The Third Workshop on AI and Software Engineering: Breaking the Toy Mold. IJCAI-95, Montreal, Quebec, April 20-21, 1995

A Similarity-Based Retrieval Tool for Software Repositories*

Igor Jurišica
Department of Computer Science
University of Toronto, Toronto, Ontario M5S 1A4

August 18, 1995

Abstract

In this paper we present a prototype of a flexible similarity-based retrieval system. Its flexibility is supported by allowing for an imprecisely specified query. Moreover, our algorithm allows for assessing if the retrieved items are relevant in the initial context, specified in the query. The presented system can be used as a supporting tool for a software repository. We also discuss system evaluation with concerns on usefulness, scalability, applicability and comparability. Evaluation of the $\mathcal{TA}$**3** system on three domains gives us encouraging results and an integration of $\mathcal{TA}$**3** into a real software repository as a retrieval tool is ongoing.

Keywords: similarity-based retrieval, context, relevance, software repositories

1 Introduction

The central role of an information system is to store and to retrieve information. Even though storage and retrieval are simple tasks in general, they become a problem when cooperative and federative information bases are considered. In such systems it is usually impossible to predict how this information will be used in the future. Thus, there is a need for a flexible information retrieval systems. Flexible systems should help the user to retrieve information by posing an imprecise and/or incomplete query.

Our similarity-based retrieval system $\mathcal{TA}$**3** can be used as a building block of such retrieval systems. $\mathcal{TA}$**3** allows for imprecise query specification and for automatic query refinements in order to control amount and relevance of retrieved information. System's flexibility is accomplished by including context in the similarity definition and by using a special representation language (e.g., `Telos` (Mylopoulos et al., 1990) where attributes are treated the same way as objects). Thus, it is possible to diminish the difference between similarity in object attributes and relations.

In Section 2 we present few existing similarity-based retrieval (SBR) systems. The following section describes our approach to SBR. Section 3 discusses suitable methods for evaluating retrieval systems and comparing performance of individual systems to each other. Concluding remarks are presented in Section 4.

2 Similarity-based retrieval

A software repository is a special-purpose information system. Various retrieval techniques are being used for retrieving software components from a repository, e.g., library cataloging, database classification and query-based retrieval (Tedjini et al., 1990), formal specification (Rittri, 1991), information retrieval schemes (Díaz and Freeman, 1987), hypertext-based classification and retrieval (Garg and Scacchi