Section 4 presents the results of the experimental evaluation conducted on the keyword-based and on the content-based search systems, thus addressing questions [*Q.3*] and [*Q.4*]; Section 5 discusses the related work; finally, Section 6 highlights the conclusions and discusses the future work.

2. FUNDAMENTALS OF SEARCH FOR MODEL REPOSITORIES

In MDE, *models* are used to formalize requirements, structure, and behavior of the addressed system; they comply with the syntax of a *modeling language*, which can be formalized as a *metamodel* [Kleppe et al. 2003]. Each model *element* has a *type* (i.e., a higher order concept) defined in the metamodel, and is related to other elements by means of typed *relationships*, also defined in the metamodel. One or more concrete syntaxes can be associated to a metamodel. The syntax can be either textual or graphical and defines the way in which the models are represented concretely. Elements and relationships in models are typically enriched with textual labels, provided by the model developer to describe some relevant domain properties or functions of the concept. During the development process, models are typically organized into *projects*, i.e., logical containers that aggregate models and artifacts of the same system or application domain; likewise, projects developed by the same organization are collected in *repositories*.

Model repositories are accessed primarily through *search*, i.e., the retrieval of relevant artifacts upon the expression of a user's need.

2.1. Information Retrieval Techniques for Model Search

The search process can be schematically represented as the chain of four main steps, as shown in Figure 2.

Starting from a repository of projects, the *Content Processing* transforms each model into a format suitable for efficient indexing and effective search. The *Indexing* step

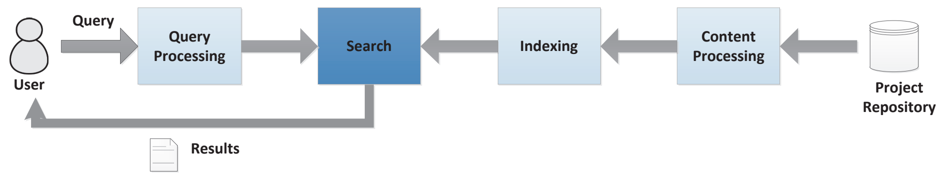


Fig. 2: Main steps of a model-driven search process.

stores the processed models into persistent data structures that contain information amenable to search, typically encoded as *index terms* or as *index data structures*.

Users express their information needs as queries defined in a given format. Among the available query paradigms, we focus on two specific forms: (i) **Keyword-based** queries (also called **text-based queries**) are expressed as bag of words; users translate their information need from their abstract representation (e.g., “find all the projects that model the shopping cart operations of a book e-commerce application”) into keywords or simplified phrases (e.g., “book shopping cart e-commerce”). (ii) **Content-based** queries are expressed as model fragments; users ask the system to “find a model like this” and thus formulate their queries in the same language in which the targeted models are expressed. This way of searching is also called **query-by-example**.

Queries are subjected to a *Query Processing* step, in which they undergo a transformation toward an *internal format*, which maps them to the same representation space as the index. For instance, a keyword query could be transformed into a set of stemmed words, or a model fragment could be mapped into a labeled graph.

The *Search* step inspects the index in order to (i) retrieve the models that match with the user query, (ii) rank the matching models according to their relevance with respect to the query, and (iii) return them to the user as a result set where more relevant hits are displayed in more prominent positions.

Figure 3 expands the view of the activities contained in each of the steps summarized in Figure 2. Activities can be *metamodel-dependent*, when they exploit knowledge defined in the metamodel (in Figure 3 they have an input data flow from the metamodel artifact), or *metamodel-independent*.

2.2. Content and Query Processing

The model search process, shown in Figure 3, requires both the repository projects and the queries to be properly analyzed to extract information relevant for indexing and searching. The *Query Processing* workflow comprises query analysis techniques that are the same as those for projects; therefore we can limit the explanation to the *Content Processing* tasks.

The *Project Analysis* task starts the analysis workflow by extracting general metadata, such as the project identifier in the collection, its name, authors, etc, useful for result presentation. The *Project Segmentation* activity splits each project into smaller units more suitable for analysis; the segmentation strategy is defined by the system designer, and can occur: (i) manually, by identifying project by project the most meaningful segmentation units; (ii) automatically based on metamodel-driven or collection-specific rules, which may take into account model types, concepts or relationship types, and element frequencies in the collection. For instance, UML class diagrams could be partitioned considering as segments the bottom elements of the package hierarchy.

Each segment is processed by a *Segment Analysis* task, which extracts relevant features for each model element contained in the segment, such as name, type, re-

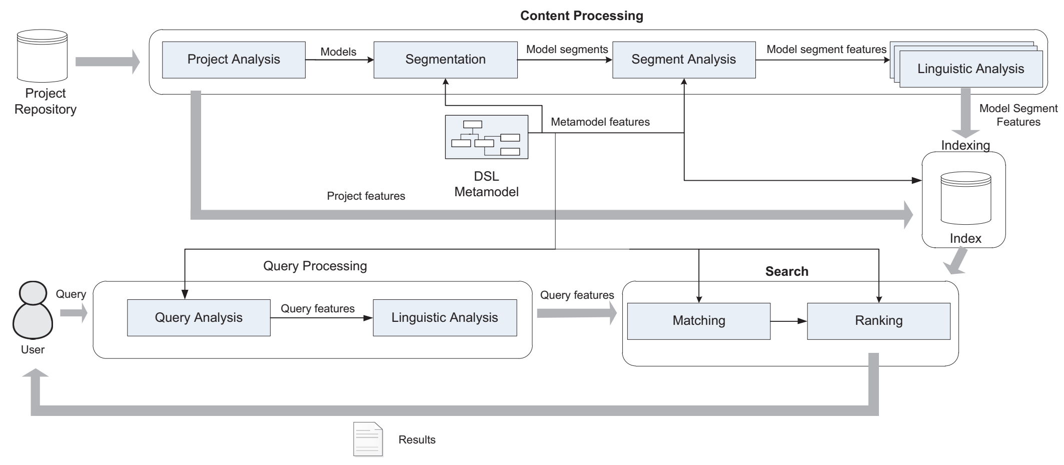


Fig. 3: Architecture of a model-driven information retrieval system.

lationships with other elements, or any other property defined in the metamodel and relevant for search purposes. The extracted textual features might be normalized by applying metamodel-independent *Linguistic Analysis* transformations (e.g., language translation, tokenization, stemming, stop-word removal, etc.).

2.2.1. Indexing. The normalized features extracted from each element are the inputs to the *Indexing* step, as *index documents* [Manning et al. 2008]. The *index* stores the project metadata, the segment-to-project mapping, and a representation of the extracted model element features, optimized for storage and search purposes. The index can be organized according to one of the following options:

— **Flat index**: the index is structured as a single field which stores all the extracted features of an index document. A flat index does not allow the representation of model relationships, as the model structure cannot be enforced.
— **Multi-field index**: the index is divided into multiple fields, each storing a different subset of the indexable information. Each field can be searched separately, i.e., query matching can be restricted to the selected fields. A multi-field index may be used to encode metamodel information, by associating each field to features (e.g., normalized words) appearing in a distinct model concept or relation. In this way, a query could be restricted only to selected model concepts. Furthermore, each index field can be assigned a *weight* that quantifies its importance according to some a priori knowledge (e.g., the significance of the metamodel concept associated with the field).
— **Structured index**: the index is organized as a (semi) structured document (e.g., mapping each segment to a graph or to an XML document) so as to preserve the relationships among model elements. Structural elements can be assigned a *weight* that quantifies their importance.

Orthogonally to the adopted index structure, terms can be assigned a *term weight* that reflects their significance. Increasing the index complexity, from flat to multi-field, to structured indexes, gives more precise representations of projects and queries, to the price of more complex storage structures, query language, and match algorithms.

2.2.2. Search. The search workflow consists of two tasks. The *Matching* task finds the documents in the index that match the internal representation of the user's query. The matching technique applied depends on the index structure. For flat and multi-field indexes, matching occurs by verifying the presence of query terms in the index; in structured indexes, matching verifies if the query internal representation is at least partially contained in an indexed segment.

The *Ranking* task sorts the found matches with respect to their relevance to the query, calculated as a numerical *matching score*. The ranking techniques also differ according to the index structure. For flat and multi-field indexes, the score can be calculated using text-based similarity measures such as cosine similarity or TF/IDF [Manning et al. 2008]. For structured indexes, ranking is based on ad hoc structural similarity metrics. More details about the latter case are provided in Section 3.3.

3. SEARCHING REPOSITORIES OF WEB APPLICATION MODELS

To make the architecture of Figure 3 concrete, it is necessary to instantiate it on a specific set of modeling languages and query paradigms. In this paper we focus on both text-based and content-based queries over repositories of models describing a specific class of applications – Web applications – and on a single modeling language, i.e., the Web Modeling Language (WebML) [Ceri et al. 2003].

3.1. The Web Modeling Language

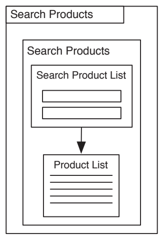
WebML is a visual Domain Specific Language that supports the high-level specification of Web applications, from the perspectives of the composition and navigation of the Web front-end, and of the data accessed by it [Ceri et al. 2000]; the language has a well-established industrial implementation and customer base [Acerbis et al. 2007] and has inspired the standard IFML (Interaction Flow Modeling Language) [Brambilla et al. 2013],[3] adopted in March 2013 by the OMG (Object Management Group).[4].

The choice of WebML as the target language for experimentation is motivated by several reasons:

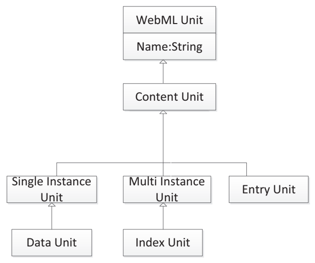
— the availability as the base for experimentation of a real-world industrial project repository created by professional developers.
— The generality and interest of the modeling domain (i.e., interactive application front-ends), in which WebML is just a representative of a family of DSLs that comprises several other languages with a similar purpose and structure, both in the academia (e.g., OOH [Gómez and Cachero 2003], UWE [Kraus et al. 2007], OOHDM [Rossi and Schwabe 2008], WADE [Gómez et al. 2007]) and in the industry (e.g., Rational Web Application Extension [Conallen 2000], Mendix, CodeCharge and Outsystems); the interaction front-end modeling domain is also the subject of an ongoing call for standardization proposals by the OMG [OMG 2011].
— The visual nature of the language, which makes it well suited to the "query by example" paradigm of content-based search.
— The nature of the WebML metamodel, which comprises different families of containers and modeling elements, with a rich set of relationships.

The main WebML constructs are *pages*, *units* and *links*, organized into *areas* and *site views*. A *site view* is a coherent hypertext, incorporating a well-defined set of requirements for a specific category of users. Site views can contain *Area*s, logical containers that group pages with a homogeneous purpose and can be nested recursively. *Page*s are

[3]Interaction Flow Modeling Language (IFML):`http://www.ifml.org`
[4]Object Management Group (OMG): `http://www.omg.org`



```
<Area  id="area1" name="Search Products" landmark="true"
linkVisibilityPolicy="inactive" defaultPage="page1"
landmarks="page1">
  <Page id="page1" name="Search Products" landmark="true">
     <ContentUnits>
          <IndexUnit id="inu1" name="Products List"
sortable="true" checkable="false" />
         <EntryUnit id="enu1" name="Search Product List"
linkOrder="ln1" >
            <Link id="ln1" name="Search" to="inu1" type=   "normal"
validate="true" automaticCoupling="true" />
         </EntryUnit>
    </ContentUnits>
  </Page>
</Area>
```



(a) (b) (c)

Fig. 4: Example of WebML model, (a) its XML representation (b) and an extract of the WebML metamodel (c).

contained in areas and site views, and represent the interface elements that are actually shown to the users. Site views, areas, and pages form the coarse structure of the front-end, which is then detailed by adding content and business logic components, called *Units*. There are two main types of units: *content units* and *operation units*. Content units are elements that express the content of a Web page, while operation units denote operations on data or arbitrary business actions; they can be activated as a result of navigation. Figure 4c contains an excerpt of the WebML metamodel taxonomy for content units: *Data Unit*s retrieve and present information about a single object; *Index Unit*s model the presentation of ordered sets of objects; *Entry Unit*s model Web input for data submission. Units are connected through *links* forming a hypertext structure. Links allow navigating hypertext front-ends, as well as passing parameters between units.

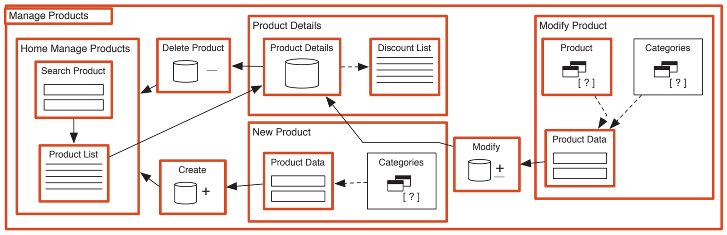
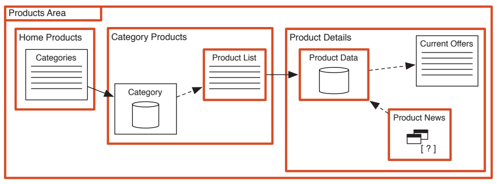
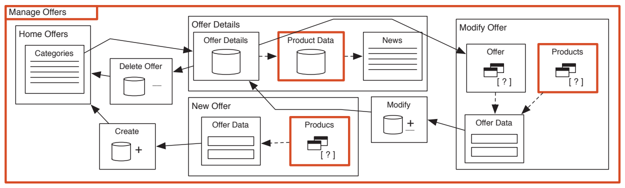
WebML models can be represented with a graphic notation or, equivalently, with an XML syntax. Figure 4a depicts an excerpt of a WebML model from an e-commerce application: the *Search Products* area contains a *Search Products* page where the user can enter data to search for a product; the search form is denoted by the *Search Product List* entry unit, while the returned product list is denoted by the *Products List* index unit; the link between the *Search Product List* unit and the *Products List* unit represents the navigation action of the user upon form submission, and it also specifies that the parameter required to execute the product search is passed to the index unit. Figure 4b contains the XML representation of the model fragment in Figure 4a, which comprises also the non displayed metadata of the model elements, e.g., their internal ID.

WebML models are designed by means of the WebRatio tool [Acerbis et al. 2007], or by any UML editor, using the WebML MOF metamodel; WebRatio has a basic in-memory project search facility, whereby the developer can execute keyword search within a single project. A repository of WebML models has been recently opened [WebRatio s.r.l. 2012], which can be browsed with an interface that organizes projects taxonomically and with tag clouds; basic keyword search is supported, with keywords matched in the textual description of projects.

In the rest of the paper, we explore both keyword-based search and content-based search over WebML repositories and illustrate the techniques adopted in the indexing, analysis, and querying processes.

Table I: Example of keyword-based query and top-3 results (with respective score values).

Query	Manage Search Product List

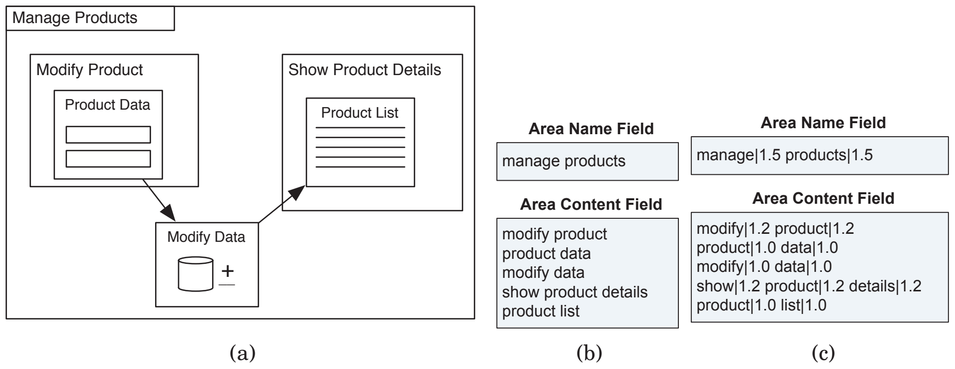
Res. ID	**Model**	**Score**
Result 1		3.9799
Result 2		2.2482
Result 3		2.2896

3.2. Keyword-Based Model Search

In keyword-based search, the input query consists of a bag of keywords. Each project from the repository is transformed into a set of terms, used to form the index. Keywords are matched against the index, and a TF/IDF based measure is used to compute the rank of the matching elements in the result set. In the following we introduce an illustrative example of keyword-based search, and then describe each of the system operations in details.

3.2.1. Illustrative example. Table I presents an example of textual query on a WebML model repository. Suppose the user is looking for a model that supports search and management of product lists in a Web-based system. He could formulate his information need as a keyword query like *Manage Search Product List*. Table I shows the top-3 results returned by our system in response to such query. Each of them consists of a model fragment (a WebML area), with decreasing matching score. The first result is a very relevant match, as the model fragment actually describes all the typical content management operations (creation, deletion, and modification) and contains a form for searching products. The subsequent matches are less precise: the second one misses some features, such as product search and updates; the third one only occasionally mentions products. The result set highlights the model elements that contain at least one of the search keywords and the match score is computed based on the number of matching and non-matching model elements.

3.2.2. Content Processing. The *Project Analysis* activity extracts only the project identifier, used to reference (at retrieval time) segments produced by the same project,



Area Name Field
manage products

Area Content Field
modify product
product data
modify data
show product details
product list

Area Name Field
manage|1.5 products|1.5

Area Content Field
modify|1.2 product|1.2
product|1.0 data|1.0
modify|1.0 data|1.0
show|1.2 product|1.2 details|1.2
product|1.0 list|1.0

Fig. 5: Example of WebML model(a) and different text indexing techniques: Metamodel-independent indexing (b) and metamodel-dependent indexing (c).

and the project name, used for result presentation. *Segmentation* is performed by using a metamodel-driven rule that considers *Area*s as segmentation units. Recall that WebML areas are logical containers of pages with similar purpose, thus guaranteeing a good degree of functional cohesion. For each resulting segment, the *Segment Analysis* task extracts the *name* attribute for each area, page, unit, and link; this attribute has a special role, because it represents an external label defined by the developer and used by the code generator to produce the rendition of the front-end (e.g., a menu item, a link anchor or button text, the heading of a page fragment, or a page name) and thus carries a high relationship to the semantics of the model element it denotes, and to the application domain where the element is applied. A reference between the extracted term and the originating model element type is also kept, to be used later in the indexing step. Finally, the *Linguistic Analysis* task tokenizes the text, removes highly frequent words that bear little information, and stems the remaining words, creating the terms to be stored in the index.

3.2.3. Indexing. To explore how the injection of metamodel information in the index impacts the retrieval performance of keyword-based search, two types of indexing strategies for WebML projects have been exploited, both based on a multi-field index:

— *Metamodel-independent* strategy: for each project, the index comprises two fields: one field (*Area Name*) is reserved for the area name, and the second field (*Area Content*) contains the index terms extracted by the *Content Processing* step. An additional auxiliary field, used only for result presentation, contains the identifier of the project to which the indexed area belongs. Figure 5a shows an example of WebML area, and Figure 5b the corresponding indexed representation in the metamodel-independent strategy.
— *Metamodel-dependent* strategy: the index field structure is the same as in the previous case, but a weight is added to each term based on the metamodel concept it comes from. Weights are configured manually offline. We experimented with several weight configurations, explained in Section 4. Figure 5c shows the model fragment of Figure 5a indexed according to the metamodel-dependent strategy, where the numerical values appended to textual terms are the applied weights.

3.2.4. Query Processing. The query is a bag of keywords, subjected to the same linguistic analysis pipeline performed on the projects.

3.2.5. Search. As for indexing, a metamodel-independent and a metamodel-dependent indexing and ranking approaches are employed.

The *metamodel-independent* approach uses the classic TF/IDF measure of IR [Manning et al. 2008], which combines the frequency of a query term in a document and its inverse frequency in the document corpus, so to penalize terms that occur frequently in a document and boost terms that occur rarely in the entire collection. The total TF/IDF score for a query and a document is computed as a sum of the scores of each query term. The total score is used to produce the ranking of the documents with respect to a given query, with higher score documents ranking higher in the result list.

In this work we propose a *metamodel-dependent* extension of TF/IDF that incorporates metamodel knowledge into a new parametric weighting term mtw, as reported in Equation 1:

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

where:

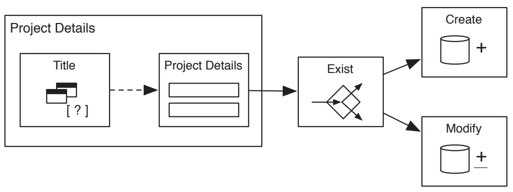
— q is a query, d is the indexed document (a WebML *Area*, in our experimental setting), and t is a term from the query q;
— $tf(t, d)$ is the *term frequency*, i.e., the number of times the query term t appears in the document d;
— $idf(t)$ is the *inverse document frequency* of t, i.e., a value calculated as $1 + \log \frac{|D|}{freq(t,d)+1}$, which measures the informative potential of the term in the entire document collection by calculating the ratio between the number of documents and the frequency of the term in the considered document; as a result, rare terms in the collection are considered more relevant than frequent ones;
— $mtw(m, t)$ is the *Model Term Weight* of a term t, i.e., a metamodel-specific boosting value that depends on the concept m containing the term t. For instance, in the example of Figure 5c, the weight for a term t associated with a *Page* element is set to be higher than the weight given to terms coming from other elements: $mtw(page, t)$ is set to 1.2, whereas mtw for all the others WebML concepts is set to 1.0.

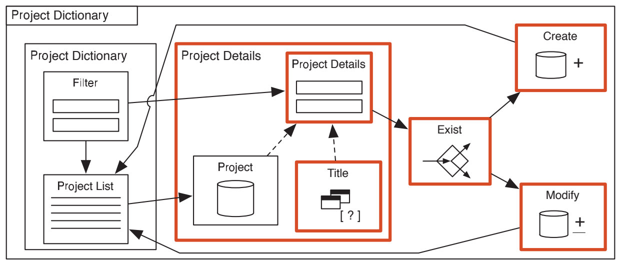
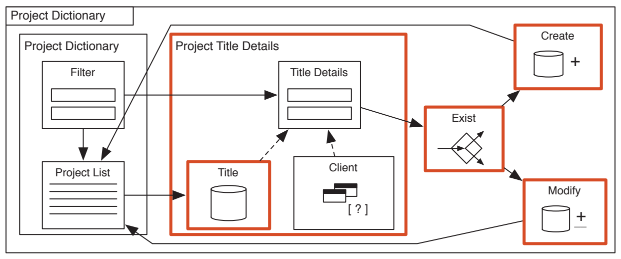
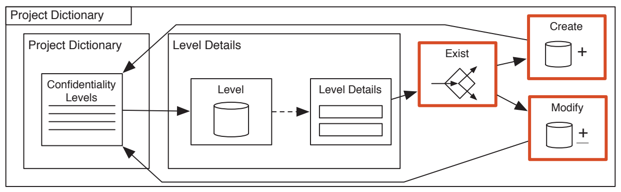
3.3. Content-Based Model Search

In content-based model search, queries are expressed as model fragments, and projects (or fragments thereof) are saved in semi-structured indexes to preserve relationships among model elements. Models, including WebML models, can be represented conveniently as *graphs* [Grigori et al. 2010; Qiao et al. 2011], which offer an abstract representation of the model elements and of their relationships. In the following we first introduce an illustrative example and then show how project and query models are indexed and queried.

3.3.1. Illustrative example. Table II shows a sample content-based query specified as a coarse WebML model; the query expresses a draft model consisting of a page and some operation units for searching a project by title and creating it (if not existing) or otherwise updating it. The top-3 results are shown; each result consists of a model fragment (a WebML area): the first one is a very precise match, where both structure and textual information fit with the query; the subsequent matches have lower scores because of the decreasing number of matching elements, either due to the imperfect structural overlap of the query and project models or to mismatches in the labels of the model elements. Note a difference with respect to the matches obtained for keyword-based search, exemplified in Table I: in the text-based case, some matches appear only because a label in the model element matches a keyword in the user's query; in the

Table II: Example of content-based query and top-3 results (with respective score values).

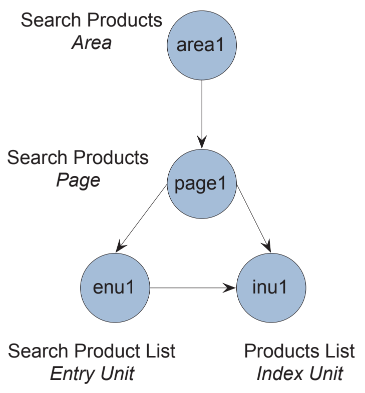
Query	 Project Details Title [?] Project Details Exist Create Modify

Res. ID	**Model**	**Score**
R1	 Project Dictionary Project Dictionary Filter Project List Project Details Project Details Project Title [?] Exist Create Modify	0.906
R2	 Project Dictionary Project Dictionary Filter Project List Project Title Details Title Details Title Client [?] Exist Create Modify	0.847
R3	 Project Dictionary Project Dictionary Confidentiality Levels Level Details Level Level Details Exist Create Modify	0.828

content-based case, matches must adhere both to the textual content and to the structure of the query: for example, in the top result listed in Table II the Project Dictionary area is not part of the match, even if it contains the word *project* that is part of the query, because it does not correspond to any element appearing in the user's query.

3.3.2. Content and Query Processing. The WebML models from the repository are first split into areas, as in the case of keyword-based search described in Section 3.2. Subsequently, they are translated into directed labeled graphs, according to a mapping proposed in this work. The resulting graphs are used to build the index. Since content-based queries are also WebML models, they can be transformed in the same way as projects.

A WebML graph is a triple $g = (N, E, L)$, where N is a set of nodes, E is a set of edges, and L is a set of labels representing metadata about the nodes. Each WebML



(a)

```
<graphml>
  <graph edgedefault="directed">
    <node id="area1">
      <name>Search Products</name>
      <type>Area</type>
      <occurence>1</occurence>
    </node>
    <node id="page1">
      <name>Search Products</name>
      <type>Page</type>
      <occurence>3</occurence>
    </node>
            . . .
    <edge id="edge inu1">
      <source>page1</source>
      <target>inu1</target>
    </edge>
    <edge id="edge link ln1">
      <source>enu1</source>
      <target>inu1</target>
    </edge>
  </graph>
</graphml>
```

(b)

Fig. 6: Pictorial (a) and XML (b) representations of the graph corresponding to the WebML example of Figure 4.

element maps to a graph node, identified with the same XML ID and annotated with its *name* and metamodel *type*. Therefore, each graph node is associated with a pair of labels l_N, l_T, that represent the *name* and the *type* label of the corresponding WebML construct. Two types of relationships between the WebML elements are mapped into graph edges: (i) *containment* relationships, which connect container elements (e.g., site views, areas and pages) with the elements they comprise; for example, a containment relationship exists among an area and all the pages contained in it. And (ii) navigational *Link*s, which model the navigation between pages and the functional dependency (i.e., parameters) between units. For each link, an edge that connects the nodes mapping its source and destination unit is added to the graph.

Figure 6a shows the graph representation of the WebML fragment in Figure 4a: the *Search Product* area (id = area1), the *Search Products* page (id = page1), the *Search Product List* entry unit (id = enu1) and the *Products list* index unit (id = inu1) are mapped to nodes labeled with the *name* attribute and the *type*, and identified with the same ID of the corresponding WebML elements. The edges connecting *area1* and *page1*, *page1* and *enu1*, and *page1* and *inu1* represent containment relationships in the original model; the edge that connects *enu1* with *inu1* represents the link connecting the two units. Figure 6b shows the equivalent XML representation of the graph in Figure 6a.

Differently from the case of keyword-based search, no linguistic analysis is performed on the text extracted from the WebML model. This is justified by the mechanism used for query-to-project matching, described in details in Section 3.3.3, which exploits a string similarity measure to compare graph nodes.

3.3.3. Search. As both projects and queries are represented as graphs, *search* is performed by verifying whether the query graph is contained in a project graph. In a query-by-example scenario, the query graph will be normally smaller than the project graph. Therefore the goal is to find whether the query graph *is a part* of the project graph; this graph matching problem can be tackled by computing *subgraph isomor-*

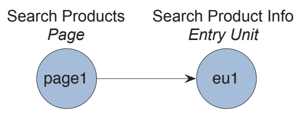


Fig. 7: An example of a query graph.

phism [Bunke 2000], i.e., an injective mapping that identifies a subgraph in the project graph that has corresponding nodes and edges in the query graph, preserving the graph structure and label equality constraints.

Subgraph isomorphism is known to be NP-complete [Cook 1971]; however, query processing does not require finding an *exact* match between the query graph and a subgraph in the project, a case that would be very rare due to differences in model concept naming and linking. A sufficient objective is to find a subgraph in the project graph that is *equivalent*, or *similar*, to the query graph. Therefore, it is necessary to consider a heuristic algorithm that matches graphs by means of an approximate measure of *similarity* [Bunke 2000]. A classical solution to this problem exploits the A-star algorithm and the graph edit distance similarity measure [Sanfeliu and King-Sun 1983; Gregory and Kittler 2002; Dijkman et al. 2009], explained in the rest of this Section.

Graph edit distance. A metric that computes the similarity of two graphs is the *graph edit distance*, defined as the minimum number of edit operations that transform one graph into the other [Dijkman et al. 2009]. Indeed, the less transformations are applied, the more similar two graphs are. Given a comparison graph G_1 (i.e., the query) and a compared graph G_2 (i.e., the project), the graph edit distance considers the following types of edit operations, namely:

— *Node substitution*: it substitutes (maps) nodes from G_2 that are *similar* to nodes from G_1, under an externally provided notion of node similarity.
— *Node insertion*: it inserts non-similar nodes from G_1 into G_2.
— *Node deletion*: it deletes from G_2 all non-similar nodes. A deletion from one of the two graphs can be treated equivalently as an insertion in the other graph.
— *Edge insertion/deletion*: it inserts into G_2 all edges that do not connect two similar (substituted) nodes of G_1; or, equivalently, it deletes from G_1 all edges that do not connect two similar nodes of G_2.

To exemplify the operations of the graph edit distance, we compare the query graph in Figure 7 with the project graph in Figure 6a. In this example, we consider that a node from the query graph is similar to a node from the project graph iff they have the same metamodel type and exactly the same name. A more flexible node similarity function will be introduced in a following paragraph. The node *"Search Products"* of type *"Page"* in the query graph is similar to the node having the same name and type in the project graph. The other query node *"Search Product Info"* of type *"Entry Unit"* has no similar nodes in the project; therefore, it is inserted into the project graph. All the other nodes of the project graph are non-similar and thus deleted. Since only one node from the query graph is similar to a corresponding node in the project graph, the edge outgoing from it in the query graph is inserted in the project graph, and all the four edges from the project graph are deleted. In summary, the query graph is obtained from the project graph as a result of 1 node substitution, 1 node insertion, 3 node deletions, 1 edge insertion and 4 edge deletions.

The *graph edit similarity*, defined in Formula 2, quantifies, in the $[0,1]$ range, graph similarity by normalizing the operations considered in the graph edit distance:

$$G_{Sim}(G_1,G_2) = 1 - \frac{w_{nI} \cdot f_{nI}(G_1,G_2) + w_{eI} \cdot f_{eI}(G_1,G_2) + w_{nS} \cdot f_{nS}(G_1,G_2)}{w_{nI} + w_{eI} + w_{nS}}. \quad (2)$$

where f_{nI}, f_{eI} are the fractions of inserted nodes and of inserted edges, calculated as the ratio of inserted nodes N_i (edges E_i) *in both graphs* with respect to the total number of nodes (edges) *in both graphs*.

$$f_{nI}(G_1,G_2) = \frac{\mid N_i \mid}{\mid N_1 + N_2 \mid} \qquad f_{eI}(G_1,G_2) = \frac{\mid E_i \mid}{\mid E_1 + E_2 \mid}. \quad (3)$$

The values of f_{nI},and f_{eI} increase as the number of non similar nodes or edges grows. The average distance of substituted nodes f_{nS} is defined as:

$$f_{nS}(G_1,G_2) = \frac{2 \cdot \sum_{(n_1,n_2)}(1 - sim(n_1,n_2))}{\mid N_s \mid}. \quad (4)$$

that is the sum of one minus the node similarities of all substituted nodes, normalized with respect to the total number of substituted nodes in both graphs. The average distance increases if the node pairs are less similar, and is 0 iff only identical nodes are substituted.

Formula 2 assigns to each edit operation a cost (weight), which gives the corresponding operation more or less influence on the result of the graph edit similarity computation. The constant values w_{nI}, w_{nS}, and w_{eI} range in the [0,1] interval and respectively represent the weights for node insertion, node substitution, and edge insertion. A higher value for an operation increases its contribution in the calculation of the distance between two graphs, i.e., the penalty incurred when one instance of that operation is applied to align the query and the target graphs. Weighting more insertion components of the graph edit distance emphasizes the dissimilarity due to graph topology; increasing the weight of the node substitution augments the penalty for considering equivalent nodes that do not match exactly.

As an example, the comparison of the query graph in Figure 7 with the project graph in Figure 6a results in $f_{nI} = 4/6 = 0.67$, because the total number of nodes *in both graphs* is 6 and the total number of node operations (deletions and insertions) is 4, while $f_{eI} = 5/5 = 1$, because the total number of edges *in both graphs* is 5 and the total number of edge operations (deletions and insertions) is also 5. The average distance of substituted nodes is $f_{nS} = \frac{2 \cdot (1-1)}{2} = 0$, because the pair of substituted nodes has similarity 1 (they are identical). If the weights for this example are chosen to be, for example, $w_{nI} = 0.3$, $w_{nS} = 0.8$, and $w_{eI} = 0.5$, then the final graph edit similarity between the two graphs is $G_{Sim} = 1 - \frac{0.3 \cdot 0.67 + 0.5 \cdot 1 + 0.8 \cdot 0}{0.3 + 0.5 + 0.8} = 0.562$.

Node similarity. A central aspect in the evaluation of the similarity of two graphs is the calculation of the similitude of two nodes in order to determine whether they match, i.e., they can be considered similar, and thus be substituted (since they are interchangeable), instead of inserted. The node similarity can be computed by evaluating a distance function that considers the properties of the evaluated nodes. In our approach, differently from previous work, we adopt a distance function that considers both the metamodel type of the model element associated with the graph node and its textual label, as shown in Equation 5:

$$Dist(n_1,n_2) = \lambda \cdot stringDist(name_{n_1}, name_{n_2}) + (1 - \lambda) \cdot typeDist(n_1,n_2) \quad (5)$$

$Dist(n_1, n_2)$ is calculated as the weighted linear combination of two distances, where:

— $stringDist$ is a string distance metric, normalized in the [0,1] range, quantifying the similarity between the nodes' labels; our experiments, detailed in Section 4, compared the performance of two state-of-the-art string distance metrics, respectively the Levenshtein distance [Levenhstein 1966], and the n-gram distance [Hylton 1996].
— $typeDist$ is the distance between two concepts in the metamodel, considered as a graph, normalized with respect to the maximum node distance in the metamodel graph.
— The parameter $\lambda \in [0, 1]$ determines the relative importance of the name and type distance. $\lambda = 0$ takes into account only type contribution, while $\lambda = 1$ takes into account only the name similarity.

To exemplify the computation of the node distance, let us consider the *Search product List* and the *Products List* units of Figure 6a. According to the WebML metamodel excerpt of Figure 4c, the type distance between an *Entry Unit* and an *Index Unit* is 0.75 (because the distance between the two classes is 3 and the maximum node distance in the graph is 4), while the string distance $stringDist(\text{“}SearchProductList\text{”}, \text{“}ProductsList\text{”})$, calculated using the *Levenshtein* distance, is 0.58. Therefore, with $\lambda = 0.5$ the distance between the two nodes is 0.66.

Variations of the λ parameter value allow for different similarity evaluation scenarios. A high value of λ describes the situation in which a user considers two model elements similar only by looking at their names. For instance, the data unit “Product” would be considered “similar” to an index unit “Products”, even if the former displays one object, whereas, the latter presents a list of objects. Conversely, a low value of λ would emphasize the semantic similarity, at a metamodel level, of model elements. In this case, an index unit “Product list” would be considered equivalent to an index unit named, e.g., “Offer List”.

A-star algorithm. The A-star algorithm is a method to compute subgraph isomorphism through graph similarity [Shapiro and Haralick 1981; Sanfeliu and King-Sun 1983]. Different variations exist; the version used in this work follows the template described in [Messmer 1996], and then modified and applied for searching repositories of business process models in [Dijkman et al. 2009]. Our approach is inspired to the latter work, but significantly extends it with metamodel-aware weights, distances, and parameters, which were not considered in the original algorithms.

The algorithm finds the optimal mapping between two graphs, using a best-first search of the solution space (the space of all mappings between the query graph and the project graph); it proceeds iteratively by searching the least-cost extension of a given initial partial graph mapping until a complete graph mapping is found; cost is computed with the graph edit similarity function, which drives the extension of the current partial mapping into the next expanded mapping that yields the maximal graph edit similarity between the query graph and the project graph.

The pseudo-code of ALGORITHM 1 illustrates the A-star procedure. It uses:

— the sets of nodes of the query graph (N_1) and of the project graph (N_2).
— A variable *open*, which is initialized with the set of all *allowed* mappings for an initial arbitrarily selected node n_1 of the query graph; the set of allowed candidate mappings is used to expand the current partial solution; a mapping is allowed if it contains node pairs with similarity above a given *threshold*, or node pairs where the query graph node is mapped to the conventional node deletion symbol ϵ.

ALGORITHM 1: A-star algorithm

```
Require: $open \leftarrow (n_1, n_2) \mid n_2 \in N_2 \cup \{\epsilon\}, sim(n_1, n_2) > threshold \vee n_2 = \epsilon$ , for some $n_1 \in N_1$
  while $open \neq \emptyset$ do
    select $map \in open$, such that $s(map)$ is max
    $open \leftarrow open - map$
    if $dom(map) = N_1$ then
      return s(map)
    else
      select $n_1 \in N_1$, such that $n_1 \notin dom(map)$
      for all $n_2 \in N_2 \cup \{\epsilon\}$, such that ($n_2 \notin cod(map)$ and $sim(n_1, n_2) > threshold$) xor ($n_2 = \epsilon$)
      do
        $map' \leftarrow map \cup \{(n_1, n_2)\}$
        $open \leftarrow open \cup map'$
      end for
    end if
  end while
```

— A variable *map*, which contains the current partial mapping solution having the maximal graph edit similarity *s(map)*; *s(map)* is evaluated as in Equation 2 by considering all node pairs contained in *map* as substituted, the remaining query nodes as inserted, the unmapped project nodes as deleted, and counting inserted/deleted edges accordingly.

A-star starts from an initially empty current mapping and a node n_q^1 in the query graph, and creates all the possible partial mappings $(n_q^1, n_p^{1_i})$ from this node to every node in the project graph. Additionally, an extra mapping with a dummy node ϵ is created, (n_q^1, ϵ), denoting the case where n_q^1 is deleted. The partial mapping $(n_q^1, n_p^{1_{i^*}})$ or (n_q^1, ϵ) with the maximal graph edit similarity is selected, and added to the current candidate solution mapping. Then, the algorithm proceeds with the next node from the query graph, and creates partial mappings with every other non-mapped node from the project graph. At each round, the current candidate solution mapping is expanded with the mapping of the nodes that produces the maximal graph edit similarity. The algorithm finishes when the current candidate solution mapping contains all the nodes from the query graph. The returned value is the maximal graph edit similarity for the query and the project graphs.

The best case complexity of the algorithm occurs when the nodes of the project and of the query graph have the same labels, and the query graph is an exact copy of the project graph. Therefore, for a query graph with m nodes and a project graph with n nodes, the best case complexity is $O(n^2m)$. The worst case occurs when the query graph is very different from the project graph, both in terms of labels and structure; in such a case, many edit operations are necessary to transform one graph into the other, resulting in exponential complexity. For a query graph with m nodes and a project graph with n nodes, the worst case complexity is $O(nm^n)$. To reduce the search space, and limit space and memory requirements, a pruning rule is used: only nodes with similarity greater than the *threshold* parameter are allowed as candidate mapping pairs.

Let us consider the comparison of the query graph in Figure 7 and the project graph in Figure 4, assuming a *threshold* for node similarity of 0.7, and the parameter $\lambda = 0.5$, which gives equal importance to name and type similarity. In the first step, if we start with the query node *page1*, this node can form two possible partial mappings:

```
< ("Search Products:Page","Search Products:Page") >
< ("Search Products:Page","ϵ") >
```

The first pair is the mapping with the maximal graph edit similarity and thus is selected and expanded into new partial mappings that include the second query node *enu1*. The following ones are the possible mappings of cardinality 2:

```
< ("Search Products:Page","Search Products:Page"),
                 ("Search Product Info:Entry Unit","Search Product List:Entry Unit") >
< ("Search Products:Page","Search Products:Page"),
                 ("Search Product Info:Entry Unit","ϵ") >
```

At the second round, the algorithm selects the former complete mapping, which has the maximal graph edit similarity, and terminates, because all query nodes are mapped. The computed mapping specifies that in order to transform the query graph into the project graph, both query nodes are substituted, and the project graph nodes (“`Manage Products: Area`”,“`Products List Index Unit`”) are deleted (or equivalently inserted into the query graph). The edge in the query graph is substituted with the edge between the corresponding nodes in the project graph and the remaining edges from the project graph are deleted (or equivalently inserted into the query graph).

A-star algorithm with local search. The original A-star algorithm can map query nodes to graph nodes arbitrarily positioned throughout the project. The cost of including in the match nodes that are far apart in the project graph is proportional to the number of edges that must be inserted to connect such nodes; the relative contribution of edge insertion in large graphs with many edges may be limited, and so A-star tends to accept matches where query nodes are associated with projects nodes far apart in the project graph. The locality of matches may impact the retrieval performance, which raises the issue whether highly connected matches are preferable to more distributed ones. To investigate the effects of locality constraints, we evaluate a variant of A-star, which attempts at boosting more cohesive matches by imposing an additional constraint for adding a node to the current partial mapping: only those nodes that are at the shortest distance with respect to already mapped nodes are used to extend the current mapping.

The pseudocode of the local search variant is listed in Algorithm 2. At the beginning, the set of partial mappings for the initial query node (n_1) contains all the nodes (n_2) from the project graph with similarity to n_1 greater than the threshold, plus the mapping with the node deletion symbol (ϵ). For every node n_2 in the project graph, denoted as $graph_2$, the shortest path to every other node in the graph is computed using the Dijkstra algorithm [Dijkstra 1959] and saved in the variable *path_set*. In the next step, the algorithm proceeds as the A-star algorithm, by selecting the partial mapping with the maximum graph-edit similarity. Then, the partial mapping is extended: the next query node is mapped to viable candidate project nodes; these are the project nodes with similarity above threshold and *positioned at the shortest distance*, defined as the minimum distance between an unmapped project node with similarity above threshold and an already mapped project node. When multiple paths exist with minimal length, all of them are considered and their source nodes treated as candidates. After identifying all nodes above threshold and within minimum distance, the algorithm expands the current mapping so to maximize the graph edit similarity and proceeds like the normal A-star algorithm.

Notice that, since the local search constrains the candidate matches, it happens that: (1) the number of matches computed by local A-star is smaller or equal than that of the original A-star; and (2) A-star and local A-star differ only when there are multiple matched node pairs between a query graph and the project graph.

ALGORITHM 2: A-star algorithm with local search

```
Require: $open \leftarrow (n_1, n_2) \mid n_2 \in N_2 \cup \{\epsilon\}, sim(n_1, n_2) > threshold \vee n_2 = \epsilon$ , for some $n_1 \in N_1$
  for all $n_2 \in N_2$ do
    $path\_set \leftarrow Dijkstra\_Shortest\_Path(graph_2, n_2)$
  end for
  while $open \neq \emptyset$ do
    select $map \in open$, such that $s(map)$ is max
    $open \leftarrow open - map$
    if $dom(map) = N_1$ then
      return s(map)
    else
      select $n_1 \in N_1$, such that $n_1 \notin dom(map)$
      $min\_path\_set \leftarrow \emptyset$
      $min\_length \leftarrow \infty$
      for all $path \in path\_set$ do
        if $source(path) \notin cod(map)$ then
          if $sim(n_1, source(path)) > threshold$ then
            if $target(path) \in cod(map)$ then
              if $length(path) == min\_length$ then
                $min\_path\_set \leftarrow min\_path\_set \cup path$
              else
                if $length(path) < min\_length$ then
                  $min\_path\_set \leftarrow \emptyset$
                  $min\_path\_set \leftarrow min\_path\_set \cup path$
                  $min\_length = length(path)$
                end if
              end if
            end if
          end if
        end if
      end for
      $source\_nodes \leftarrow extract\_source\_nodes(min\_path\_set)$
      for all $n' \in source\_nodes \cup \epsilon$ do
        $map \leftarrow map \cup \{(n_1; n')\}$
        $open \leftarrow open \cup map$
      end for
    end if
  end while
```

4. EXPERIMENTAL EVALUATION

In this Section we present the experiments on different model search scenarios constructed according to the techniques described in Section 3. We structured our experiments in two parts: (1) a technical evaluation of the keyword- and content-based systems, to analyze performance under different system configurations using quality indicators based on a gold standard created by a panel of experts; (2) a user study, where WebML practitioners were asked to assess how much the proposed methods could help them reusing existing modeling artifacts. The two experiments complement each other and provide a comprehensive assessment of the systems behavior.

4.1. Experimental setting and dataset

4.1.1. Test bed. The experiments were performed on a project repository provided by WebRatio[5]; the company that develops the homonymous MDD tool for WebML and

[5]http://www.webratio.com

IFML modeling and automatic generation of Web applications. The repository contains 12 real-world WebML projects from different application domains (e.g., trouble ticketing, human resource management, Web portals, etc.). The projects are encoded as XML files conforming to the WebML DTD, and their domains and size are presented in Table III. We segmented projects at the area level, which resulted in 341 areas.

The choice of exploiting a proprietary project repository is motivated from the following considerations: first, having access to a collection of projects built by professional modelers in real-world projects (which we publish as model graphs for further experiments by the community); second, existing publicly accessible model repositories proved unsuitable for realistic model search experiments because they either contain very elementary models or do not provide full access to their content (see the Related Work Section for further details).

Table III: Testbed repository. Project ID, domain, and number of contained *area*s.

Project ID	**Domain**	**Number of *areas***
1	Administration	23
2	Human resource management	53
3	Call center web portal	56
4	Calendar management	3
5	Bank account management	58
6	E-commerce	15
7	Rent-a-car	2
8	Adminstration	30
9	Company intranet	58
10	Web portal	5
11	Candidate evaluation	24
12	Trouble ticketing	12

An evaluation set of 10 queries was built as follows: first, to ensure a good coverage of all the system features that we wanted to test in the experiments and the coherence between the evaluation set and the repository content, we defined several exemplary models, satisfying the following properties: (i) they implemented theoretical [Ceri et al. 2003] and real-world WebML modeling patterns; (ii) they used a broad mix of the WebML metamodel concepts; and (iii) they were based on a vocabulary (of labels) consistently used in the experimental project repository. Out of this initial set, a group of three experienced WebML developers were consulted to select the 10 exemplary models which, in their opinion, better represented the typical user need of MDD developers in their everyday activities. Finally, the exemplary models were transformed into keyword-queries, by selecting as keywords all the significative labels; and into content-based queries, by mapping each WebML model into a graph as explained in Section 3.3.

4.1.2. Gold Standard Creation. The gold standard for the comparison of the retrieval methods was constructed by manually assessing the extent to which each area in the repository contained a model similar to those in the evaluation set; the three experts assigned a relevance score expressed in a tertiary scale where, (i) 0 relevance means no similarity, (ii) 1 means that some textual *xor* structural similarity exists, and (iii) 3 corresponds to a judgment of strong similarity (textual *and* structural). The final relevance was calculated as the average of the three judgments, rounded to the nearest integer. To reduce fatigue and learning bias, the evaluation task was spread over multiple days.

Figure 8 exemplifies the kind of judgements about query-area matches expressed by evaluators. The query (Figure 8(a)) looks for an area that implements a creation/modification pattern for new/existing products; the area shown in 8(b) contains a pattern that performs the same action, but using slightly different labels. Given that the query's structural pattern is present in the project, and there is also a partial textual similarity (the terms *product* and *title* are present both in the query and in the matching area), a relevance value of 3 qualifies this match. Figures 8(c) and (d) show examples of areas where the relevance with respect to the query is 1 and 0, respectively. As it can be noticed, the area with similarity 1 contains a pattern that verifies the existence of a *store* in order to be created or modified. This area has only structural pattern similar to the query, and no textual similarity. Finally, the project with similarity 0 has nothing in common with the query.

The gold standard dataset ranks for each query the areas, according to the average relevance score of the match, breaking ties with a deterministic rule.

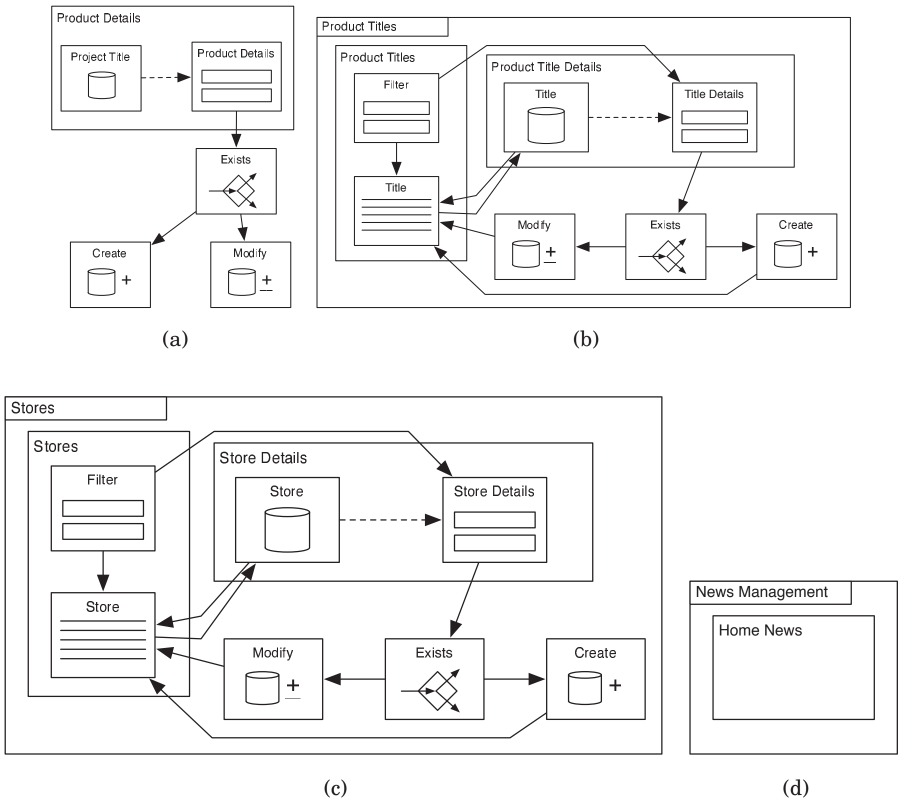
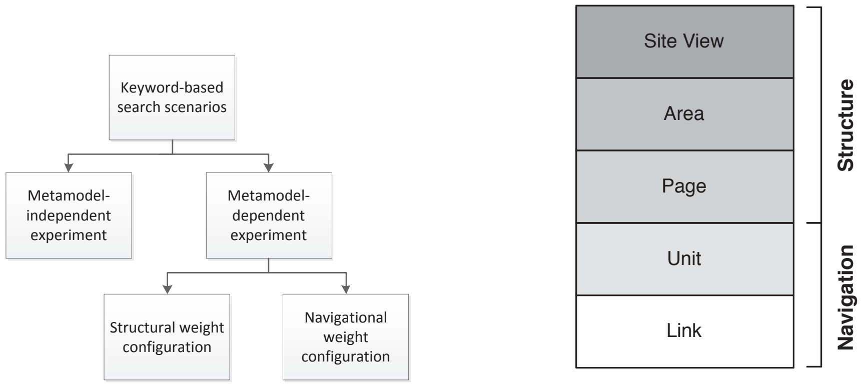


Fig. 8: Example of a WebML query (a), area with similarity 3 (b), area with similarity 1 (c) and area with similarity 0 (d).

The full set of queries, areas, and evaluation scores can be downloaded from the following URL: `http://webml.org/webml/modelsearch/modelsearch.html`.

4.1.3. Experimental scenario. As a baseline for comparing the retrieval accuracy of both the text-based and content-based approaches, we use randomly generated result sets. A random result set is a sequence of areas randomly extracted from the projects in the repository, ordered randomly. The value of each performance indicator in the baseline


Keyword-based search scenarios
Metamodel-independent experiment
Metamodel-dependent experiment
Structural weight configuration
Navigational weight configuration
Site View
Area
Page
Unit
Link
Structure
Navigation

(a) (b)

Fig. 9: (a) Configurations for the keyword-based search scenario; (b) Classification of WebML metamodel concepts.

case is calculated by averaging, for each query, the results of 10 random area extraction and ranking steps.

We could not compare the performance of our implementation directly with existing works in model search for various reasons: some works are not metamodel-aware at all, and therefore cannot be ported to other modeling languages; some others assess their quality upon non public data sets; and finally, others do not provide publicly available systems to compare with.

Keyword-based search. Figure 9a summarizes the evaluation scenarios for keyword based search. Experiments were conducted under a (i) *metamodel-independent*, and a (ii) *metamodel-dependent* configuration. The index of the metamodel-independent experiment contains equally weighted terms, regardless of the metamodel type of the element of origin. Conversely, the indexes of the metamodel-dependent experiments weight terms according to the category of the metamodel type of the element where the term appears.

We categorize the five WebML model primitives relevant for matching the query to the projects into *structural* and *navigational*. Figure 9b shows such classification: site views, areas, and pages represent mainly modularization constructs used to group more detailed elements; units and links embody the composition and navigation aspects of the user interface and denote the functions triggered by the user's navigation. The top-down order of elements in Figure 9b follows the element containment relations: siteviews contain areas, which in turn contain pages, which in turn contain units, connected through links.

Assuming 1.0 as the minimum term weight, we assigned weights in the [1,2] range[6]. We tested two different weight configurations as illustrated in Figure 9a. The first one, named *Structural*, gives more importance to structural metamodel concepts, assigning higher weights to terms associated with site views, areas and pages. The second configuration (named *Navigational*) reverts the weight distribution and assigns more weight to links and units. Table IV shows the two weight assignments used in our experiments.

[6]Higher weight values would introduce too much bias in the evaluation of Equation 1, causing the relative weight (with respect to the overall score) of the term to dominate other factors.

Table IV: Keyword-based search: weight configurations for the metamodel-dependent experiment.

Metamodel concept	***Structural*** Configuration	***Navigational*** Configuration
site view	2.0	1.0
area	1.8	1.2
page	1.5	1.5
unit	1.2	1.8
link	1.0	2.0

Content-based search. The content-based scenario tested four configuration dimensions: (i) the w_{nI} (node insertion), w_{nS} (node substitution), and w_{eI} (edge insertion) weights of the graph edit distance operations, (ii) the parameter λ, which determines the importance of the string distance and type distance in the node similarity function; (iii) the string similarity function used to calculate the node similarity; and (iv) the adoption of locality constraints in the subgraph isomorphism algorithm. In all the reported experiments, after an initial set-up phase with the A-star algorithm, we fixed to 0.6 the node similarity threshold for the pruning rule that discards non-allowed matches, as the value proved best in all the considered settings.

The first set of experiments aimed at understanding the impact of the weights assigned to the graph edit distance operations, and we considered three configurations (summarized in Table V):

— *Maximal Substitution* boosts the contribution of the node substitution.
— *Maximal Substitution and Insertion* emphasizes both insertion of nodes/edges and their substitution.
— *Maximal Insertion* stresses only the insertion of nodes/edges.

Table V: Content-based search: different weight configurations.

weights	**Maximal substitution**	**Maximal substitution and insertion**	**Maximal insertion**
w_{eI}	0.1	1.0	1.0
w_{nI}	0.1	1.0	1.0
w_{nS}	1.0	1.0	0.1

The second set of experiments examined the influence of the λ parameter in the node similarity function. We varied the values of λ from $\lambda = 0$ to $\lambda = 1$, with step 0.25. The λ values and corresponding experiment names are reported in Table VI: recall that higher values of λ give more importance to name similarity w.r.t. metamodel type similarity.

Table VI: Content-based search: different λ values.

Only type contribution	$\lambda = 0$
High type contribution	$\lambda = 0.25$
Intermediate type contribution	$\lambda = 0.5$
Low type contribution	$\lambda = 0.75$
No type contribution	$\lambda = 1.0$

The third set of experiments analyzed the impact of the adopted string distance metrics. String distance metrics are similarity functions that do not consider prior knowledge and thus exhibit performance that is strongly related to the specific application domain [Bilenko et al. 2003]. We compared two frequently used functions: the *Levenshtein* distance [Levenhstein 1966], and the *n-gram* distance [Hylton 1996].

The *Levenshtein* distance is a string-edit distance that, given two strings, finds the minimal number of string edit operations that transform one string into the other, normalized with the length of the longer string. For two identical strings, the *Levenshtein* distance is 0, and the corresponding similarity value is 1.

The *N-gram* distance is a string token distance which finds the common number of n-grams (substrings of the original string with fixed length n) for two strings, normalized with the total number of n-grams. The *N-gram* distance has values in the [0,1] interval, where 0 means no similarity, and 1 is an exact match.

Finally, the fourth set of experiments assessed the effect of applying locality constraints in the selection of candidate mappings in A-star. In particular, we evaluated the original version (Algorithm 1) and the local version (Algorithm 2) of A-star.

4.2. Evaluation Metrics

Performance is evaluated using three standard information retrieval measures: (i) 11-point interpolated average precision; (ii) Mean Average Precision (MAP); and (iii) Discounted Cumulative Gain (DCG).

Precision and recall are the two most used IR evaluation measures. Precision considers the fraction of retrieved documents that are relevant, regardless of the ranking, while recall measures the fraction of relevant documents that are retrieved.

The *11-point interpolated average precision* combines precision and recall by measuring the best precision that can be obtained at 11 standard levels of recall (0.0, 0.1,...1.0) [Manning et al. 2008]. At each each recall level r_i, the interpolated precision is obtained as an average over the sample queries and represents the highest precision that can be obtained for recall values $r_j \geq r_i$. The 11-point precision value decreases for increasing recall, as for a growing number of retrieved results, the likelihood of irrelevant matches typically increase.

Mean Average Precision (MAP) [Manning et al. 2008] is a single figure quantification of the average precision across recall levels and queries: for each query, the average precision is computed as the average of the precision value obtained in the set of top-k documents that are retrieved to get to the j-th relevant document. More precisely, if the set of relevant documents for a query $q_j \in Q$ is $\{d_1, \ldots d_{m_j}\}$, where m_j is the number of relevant documents, and R_{jk} is the ordered set of the first k ranked results, then:

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk}) \tag{6}$$

When the first *k* positions of the result set contain no relevant documents, the precision value in Equation 6 is 0. In our case, MAP is calculated up to the top 10 matching projects.

Finally, the *Discounted Cumulative Gain* (DCG) [Järvelin and Kekäläinen 2002] is a graded relevance measure that evaluates the ability of an IR system to retrieve highly relevant documents at high positions in the result set. DCG considers the fact that the lower a document is ranked in a result set, the less likely it is for such a document to be examined by a user. DCG is computed as:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(1+i)} \tag{7}$$

where rel_i is the relevance of the document at the i-th rank position obtained from the gold standard dataset evaluation.

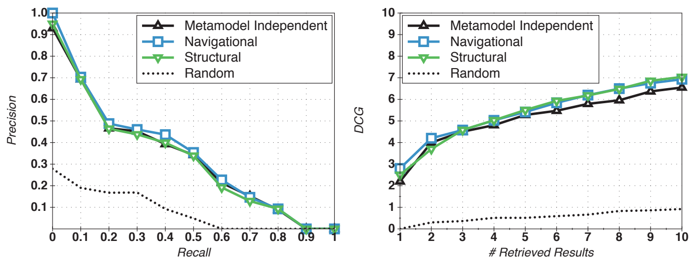
4.3. Quantitative Evaluation

4.3.1. Keyword-based search. Table VII shows the values of MAP for different indexing structures. All index structures achieve good performance (with peak MAP value of 81%), significantly better than the random baseline. Adding metamodel-dependent weights to the index slightly increases the performance for the tested queries (4% MAP increase in the best case).

Figure 10a and Figure 10b show the results of DCG and 11-point precision. Also these measures support the conclusion that the different configurations of the index for the textual search exhibit a comparable *average* behavior. Boosting the weight of more specific elements (units and links) over high-level ones (site views, areas) provides slightly improved performance: the average performance of the navigational configuration increases by 2% for DCG and 5% for 11-point precision with respect to the structural configuration, and 7% for DCG and 3% for the 11-point precision with respect to the metamodel-independent configuration.

Table VII: Keyword-based search: values of MAP

Experiment	*MAP*
Random	0.19
Metamodel-independent	0.77
Metamodel-dependent *Structural* Configuration	0.78
Metamodel-dependent *Navigational* Configuration	**0.81**

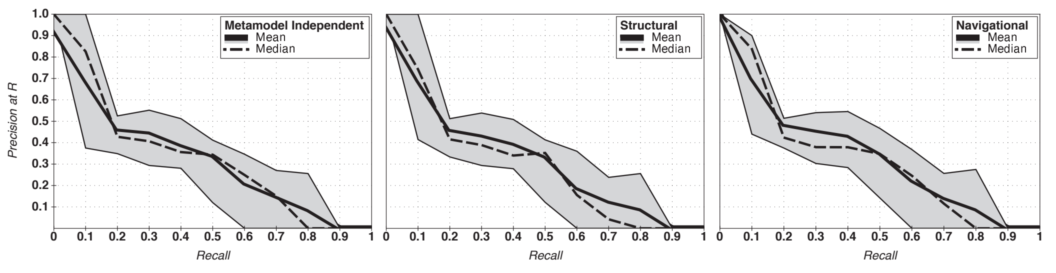


(a) (b)

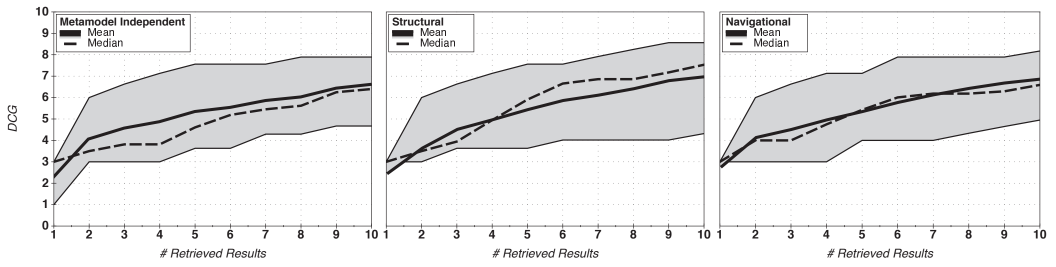
Fig. 10: Keyword-based search: 11-point precision (a) and DCG (b).

Figure 11a and 11b explode Figure 10 to examine the average and median values over the sample queries, and the upper and lower quartiles (gray area). The growth of the DCG values slows down at higher rank positions. Since DCG depends not only on the precision and recall but also on the rank order of the retrieved documents, the slow down at higher ranks shows that even if relevant documents are retrieved they are not ranked optimally, w.r.t the gold standard, when one looks at larger result sets.

The comparison of the metamodel-dependent and the metamodel-independent index structures in Figure 11b shows that the latter exhibit an average DCG value consistently higher than the median, thus indicating the presence of several outliers and, therefore, a less uniform ranking behavior across sample queries. The distribution of differences in the 11-point average precision graph, i.e., the gray area between the lower and upper quartile in Figure 11a, at lower levels of recall shows that the structural setting has more performance fluctuation in finding the top matches than the navigational one.



(a)



(b)

Fig. 11: Keyword-based search: average, median, lower and upper quartile of: 11-point precision (a) and DCG (b), for Metamodel-independent, Structural, and Navigational index configurations.

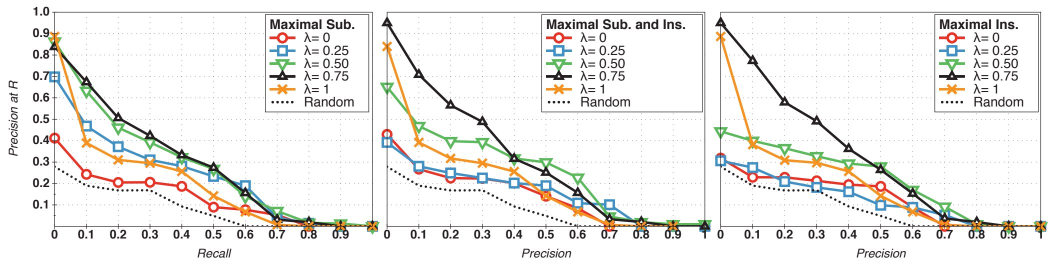
4.3.2. Content-based search. The evaluation of the content-based search first examined the influence of the λ parameter with respect to each weight configuration in the graph edit distance, adopting the *Levenshtein* string distance metric.

Table VIII summarizes the MAP values for different λ values and graph edit distance weight configurations. Figure 12a and Figure 12b respectively show the 11-point interpolated average precision and DCG results for the various weight configurations; each curve in one diagram corresponds to a specific value of λ.

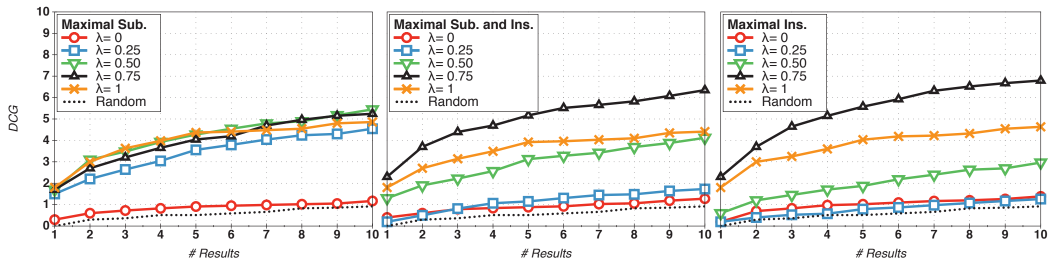
Table VIII: Content-based search: values of MAP for different values of λ and weight configurations in the graph edit distance.

Experiment	*Maximal substitution*	*Maximal subst. & insertion*	*Maximal insertion*
Random	0.19		
Only type contribution (λ=0)	0.34	0.32	0.29
High type contribution (λ=0.25)	0.56	0.34	0.24
Intermediate type contribution (λ=0.5)	0.74	0.55	0.38
Low type contribution (λ=0.75)	0.72	**0.83**	**0.86**
No type contribution (λ=1)	0.74	0.7	0.73

From Table VIII and Figure 12a and 12b, it emerges that neither the metamodel type alone nor the element label alone are the best options for node matching. When the *Only type contribution* configuration is used, the 11-point precision graphs show that, regardless of the adopted weights configurations, very few relevant documents are retrieved (curves show low precision values), which is confirmed by the DCG graphs and MAP values. Adding a “touch” of metamodel type knowledge to the node similarity function leads to better performance: the *Low Type Contribution* configuration (λ=0.75) emerges in most cases as the most viable trade-off between label and metamodel type



(a)



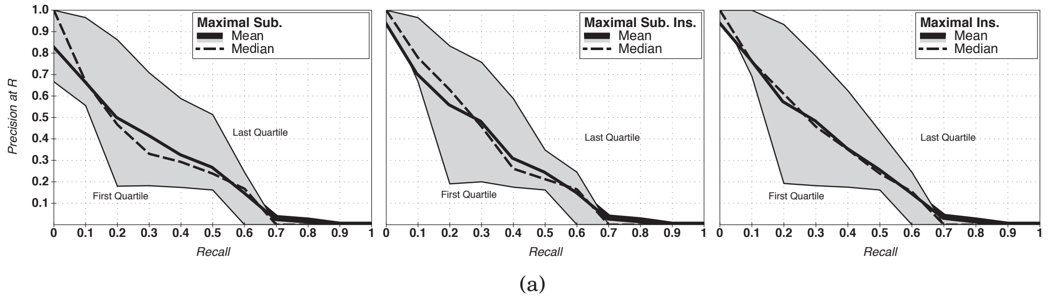
(b)

Fig. 12: Content-based search: 11-point interpolated average precision (a) and DCG (b) for different λ values and weight configurations (maximal substitution, maximal substitution and insertion, and maximal insertion).

information (up to 13% better than *No type contribution* and up to 57% better than *Only type contribution* in the MAP table).

The greater relative importance of element names over types in the best performing case is explained by the occurrence of false positive matches: overemphasizing metamodel types quickly leads to cases in which some project graph nodes representing a modeling concept present in the query (e.g., a given type of operation on data) are considered similar and thus matched to project nodes that operate on content unrelated to the query.

The DCG graphs (Figure 12b) suggest a correlation between the value of λ and the graph edit distance weight configuration policy. The spread among the curves at different values of λ is very limited for the *Maximal Node Substitution* configuration, and more sensible for the other two configurations. This shows that *Maximal Node Substitution*, which gives importance only to node substitution operations (i.e., similarity depends on finding as many "right" model elements as possible, and not, or less, on how the model elements are arranged or on missing model elements), makes the rank order of results less sensitive to the name-type tradeoff in the node similarity metrics, but for the case of λ=0 which remains dominated in all weight configuration policies. Symmetrically, the policies that emphasize node/edge insertions (*Maximal substitution and insertion* and *Maximal insertion*) achieve better MAP figures, but the rankings they produce are more sensitive to the tuning of λ. A possible interpretation of this phenomenon is that the *Maximal substitution and insertion* and the *Maximal insertion* policies, which penalize node and edge insertions in graph similarity, require the "right" node similarity function, to compensate the fact that even slight topological differences (e.g., differences in containment and linking, or missing model elements) in the query and the project model can push a relevant match down in the result list



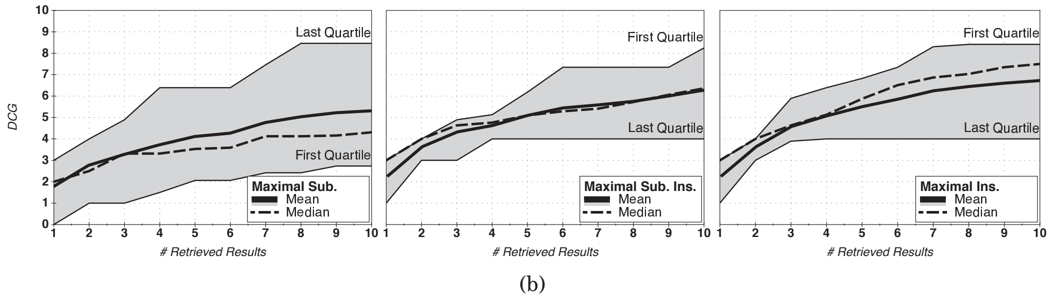
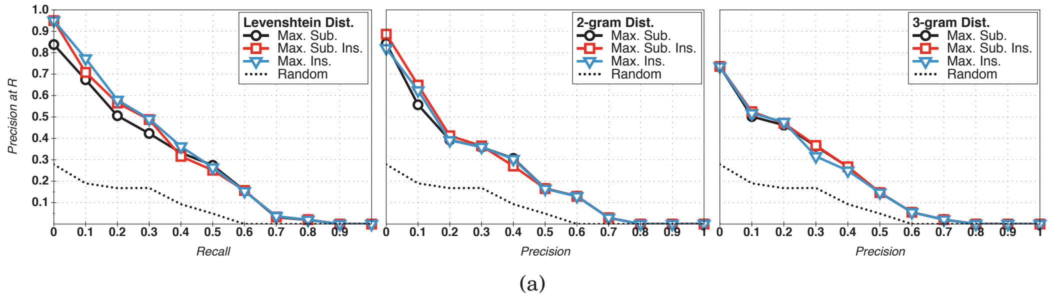


Fig. 13: Content-based search: average, median, and lower and upper quartile of 11-point precision (a) and DCG (b), with $\lambda = 0.75$



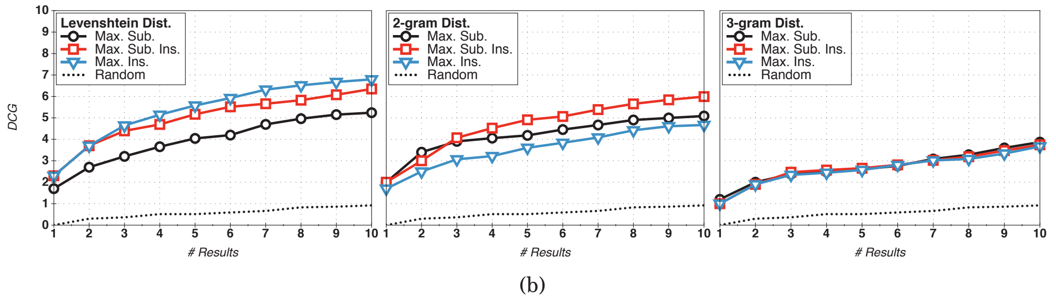


Fig. 14: Content-based search: 11-point interpolated average precision (a) and DCG (b) for of *Levenstheim*, *2-gram*, and *3-gram* string distances ($\lambda = 0.75$)

(and hence, the DCG curves for "wrong" λ values are more separated from the curve at the "right" value λ=0.75).

Figure 13 shows the average, median, and lower and upper quartile for 11-point precision and DCG curves. It confirms the performance improvement obtained when considering insertion operations, because in both the 11-point precision and DCG the distribution of differences shows less variations with respect to the *Maximal Node Substitution* configuration. However, Figure 13 also shows that the distribution of results is wider than the one shown in Figure 10 for keyword-based search; this means that the performance of the content-based scenario varies more across the sample queries.

Table IX: Content-based search: values of MAP for different string distance metrics.

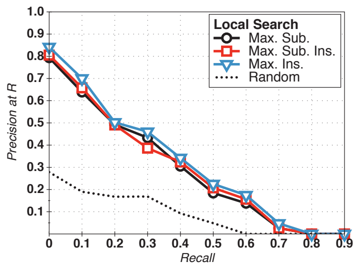
Experiment	*Maximal substitution*	*Maximal substitution and insertion*	*Maximal insertion*
Levenshtein distance	**0.72**	**0.83**	**0.86**
2-gram distance	0.72	0.75	0.78
3-gram distance	0.60	0.63	0.59

String similarity function comparison. Figure 14 shows the third experiment with content-based search, which evaluates the adoption of different string similarity functions. We set λ to 0.75 (*Low type contribution*) and evaluated the *Levenshtein* distance and the *N-gram* distance under the three graph edit distance configurations. The best results are obtained when using the *Levenshtein* distance. N-gram distance was tested for 2-grams and 3-grams. With respect to the *Levenshtein* distance, 2-grams respectively decrease the 11-point precision and DCG, for an average of 22% and 13%, while 3-grams decrease, on average, the 11-point precision by 43% and the DCG by 32%. Noteworthy, the three graph edit configurations perform consistently with both the *Levenshtein* and the *n-gram* distances, as the *Maximal Insertion* configuration outperforms the others. The performance behavior of each string distance metric is further confirmed by the MAP results reported in Table IX.

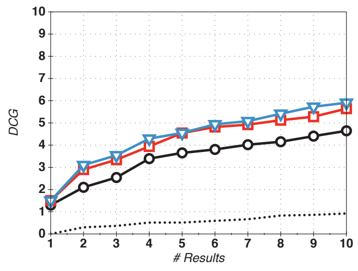
In summary, the best performance in both precision and ranking is obtained for a moderate metamodel type contribution in node similarity evaluation (*Low type contribution* a.k.a λ=0.75), *Levenshtein* distance for name similarity, and weight assignment configurations that appraise *both* node similarity and model topology. Therefore, textual similarity remains fundamental to achieve good results also in content-based search, but metamodel-dependent information *and* the topology of the query must be exploited to retrieve more relevant results and sort them in a more proper order.

4.3.3. Content-based search with locality constraints. As a last experiment, we compared content-based search with and without locality constraints for candidate mapping nodes. For both the original A-star and A-star with locality constraints we set λ to 0.75 (*Low type contribution*) and used *Levenshtein* distance in the node similarity function. Table X reports the MAP values for A-star and A-star with locality constraints. Figures 15 and 16 chart the 11-point precision and DCG curves. As can be noted in the above mentioned results, the application of locality constraints slightly worsens on average the performance of content-based search. Inspection of results reveals the following behavior:

— Locality constraints prevent the selection of disconnected matching nodes. This promotes in the result set matches with patterns that conform to the majority of the elements in the query and penalizes matches with projects that, although topically relevant, contain only partial reusable patterns scattered in different places of the

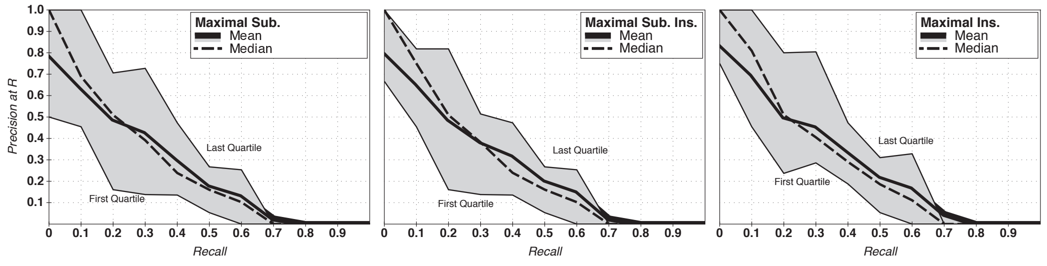


(a)

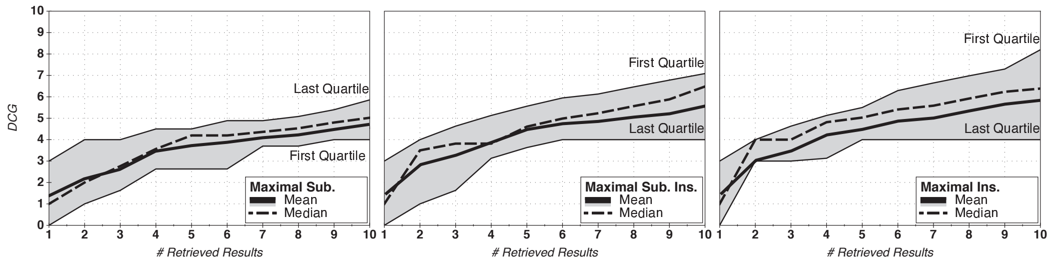


(b)

Fig. 15: Content-based search: 11-point interpolated average precision (a) and DCG (b) with locality constraints ($\lambda = 0.75$, *Levensthein* distance)



(a)



(b)

Fig. 16: Content-based search: average, median, and lower and upper quartile of 11-point precision (a) and DCG (b), with $\lambda = 0.75$, *Levenshtein* distance, and locality constraints.

model. This effect, in our experimental query panel and project repository, tends to favor local A-star.

— Some queries that perform well with A-star worsen their performance when locality of matching is applied, because the relevant results end up having less matching nodes, which lowers their rank score and thus diminishes their separation from not so relevant results; then it may happen that a less relevant result overcomes a more relevant one in the result list. This behavior tends to favor the original A-star.

— The two abovementioned effects compensate each other, with a slight predominance of the cases where locality worsens the performance.

Table X: Content-based search: MAP values of for A-star algorithm with local search.

Experiment	*Maximal substitution*	*Maximal substitution and insertion*	*Maximal insertion*
A-star	**0.72**	**0.83**	**0.86**
A-star + locality constraint	**0.70**	**0.72**	**0.77**

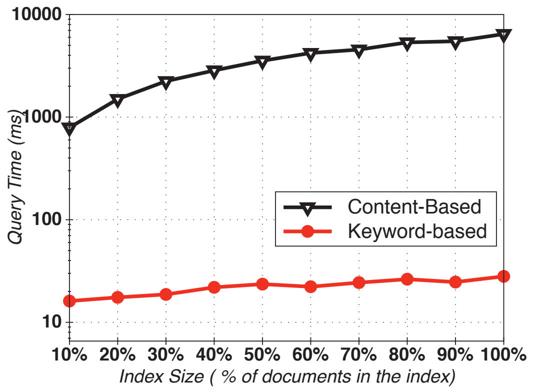


Fig. 17: Comparison of response time at varying number of indexed projects for keyword-based search and content-based search.

4.3.4. Query Execution Time. The last quantitative experiment compares the performance of the *keyword-based* and *content-based* search approaches with respect to *response time* required for query execution. All the experiments have been conducted on a machine equipped with Intel dual Core Processor 2.4GHz, 6GB RAM, and Windows 7 (64-bit) operating system; the reported values are averaged over 10 executions.

Figure 17 shows the query execution time for all the 10 queries considered in the experiments with respect to the index size. As expected, content-based search is considerably slower than keyword-based search, which executes in quasi-constant time. Despite the exponential complexity of graph matching, the content-based approach shows a quasi-linear correlation with respect to the index size for the considered repository, a result that confirms one of the findings of our previous works [Bislimovska et al. 2011b]. Notice that no query execution optimization (including optimized indexing of the repository) has been adopted during experiments and therefore we expect a wide range of possibilities for improving the performance of the content-based system.

4.4. User Study

The evaluation reported in Section 4.3 compared the results of the keyword- and content-based search systems with a gold data set constructed manually by experts and aimed at assessing the ability of each system to extract models similar to the user need, under the notion of structural and topical similarity provided by the experts. To evaluate the user-perceived utility of both systems during a development task, we conducted a controlled study organized in two distinct sessions, with the help of 25 industrial software developers (7 females and 18 males). Participants were volunteers with at least one project developed using WebML, engaged as follows:

— First, users had to fill-in a pre-experiment questionnaire, to provide demographic information and self-assess their experience with WebML on a 3-point Likert scale, ranging from 1 (novice) to 3 (expert). Of the 25 participants, 9 evaluated themselves as *expert*, 9 as *practitioner*, and 7 as *novice*.

— Before the start of the study, participants watched a video tutorial showing how to perform two different evaluations, described next.
— Next, users accessed an ad hoc Web application and performed the actual evaluation[7].
— Finally, users filled-in a post-experiment questionnaire, where they could provide feedback in free text format.

A pool of 10 tasks, defined in collaboration with the WebML experts and inspired to the development of the exemplary models used for the gold standard creation, was exploited in the user study. The following is an example of such tasks:

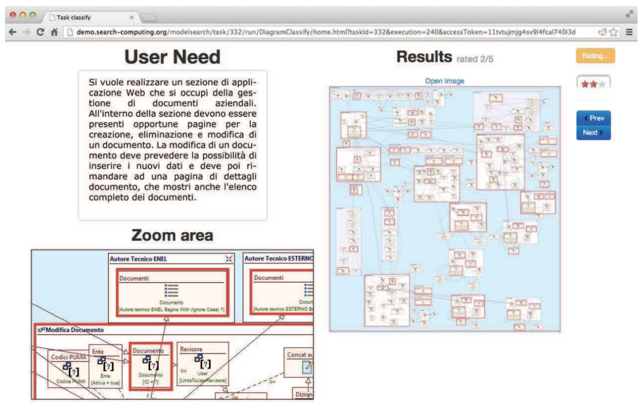
> *Assume you have to design a new Web application for the management of an e-commerce system. One of the requirements is the management of the sales operation; specifically, the site should contain a Web page devoted to the search of products in the catalogue; upon submission of the search conditions, the same page should show the list of products matching the user query. You want to identify existing projects (or fragments thereof) that can be reused to fulfill this requirement.*

Given a task description, the queries representing it in textual and WebML format were defined and respectively submitted to the keyword-based and to the content-based system, set-up in their best configuration (the *Metamodel-dependent Navigational* configuration for keyword search and the *Maximal Insertion* with $\lambda = 0.75$ and Levenshtein string similarity for content-based search, as discussed in Section 4.3). Results of query processing were collected and used for building the user evaluations described in the following sections.

4.4.1. User Study 1: single system evaluation. The first session elicited the users's judgement on the utility for reuse of each result computed by one of the two systems. Given a task such as the one exemplified above, users were presented the top-5 results, without disclosing which system they originated from. Users had to assess each result using a tertiary scale, where: (i) 0 meant not useful for reuse, (ii) 1 meant partially useful, and (iii) 3 meant very useful. Figure 18 shows the interface created for performing the User Study 1; it contains the task description and one result at a time, with commands for zooming the model, evaluating it, and scrolling to the other results of the top-5 result set. Each user evaluated the result sets of 10 tasks, assigned by mixing an equal number of responses to keyword- and content-based queries. To reduce learning bias and fatigue, the experiment was designed using a *graeco-latin square* scheme [Street and Street 1987; Joho 2011], with system type (keyword-based and content-based) and task as dependent variables. To minimize the impact of prior experience in WebML projects, tasks were assigned to participants randomly. To reduce bias due to the rank position, the order of presentation of results in the interface was random.

For each system, task, and result position, votes were averaged to calculate a global DCG curve for the keyword- and content-based systems, reported in Figure 19. Figure 20 shows the DCG curves, broken down task-by-task. Note that the DCG curves determined with the User Study 1 compare the result sets produced by the search systems with the best ordering of results emerging from the user's votes based on the perceived reusability of the project fragments with respect to the task description; conversely,

[7]The evaluation system is available for reviewers' consideration at `http://webml.org/webml/modelsearch/modelsearch-evaluation.jsp`.



Si vuole realizzare un sezione di applicazione Web che si occupi della gestione di documenti aziendali. All'interno della sezione devono essere presenti opportune pagine per la creazione, eliminazione e modifica di un documento. La modifica di un documento deve prevedere la possibilità di inserire i nuovi dati e deve poi rimandare ad una pagina di dettagli documento, che mostri anche l'elenco completo dei documenti.

Fig. 18: User Study 1: Interface of the evaluation system.

the DCG curves previously shown in Section 4.3 compare the results calculated by the system under multiple configurations with the gold standard created by the experts, who evaluated the technical quality of matches based on the degree of textual and/or structural relevance of the WebML area.

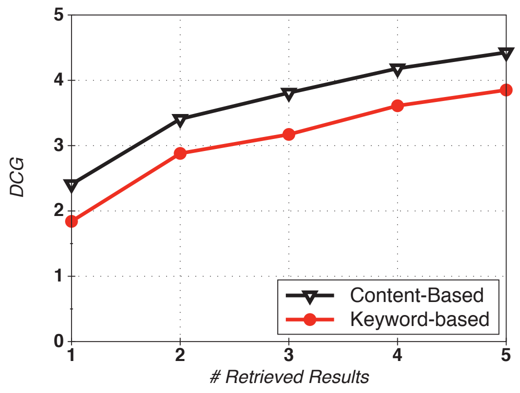


Fig. 19: User Study 1: DCG curves averaging the user evaluations (top-5 results).

4.4.2. User Study 2: system to system comparison. The second user study focused on the direct comparison of the top-5 result sets produced by the keyword- and content-based search systems. The experiment complements the first user study by including in the evaluation also the ranking performance of the two systems. To this end, we designed a pairwise comparison task, with the intent of reducing the cognitive effort that otherwise would be required for the separate evaluation of two ranked sets of models; the face-to-face appraisal of whole result sets supports not only the judgement about the relevance of the retrieved models, but also the direct comparison of the order in which these are presented. Given a task, users reviewed two result sets and indicated the one that in their opinion was globally more useful in terms of reuse, considering both the utility of the returned results, and their ranking positions. Figure 21 shows the

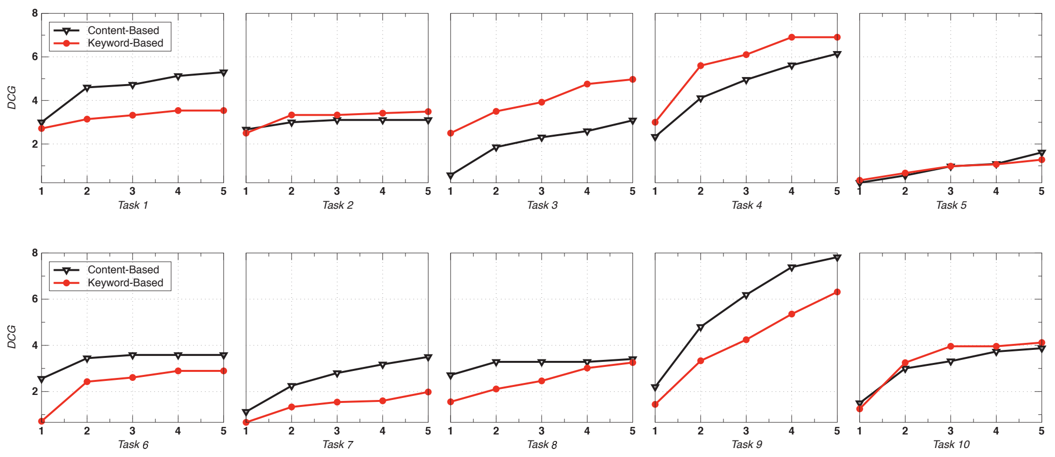
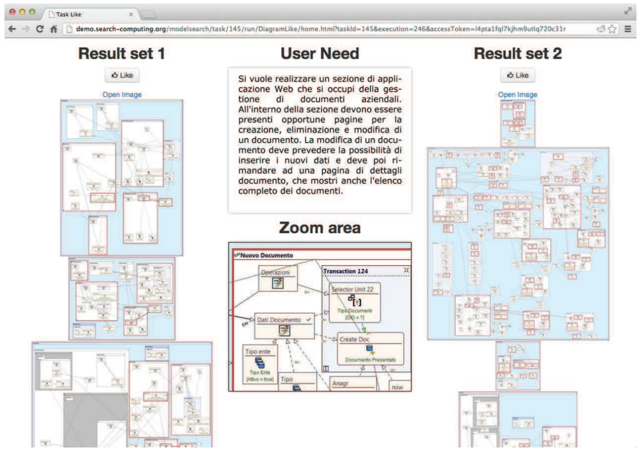


Fig. 20: User study 1: Task-by-task DCG curves (top-5 results).

evaluation interface developed for the second experiment: the description of the task is shown in the middle of the page, with the two result sets to be compared placed at its left and right. Figure 22 reports the direct comparison of the preferences for one system or the other, task by task.



Si vuole realizzare un sezione di applicazione Web che si occupi della gestione di documenti aziendali. All'interno della sezione devono essere presenti opportune pagine per la creazione, eliminazione e modifica di un documento. La modifica di un documento deve prevedere la possibilità di inserire i nuovi dati e deve poi rimandare ad una pagina di dettagli documento, che mostri anche l'elenco completo dei documenti.

Fig. 21: User Study 2: Interface of the evaluation system.

4.4.3. Analysis of Results. Coherently with the gold standard evaluation, both user studies show that the content-based search system provides, on average, better results than the keyword-based system, which suggests a correlation between the performance of a retrieval system and the user-perceived utility for reuse.

The DCG curves of Figure 19 show values similar to the ones described in Section 4.3, but with higher values for the content-based system; the histogram of Figure 22 show that the result lists produced by the content-based system have been preferred 60% of the times. Further analysis can be done by considering the task-by-task performance in Figure 20 and 22. The former shows how well the ordering of the result set *of a single system* adheres to the preferences expressed by the users; the latter shows,

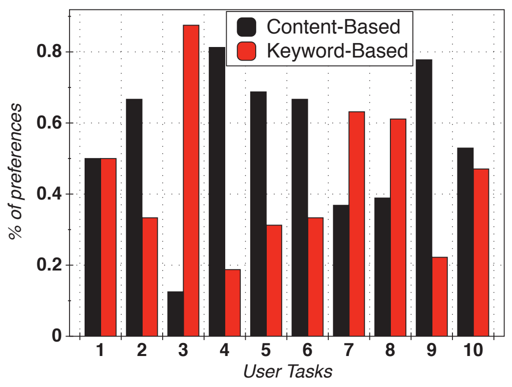


Fig. 22: User Study 2: Task-by-task preferences for the two systems.

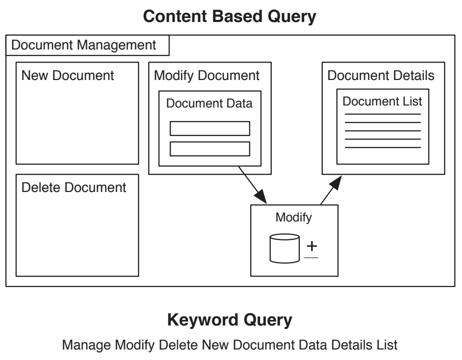
task-by-task, which system the users preferred, when confronted simultaneously with the result sets produced by both ones. Four situations emerge:

— *Content-based search is better* for `Task 1`, `Task 2`, `Task 6` and `Task 9`. Note that in Task 1 keyword- and content-based search get an equal share of preference in the direct comparison of result sets, but the DCG curve shows that the ordering of results is closer to the user's judgement for content-based search. Figure 23a reports an example of content-based query in this class: it expresses an object management pattern (distinct pages for the creation, modification, and deletion of instances) over the `document` entity. The better performance of content search is due to the nature of the query, which exploits a very characteristic design pattern and thus benefits from the match computed using graph similarity. Conversely, the corresponding keyword-based query contains rather frequent words ("document" occurs 81 times in the repository, "create" occurs 112 times, "modify" occurs 127 times, and "delete" occurs 118 times), which do not produce selective matches in the text retrieval system.

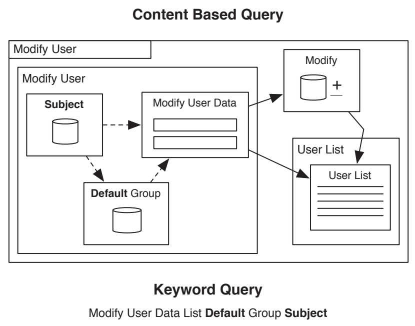
— *Keyword search is better* for `Task 3` (depicted in Figure 23b) and `Task 4`. In this case the selectivity of textual terms dominates the characteristics of the structural pattern. For instance, for `Task 3` the total number of occurrences of the term "default" in the repository is 5, while the term "subject" occurs 9 times. The specificity of these terms, which are rare in the repository, makes the keyword-based search more selective than the content-based counterpart, even if the content-based query exhibits a fairly articulated model. The greater number of preferences obtained by `Task 4` in the second user study is justified by visual bias in the comparison of result sets (see point (2) below), which diminishes the perceived utility of the retrieved set of results.

— *Comparable results* for `Task 7`, `Task 8` and `Task 10`. In this case both systems exhibit a comparable performance, with no clear winner or discordance between the direct comparison of results sets and the appreciation of each result in isolation. As an example, Figure 23c shows `Task 10`, which features a fairly complex structural model and good keyword selectivity (terms such as "dictionary", 44 occurrences, and "contract", 5 occurrences).

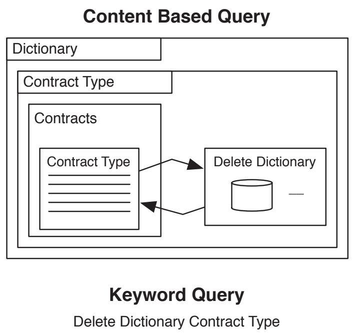
— *No satisfactory results* are retrieved for `Task 5` (shown in Figure 23d), which expresses a need formulated either as a model fragment with rather general structure and labels or as a bag of keywords having low selectivity. In such a case, both A-star graph matching and *TF-IDF* text matching do not perform well, as no distinctive feature of the query allows for high-confidence retrieval.



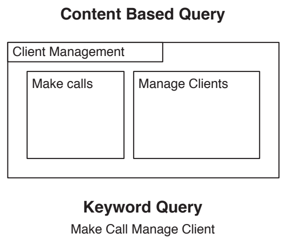
(a) `Task 9`: good content-based search



(b) `Task 3`: good keyword search



(c) `Task 10`: equivalent keyword and content search



(d) `Task 5`: no satisfactory results

Fig. 23: Example of task with a) good performance in content-based search, b) good performance in keyword-based search, c) equivalent performance, and d) unsatisfactory overall performance.

Further analysis of the results of the two user studies, also confirmed by the feedback provided by the users, show that:

(1) In some cases (e.g., `Task 1`, `Task 2`, `Task 6`, and `Task 8`) the main contribution to the utility of the result set is due only to a very relevant top-1 result, as shown by the DCG curve starting at the highest value for x=1 (i.e., 3) and then flattening out. In this case, the system retrieves a very good match to either the keyword-based or the content-based query, but then the other results are judged much less useful.
(2) Some other tasks (`Task 4` and `Task 9`) instead retrieve results that are perceived as good all over the result set, as shown by a steadily increasing DCG curve both in keyword-based and content-based search. In the direct comparison of the result sets, users tend to assign higher preference to content-based results though, even when the precision and order of the result set is judged better for keyword-based search (this is the case of Task 4). Post-experiment comments from the users suggest that the favorable perception for content-based search is influenced not by the relevance per se of the result, but by a visual bias induced from the highlight of the matching elements. Content-based search results match mostly elements that appear visually also in the content-based query and in its neighborhood, whereas

keyword-based search matches all elements that contain at least one keyword. In the abovementioned tasks, it was easier for the users to appreciate the reusability of the content-based result than of the keyword-based one, which had many highlighted elements and resulted confusing.

(3) Another factor that blurs the perceived differentiation between keyword- and content-based search is the size of the returned model element. Tasks like `Task 7`, `Task 8`, and `Task 10` happen to match well with rather large WebML areas, making it more difficult for the users to perceive the utility for reuse.

4.5. Discussion

4.5.1. Relevance of Metamodel information. Overall, the results of the experimental evaluation show that the inclusion of metamodel-dependent information in the model-search process is beneficial for performance; this is demonstrated both in the keyword-based search system, where the evaluated *Metamodel-dependent* strategies outperform the *Metamodel-independent* one, and in the content-based search system, where the injection of metamodel information in the node similarity function provided a considerable performance boost. However, in keyword-based search the very simple approach of extracting the text content from projects and indexing it with off-the-shelf IR tools still yields acceptable results (MAP = 0.77). This responds to the research question [*Q.2*] presented in Section 1.2.

4.5.2. Keyword- Vs. Content-Based Search. We compare keyword-based search and content-based search in their most performing settings, respectively the *metamodel-dependent navigational configuration* and the *Low type contribution, maximal insertion* configuration that uses *Levenshtein* distance.

The MAP values suggest that content-based search ($MAP = 0.86$) is overall more precise than keyword-based search ($MAP = 0.81$); however, results from 11-point interpolated average precision show that the best keyword-based experiment at $recall = 0$ slightly outperforms content-based search, as the former features a precision of 1. Content-based search provides better precision for $(0.1, 0.2, 0.3)$ recall levels (up to 30% of relevant projects); for greater recall levels the keyword-based search consistently outperforms content-based search. A similar performance profile resulted from the first experiment of the user study, where the DCG curves resulting from the evaluation of the top-5 results show that, on average, the content-based system is perceived as performing slightly better for reuse purposes.

Therefore, we might conclude that, in the evaluated setting, content-based search is suitable for applications where precision matters the most. On the other hand, keyword-based search can prove suitable in applications where recall is important (e.g., recommendation systems). Obviously, these considerations must be taken with care, because comparing information retrieval results across diverse systems and query paradigms can only give a coarse indication of the respective capabilities.
The quantitative experiment over gold data and the user study respectively respond to the research questions [*Q.3*] and [*Q.4*] presented in Section 1.2.

4.5.3. Search system design guidelines. The user study revealed possible sources of cognitive bias that may alter the perception of the utility of the retrieved results, even if they are relevant from a technical standpoint (i.e., they do contain the queried keywords or model fragment). These results suggest two recommendations for the designers of model search systems:

— *Project segmentation*: project segments (WebML areas in the case of our experiments) should be semantically meaningful as potential units of reuse and have comparable size.

— *Matching results highlight*: keyword-based search may be an interesting tool to recall more potentially relevant matches than content-based search, but it suffers from the visual overhead induced by the matches of many model elements of different types. As a possible countermeasure, the interface should support commands for toggling the highlight of selected metamodel types. In this way, the user could selectively turn on the highlight for the type of model element he is looking for (e.g., only for pages, or units of a given kind), exploiting metamodel information also for the visualization of results.

4.5.4. Project design guidelines. As a final remark, the findings about the performance of model-based search systems can be read also as recommendations for project developers and DSL designers. In general, using selective and precise textual labels for model elements is the first best practice to consider; given the importance of the text match component in both keyword- and content-based search, using scarcely descriptive labels and omitting comments to model elements obviously degrades search. Most of the reviewed projects contained no comments associated with model areas, pages, and meaningful patterns, even if this feature is supported by WebML and WebRatio. Another best practice is the adherence to standardized design patterns: many functions (e.g., the interfaces for performing CRUD operations on data, composition and sending of messages, and so on) can be modeled in standard ways, but the projects exhibited a lot of semantically equivalent but slightly different variants for doing the same thing. This (not so necessary) variability impacts the calculation of the graph edit distance, which is sensitive to link and containment topology. Last, DSLs that are designed to be extensible and incorporate third-party components, like WebML, should care for preserving the precision of the metamodel: a good classification taxonomy of custom components can help the metamodel type part of the node matching function of content-based search.

4.6. Threats to Validity

The paper provides two main contributions: an approach for applying metamodel-based search to model repositories; and a concrete experiment over a repository of models conforming to a specific DSL, namely WebML. While the former contribution is general, the latter, especially the quantitative results from the experiments, is relevant for the WebML case and cannot be directly generalized to other languages. However, the discussed method for studying the configurations of the search systems and for tuning their parameters can be reused. Also, as discussed in Section 3.1, WebML is a representative of a family of languages for interactive application modeling, which is also being proposed for standardization by the OMG. This makes the experiments described in the paper, although not directly portable to other DSLs, potentially useful for supporting the evaluation of search system in other languages for model-driven interactive application development.

While it would have been good to evaluate the system on multiple or larger repositories, finding realistic and sufficiently rich datasets has been challenging. Indeed, models are the core asset of MDE companies, which are therefore reticent to share them. Anyway, the repository we have been able to collect contains a considerable amount of model artifacts (a total of 19,246 searchable elements have been counted) and covers a wide spectrum of application domains. Based on the user feedback and on our empirical assessment of the repository, we think that the obtained results are accurately describing the system behavior for the WebML modeling language.

Another potential threat comes from the quality of the testbed and of the gold standard. We applied all the known techniques for reducing the bias of evaluators and we were not included in the set of experts evaluating the data. The same care has been applied to the definition and execution of the user study. While the projects in the repository could not be chosen (they were provided by WebRatio), the selection of the queries for the experiments was performed based on various language and dataset objective characteristics, to minimize the introduction of bias from our side. The selected number of tasks (10) has been deemed a reasonable compromise between the effort required for constructing the gold standard and the coverage of several aspects of the DSL and of typical design patterns that we observed in the provided repository.

The result of the user study could have been influenced by the number and expertise of the involved participants. However we believe that number of involved users (25) suffices for a meaningful evaluation, while the levels of expertise were fairly distributed.

Finally, a last factor that could have influenced the user study is the user interface we built for the purpose. This interface may have introduced exogenous complexity to the evaluated variables, e.g., due to factors such as the limitations of a browser-based interface, the system response time, the cognitive load associated with a new interface, time pressure, and the kind of interaction commands allowed. To minimize the impact of such factors, we provided equal training to all the participants, and did not pose a time limit for the execution of the evaluations. However, as we have commented in Section 4.4, model-driven search surely poses challenges in the system interface design, related to model complexity and size and highlight of matches, which we consider interesting future research directions to explore.

5. RELATED WORK

The problem of searching relevant artifacts in software repositories has been extensively studied in many academic works and widely adopted by the community of developers. This Section provides analysis of the state-of-the-art, classifying it by the type of retrieved software artifacts, i.e., components, source code and models. Furthermore, it associates our contribution with our previous works on model search.

Component Search. Searching software components from software libraries in an effective way for their reuse is an important research problem [Goguen et al. 1996]. One of the earlier proposed approaches is Agora [Seacord et al. 1998],a component search engine, which automatically generates and indexes a worldwide database of software products, classified by component model, allowing users to search for components by specifying the properties of its interface. *Merobase* [8] is an online tool for finding software components through simple text-based search, lookup capability and API search. The work in [Ben Khalifa et al. 2008] presents a structural and behavioral based technique for retrieval of software components, considering their heterogeneity, such as the domain, abstraction level and the underlying technology. [Platzer and Dustdar 2005] investigates the discovery and analysis of Web services using a vector space search engine to index descriptions of existing services.

Source Code Search. The need for searching source code for improving the process of software development and supporting software reuse resulted in emergence of several on-line tools and research works that implement and explore this problem. Some examples of existing on-line tools for sharing and retrieving source code are *Google code*, *Snipplr*, *Koders*, and *Codase*[9]. As explained in [Bozzon et al. 2010],the most basic solution is the case where queries in form of keyword(s) are simply matched to the code and the results are the exact locations where the keyword(s) appear in the matched code snippets. However, online tools allow advanced search by using reg-

[8]http://www.merobase.com

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

ular expressions (Google Codesearch), wildcards (Codase); supporting search of specific syntactical categories, like class names, method invocations, variable declarations (Jexamples and Codase); making the search more specific by indicating fixed set of metadata (e.g., programming language, license type, file and package names). Source code online tools also have to consider a way to compute a relevance score between the query and the matched source code, and present the corresponding results to the user [Bozzon et al. 2010]. Regarding this aspect, some approaches retrieve a list of matches without providing ranking, while others implement IR-style ranking using the standard TF/IDF measure, or ranking which besides the matches with the source code takes into account the project properties such as recency of the project, number of downloads, activity rates etc.

Research works for source code search are based on IR techniques [Frakes and Nejmeh 1987; McMillan et al. 2012] and techniques which employ the source code structure in the search [Bajracharya et al. 2009; Holmes and Murphy 2005]. Sourcerer [Bajracharya et al. 2009] is an infrastructure which provides foundation for building source code search engines and tools by sustaining large-scale indexing and analysis of open source code by exploiting the code structural information. [Holmes and Murphy 2005] describes a method for locating relevant code in an example repository by heuristically matching the structure of the code under development to the example code. Exemplar [McMillan et al. 2012] is an approach for finding highly relevant software projects from large archives of applications by using information retrieval and program analysis techniques. Sniff (Snippet for Free-From queries) [Chatterjee et al. 2009] is a Java code search technique which allows free-form queries in natural language for obtaining a set of relevant code snippets by combining API documentation with publicly available Java code. The work in [McMillan et al. 2011] describes Portfolio, a source code search system that provides retrieval and visualization of functions, and supports the analysis of chains of dependencies of the retrieved functions with the help of navigation and association models.

Model search. Model search approaches are not so abundant as those based on code retrieval, but a few systems have been described recently. Moogle is a model search engine that uses UML or Domain Specific Language (DSL) metamodels to create indexes for evaluation of complex queries [Lucrédio et al. 2010]. The work in [Gomes et al. 2004] 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 query represents a partial UML model and it may contain UML packages, classes or interfaces. Unlike our approach, Moogle supports only text queries which are refined by specifying the type of the desired model element to be returned, while [Gomes et al. 2004], is limited to UML model queries where the name of every model element is classified into a specific context synset (WordNet cognitive synonym)category. The techniques proposed in [Akehurst and Bordbar 2001] and [Calì et al. 2012] do not use IR or graph matching techniques but rely on query languages for UML models. [Akehurst and Bordbar 2001] uses the detailed semantics of UML and OCL with additional extensions for querying UML models. Calì et al. [Calì et al. 2012] study the problem of answering queries over UML class diagrams by relating it to the problem of query answering under guarded Datalog$\pm$, a powerful Datalog-based language for ontological modeling, in order to verify whether an instance of a system modeled by the UML class diagram satisfies a specific property. *WISE* [Shao et al. 2009] is a search engine that allows querying workflow hierarchies using keywords. CORE [Fernández et al. 2006], a tool for Collaborative Ontology Reuse and Evaluation, determines which ontologies from an ontology repository most appropriately describe a set of terms, by applying similarity measures.

The work [Mendling et al. 2007] analyzes similarity between process model behaviors, defined in terms of causal footprint. This raises the level of abstraction of the models and thus allows comparison of models specified in different languages (but still within the domain of business processes). Similarity is calculated with a vector model that considers nodes, look back links, and look ahead links of the causal footprints as features. Our work instead compares retrieval techniques based on purely textual representations and on graph representations upon which graph similarity is computed.

Other approaches define extensions of OCL (Object Constraint Language) for allowing queries over complex model repositories: for instance, [Kling et al. 2011] propose MoScript, a textual language for model querying and management. With MoScript, users can write scripts containing queries and manipulation instructions (e.g., transformations on sets of models) upon models and store them back in the repository. While the approach is metamodel-independent, the user is left in charge of writing complex OCL-like queries that only retrieve exact, non-ranked models. Our approach departs quite radically from the mentioned ones, as none of these systems considers query-by-example scenarios, and most of them require the usage of a query language for the specification of the query that is not suitable for end users.

Graph-based model search. Several approaches perform content-based search relying on graph matching. The work in [Grigori et al. 2010] introduces a BPEL ranking platform for service discovery employing graph matching, which finds a set of service candidates satisfying user requirements and ranks them using a behavioral similarity measure. Another behavioral similarity measure for artifact-oriented business processes, using the Petri Net notation, is proposed in [Liu et al. 2012] and is based on artifacts and their lifecycles, which reflect the behavior of a business process. Zhuge *et al.* [Zhuge 2002] implement an approximate matching approach based on SQL-like queries on ontology repositories. The focus is on reuse, based on a multi-valued process specialization relationship. Three similarity metrics for querying business process models are presented in [Dijkman et al. 2011]: label matching similarity, structural similarity, which considers the topology of models, and behavioral similarity, which focuses on the causal relations in models. The same authors propose a structural similarity approach in [Dijkman et al. 2009], which computes similarity of business process models, encoded as graphs, by using four different graph matching algorithms: a greedy algorithm, an exhaustive algorithm with pruning, a process heuristic algorithm, and the A-star algorithm. Although the graph-based part of our work is inspired by the abovementioned approaches, the existing works are restricted to queries over BPM models, which have a simpler syntax and semantics than a DSL for interactive application front-ends modeling. [Niemann et al. 2012] discusses a technique for process models retrieval based on clustering of related pairs, which combines semantic, string-based, and an hybrid metric for comparing process models. The related cluster pairs are then used to compute the overall process similarity. The main differences with respect to our work is the focus on business processes and the use of comparison mainly based on node labels rather than on structural information.

The works in [Kunze and Weske 2010; Qiao et al. 2011; Jin et al. 2011] present two-step approaches for graph-based search of Business Process Models repositories by applying filters to narrow the search space, and then performing graph matching on the filtered candidates only. [Kunze and Weske 2010] discusses an indexing approach for business process models based on metric trees (M-Trees), and a similarity metric based on the graph edit distance. The work in [Jin et al. 2011] introduces a structural technique for efficient retrieval of BPM models represented as Petri nets, with the help of an edge-based index which filters promising candidates, followed by a similarity computation of Maximum Common Edge Subgraph for the candidates that passed the filter. A two-level business process clustering and retrieval method that combines language modeling and structure matching is proposed in [Qiao et al. 2011]. The first level clustering is based on topic similarity, while the second-level clustering considers the detailed structure of processes within a cluster, and groups them according to their structural similarities using a graph-partition approach. In comparison to the graph-based part of our work, these approaches are limited to business process models, and they introduce filtering as another step in the processing to achieve more efficient search, which might be explored as a future direction in the graph-based search of models.

Other content-based approaches. Other approaches use specific algorithms for similarity search. The work in [Syeda-Mahmood et al. 2005] uses domain-independent and domain-specific ontologies for retrieving Web services from a repository by enriching their descriptions with semantic associations. A framework for model querying in the business process modeling phase, enabling reuse, support of the decision making, and querying of the model guidelines is presented in [Markovic et al. 2008].

Other approaches exploit domain knowledge, also in terms of ontologies, for formulating the queries; however, these approaches are typically bound to one specific kind of models (e.g., business process models). The work [Belhajjame and Brambilla 2011] 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. The work [Kiefer et al. 2007] proposed the use of semantic business processes and offer an approximate query engine based on iSPARQL to perform the process retrieval task and to find inter-organizational matching between business partners. With respect to our work, these techniques leverage on semantic descriptions of the models. This means that models need to be enriched with annotations from ontologies for improving the retrieval performance.

Our previous work. The results described in this paper are rooted in our previous work on model search. The early work [Bozzon et al. 2010] defined the problem of searching over DSL repositories and proposed a technical architecture for the case of keyword-based model retrieval; no evaluation with a gold standard for the keyword-based retrieval system was reported yet. The subsequent works [Bislimovska et al. 2011b; Bislimovska et al. 2011a] introduced the approach of content-based search, supported with the A-star algorithm, and reported a preliminary evaluation of results with a limited ground truth dataset compiled by the authors.

To the best of our knowledge, our work is the first one that systematically compares keyword-based and content-based search for models expressed in a Domain Specific Language, providing insight on the interplay between configuration parameters of the search engines, the structure of the modeling language, and the nature of the user's queries. Our approach to keyword-based search is inspired by information retrieval techniques and it is related to works like [Lucrédio et al. 2010]. With respect to this work, we focus on the comparison with content-based search and thus adopt a rather straightforward approach to indexing and search, which uses only the knowledge present in the text content and in the metamodel. The extension to a semantically richer treatment of the domain knowledge, e.g., for term expansion and domain-driven clustering of projects, can be easily envisioned for our approach. As for the content-based search, our approach mostly draws inspiration from graph-based works in the context of business process models, most notably, [Dijkman et al. 2009; Dijkman et al. 2011]. With respect to BPM-oriented content-based search, DSL-oriented search shares the mix of label and structural knowledge exploited in indexing and searching, but must cope with a richer language syntax and semantics, which we have considered in the design of parameter configurations.

6. CONCLUSIONS

In this paper we have addressed the problem of designing search systems for repositories of projects in a model-driven Web application development environment. We have contrasted two major approaches for the implementation of search: keyword-based and content-based. Extensive experimentation has been conducted with a sample of 10 queries against a real-world repository of 341 WebML areas, for which a gold standard set has been constructed that embodies what experts consider good responses to both keyword-based and content-based queries. Experiments have shown that even traditional text indexing techniques can deliver good performance for keyword-based queries, but adding metamodel knowledge to the index can improve accuracy. For content-based search, the conclusion is that matching the textual content of project is still important, but the system benefits from an appropriate injection of metamodel knowledge regarding both the types of the elements and the structure and topology of models. Furthermore, content-based search results exhibited greater variability and dependency on the queries than keyword-based results.

We underline once more that these results have been gathered in the context of a mid-scale experiment and cannot be generalized in an absolute way. They provide insight about what expert WebML modelers consider suitable queries and responses and about the way in which two different classes of information retrieval systems can be configured to respond to the expectations of these searchers. Nonetheless, we believe that the results presented in this work provide a number of interesting observations about the usage of keyword-based and content-based techniques for model search and therefore respond to the research questions we initially defined in Section 1.2.

Future work will develop along several complementary lines:

— On the search systems side, both keyword-based and content-based approaches will be further explored. The keyword-based approach will be expanded with a better exploitation of semantics and domain knowledge, both at query time (e.g., by means of keyword expansion), and at indexing time (e.g., by means of text feature extraction and project topical clustering). The content-based approach also lends itself to several investigation directions: other graph similarity functions and graph matching algorithms exist that could be profitably compared to the approach presented in this paper.

— On the usage of metamodel information, several additional options for embodying such knowledge in the search system can be evaluated. Besides using metamodel knowledge to segment projects and to influence the matching and ranking of the IR system, it is also possible to use it for mining relevant information from the project repository, such as term distribution, and for automating concept weighting based on the analysis of concept centrality in the collection of model element graphs.

— On the Web engineering side, it would be interesting to proceed with the analysis of other DSLs, and to compare search techniques for general purpose (notably, UML) and domain specific languages. We also plan to investigate how the introduction of explicit reuse-oriented constructs, e.g., WebML reusable modules, alters the structure of projects and the modeling style of developers, and thus impacts content-based search.

— On the evaluation side, we are building a system for large scale evaluation with "expert crowds". The CrowdSearch platform [Bozzon et al. 2012] is a general-purpose task crowdsourcing system that can be used to design user studies and deploy them on top of open social networks and/or closed groups. We plan to formulate as crowd tasks several types of search result evaluation questions, so to gather a large scale collection of queries and expert-validated result relevance scores, exploiting both open groups (e.g., LinkedIn MDE groups) and closed communities (e.g., the WebRatio developers network).

REFERENCES

ACERBIS, R., BONGIO, A., BRAMBILLA, M., AND BUTTI, S. 2007. Webratio 5: An eclipse-based case tool for engineering web applications. In *ICWE*, L. Baresi, P. Fraternali, and G.-J. Houben, Eds. Lecture Notes in Computer Science Series, vol. 4607. Springer, 501–505.

AKEHURST, D. H. AND BORDBAR, B. 2001. On querying uml data models with ocl. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*. Springer-Verlag, London, UK, 91–103.

ANDA, B., HANSEN, K., GULLESEN, I., AND THORSEN, H. 2006. Experiences from introducing uml-based development in a large safety-critical project. *Empirical Software Engineering 11,* 4, 555–581.

ARTECH CONSULTORES S.R.L. Last accessed August 2012. Genexus Marketplace. http://marketplace.genexus.com.

ATLANMOD GROUP. Last accessed August 2012. AtlanMod Zoos. http://www.emn.fr/z-info/atlanmod/index.php/Zoos.

BAJRACHARYA, S., OSSHER, J., AND LOPES, C. 2009. Sourcerer: An internet-scale software repository. In *Search-Driven Development-Users, Infrastructure, Tools and Evaluation, 2009. SUITE '09. ICSE Workshop on*. 1–4.

BELHAJJAME, K. AND BRAMBILLA, M. 2011. Ontological description and similarity-based discovery of business process models. *International Journal of Information System Modeling and Design (IJISMD) 2*, 47–66.

BEN KHALIFA, H., KHAYATI, O., AND GHEZALA, H. 2008. A behavioral and structural components retrieval technique for software reuse. In *Advanced Software Engineering and Its Applications, 2008. ASEA 2008*. 134–137.

BILENKO, M., MOONEY, R., COHEN, W., RAVIKUMAR, P., AND FIENBERG, S. 2003. Adaptive name matching in information integration. *Intelligent Systems, IEEE 18,* 5, 16–23.

BISLIMOVSKA, B., BOZZON, A., BRAMBILLA, M., AND FRATERNALI, P. 2011a. Content-based search of model repositories with graph matching techniques. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. SUITE '11. ACM, New York, NY, USA, 5–8.

BISLIMOVSKA, B., BOZZON, A., BRAMBILLA, M., AND FRATERNALI, P. 2011b. Graph-based search over web application model repositories. In *Proceedings of the 11th international conference on Web engineering*. ICWE'11. Springer-Verlag, Berlin, Heidelberg, 90–104.

BOZZON, A., BRAMBILLA, M., AND CERI, S. 2012. Answering search queries with crowdsearcher. In *Proceedings of the 21st international conference on World Wide Web*. WWW '12. ACM, New York, NY, USA, 1009–1018.

BOZZON, A., BRAMBILLA, M., AND FRATERNALI, P. 2010. Searching Repositories of Web Application Models. *International Conference on Web Engineering*, 1–15.

BRAMBILLA, M., BONGIO, A., BUTTI, S., FRATERNALI, P., KLING, W., MOLTENI, E., AND SEIDEWITZ, E. 2013. Interaction Flow Modeling Language (IFML). Standardization specification ptc/2013-03-08, Object Management Group (OMG), http://www.omg.org/spec/IFML/. March.

BUNKE, H. 2000. Graph matching: Theoretical foundations, algorithms, and applications. In *International Conference on Vision Interface*. 82–88.

CALÌ, A., GOTTLOB, G., ORSI, G., AND PIERIS, A. 2012. Querying uml class diagrams. In *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2012)*. Lecture Notes in Computer Science Series, vol. 7213. Springer, Tallinn, Estonia, 1–25.

CERI, S., FRATERNALI, P., AND BONGIO, A. 2000. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks 33,* 1, 137–157.

CERI, S., FRATERNALI, P., BONGIO, A., BRAMBILLA, M., COMAI, S., AND MATERA, M. 2003. *Designing data-intensive Web applications*. Morgan Kaufmann Publisher.

CHATTERJEE, S., JUVEKAR, S., AND SEN, K. 2009. Sniff: A search engine for java using free-form queries. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*. FASE '09. Springer-Verlag, Berlin, Heidelberg, 385–400.

CONALLEN, J. 2000. *Building Web applications with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

COOK, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*. STOC '71. ACM, New York, NY, USA, 151–158.

DIJKMAN, R., DUMAS, M., AND GARCÍA-BAÑUELOS, L. 2009. Graph matching algorithms for business process model similarity search. In *Proceedings of the 7th International Conference on Business Process Management*. BPM '09. Springer-Verlag, Berlin, Heidelberg, 48–63.

DIJKMAN, R., DUMAS, M., VAN DONGEN, B., KÄÄRIK, R., AND MENDLING, J. 2011. Similarity of business process models: Metrics and evaluation. *Inf. Syst. 36,* 2, 498–516.

DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1,* 1, 269–271.

FERNÁNDEZ, M., CANTADOR, I., AND CASTELLS, P. 2006. CORE: A tool for collaborative ontology reuse and evaluation. In *Proceedings of the 4th Int. Workshop on Evaluation of Ontologies for the Web (EON'06), at the 15th Int. World Wide Web Conference (WWW'06). Edinburgh, UK*. Citeseer.

FRAKES, W. B. AND NEJMEH, B. A. 1987. Software reuse through information retrieval. *SIGIR Forum 21,* 1-2, 30–36.

FRANCE, R., BIEMAN, J., AND CHENG, B. H. C. 2006. Repository for model driven development (remodd). In *Proceedings of the 2006 international conference on Models in software engineering*. MoDELS'06. Springer-Verlag, Berlin, Heidelberg, 311–317.

FRANCE, R., BIEMAN, J., MANDALAPARTY, S., CHENG, B., AND JENSEN, A. 2012. Repository for model driven development (remodd). In *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE Press, 1471 –1472.

GOGUEN, J., NGUYEN, D., MESEGUER, J., ZHANG, D., AND BERZINS, V. 1996. Software component search. *Journal of Systems Integration 6,* 1, 93–134.

GOMES, P., PEREIRA, F. C., PAIVA, P., SECO, N., CARREIRO, P., FERREIRA, J. L., AND BENTO1, C. 2004. Using wordnet for case-based retrieval of uml models. *AI Communications 17,* 1, 13–23.

GÓMEZ, J., BIA, A., AND PÁRRAGA, A. 2007. Tool support for model-driven development of web applications. *IJITWE 2,* 3, 65–78.

GÓMEZ, J. AND CACHERO, C. 2003. *Information Modeling for Internet Applications*. Idea Group Publishing, Hershey, PA, USA, Chapter OO-H Method: Extending UML to Model Web Interfaces, 144–173.

GREGORY, L. AND KITTLER, J. 2002. Using graph search techniques for contextual colour retrieval. *Structural, Syntactic, and Statistical Pattern Recognition*, 193–213.

GRIGORI, D., CORRALES, J. C., BOUZEGHOUB, M., AND GATER, A. 2010. Ranking bpel processes for service discovery. *IEEE Transactions on Services Computing 3*, 178–192.

HOLMES, R. AND MURPHY, G. C. 2005. Using structural context to recommend source code examples. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*. ACM, New York, NY, USA, 117–125.

HUTCHINSON, J., ROUNCEFIELD, M., AND WHITTLE, J. 2011. Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. ACM, New York, NY, USA, 633–642.

HYLTON, J. 1996. Identifying and merging related bibliographic records. Ph.D. thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY.

JÄRVELIN, K. AND KEKÄLÄINEN, J. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst. 20*, 422–446.

JIN, T., WANG, J., AND WEN, L. 2011. Efficient retrieval of similar business process models based on structure. *On the Move to Meaningful Internet Systems: OTM 2011*, 56–63.

JOHO, H. 2011. Diane kelly: Methods for evaluating interactive information retrieval systems with users - foundation and trends in information retrieval, vol 3, nos 1-2, pp 1-224, 2009, isbn: 978-1-60198-224-7. *Inf. Retr. 14,* 2, 204–207.

KIEFER, C., BERNSTEIN, A., LEE, H. J., KLEIN, M., AND STOCKER, M. 2007. Semantic process retrieval with iSPARQL. In *ESWC*. 609–623.

KLEPPE, A. G., WARMER, J., AND BAST, W. 2003. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

KLING, W., JOUAULT, F., WAGELAAR, D., BRAMBILLA, M., AND CABOT, J. 2011. Moscript: A dsl for querying and manipulating model repositories. In *SLE*, A. M. Sloane and U. Aßmann, Eds. Lecture Notes in Computer Science Series, vol. 6940. Springer, 180–200.

KRAUS, A., KNAPP, A., AND KOCH, N. 2007. Model-driven generation of web applications in uwe. In *MDWE* (2008-05-30), N. Koch, A. Vallecillo, and G.-J. Houben, Eds. CEUR Workshop Proceedings Series, vol. 261. CEUR-WS.org.

KUNZE, M. AND WESKE, M. 2010. Metric trees for efficient similarity search in large process model repositories. In *Business Process Management Workshops*, M. zur Muehlen and J. Su, Eds. Lecture Notes in Business Information Processing Series, vol. 66. Springer, 535–546.

LEVENHSTEIN, V. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*. Vol. 10.

LIU, H., LIU, G., WANG, Y., AND LIU, D. 2012. A novel behavioral similarity measure for artifact-oriented business processes. *Technology for Education and Learning*, 81–88.

LUCRÉDIO, D., DE M. FORTES, R., AND WHITTLE, J. 2010. MOOGLE: A model search engine. *Model Driven Engineering Languages and Systems*, 296–310.

MANNING, C. D., RAGHAVAN, P., AND SCHÜTZE, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

MARKOVIC, I., PEREIRA, A., AND STOJANOVIC, N. 2008. A framework for querying in business process modelling. *In Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI), Munchen, Germany*.

MCMILLAN, C., GRECHANIK, M., POSHYVANYK, D., FU, C., AND XIE, Q. 2012. Exemplar: A source code search engine for finding highly relevant applications. *Software Engineering, IEEE Transactions on*. To appear.

MCMILLAN, C., GRECHANIK, M., POSHYVANYK, D., XIE, Q., AND FU, C. 2011. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. ACM, New York, NY, USA, 111–120.

MENDIX. Last accessed August 2012. The Mendix App Store. https://appstore.mendix.com.

MENDLING, J., VAN DONGEN, B. F., AND VAN DER AALST, W. M. P. 2007. On the degree of behavioral similarity between business process models. In *EPK*. 39–58.

MESSMER, B. 1996. Efficient graph matching algorithms for preprocessed model graphs. Ph.D. thesis, University of Bern, Switzerland.

MIT. Last accessed August 2012. MIT process handbook. http://ccs.mit.edu/ph/.

MOHAGHEGHI, P. AND DEHLEN, V. 2008. Where is the proof? - a review of experiences from applying mde in industry. In *Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications*. ECMDA-FA '08. Springer-Verlag, Berlin, Heidelberg, 432–443.

NIEMANN, M., SIEBENHAAR, M., SCHULTE, S., AND STEINMETZ, R. 2012. Comparison and retrieval of process models using related cluster pairs. *Computers in Industry*.

OMG. 2011. Interaction Flow Modeling Language (IFML) Request For Proposal. http://www.omg.org/cgi-bin/doc?ad/11-12-06.

OUTSYSTEMS INC. Last accessed August 2012. The Agilenetwork Component Store. https://www.outsystems.com/NetworkSolutions/Home.aspx.

PLATZER, C. AND DUSTDAR, S. 2005. A vector space search engine forweb services. In ECOWS '05: Proceedings of the Third European Conference on Web Services. *Web Services, 2005. ECOWS 2005. Third IEEE European Conference on*, 62+.

QIAO, M., AKKIRAJU, R., AND REMBERT, A. 2011. Towards efficient business process clustering and retrieval: combining language modeling and structure matching. *Business Process Management*, 199–214.

REMODD TEAM. Last accessed August 2012. ReMoDD The Repository for Model-Driven Development. `http://www.cs.colostate.edu/remodd/v1/`.

ROSSI, G. AND SCHWABE, D. 2008. Modeling and implementing web applications with OOHDM. In *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, Eds. Human-Computer Interaction Series. Springer, London, Chapter 6, 109–155.

SANFELIU, A. AND KING-SUN, F. 1983. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics 13,* 3, 353–362.

SEACORD, R. C., HISSAM, S. A., AND WALLNAU, K. C. 1998. Agora: A search engine for software components. *IEEE Internet Computing 2,* 6, 62–70.

SHAO, Q., SUN, P., AND CHEN, Y. 2009. Wise: A workflow information search engine. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*. ICDE '09. IEEE Computer Society, Washington, DC, USA, 1491–1494.

SHAPIRO, L. AND HARALICK, R. 1981. Structural descriptions and inexact matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 5, 504–519.

STREET, A. P. AND STREET, D. J. 1987. *Combinatorics of Experimental Design*. Oxford University Press.

SYEDA-MAHMOOD, T., SHAH, G., AKKIRAJU, R., IVAN, A.-A., AND GOODWIN, R. 2005. Searching service repositories by combining semantic and ontological matching. In *Proceedings of the IEEE International Conference on Web Services*. ICWS '05. IEEE Computer Society, Washington, DC, USA, 13–20.

WEBRATIO S.R.L. Last accessed August 2012. The WebRatio Store. http://store.webratio.com.

YESSOFTWARE, INC. Last accessed August 2012. CodeCharge Marketplace. http://www.codecharge.com/marketplace.

ZHUGE, H. 2002. A process matching approach for flexible workflow process reuse. *Information & Software Technology 44,* 8, 445–450.

Received August 2012; revised February 2013; accepted September 2013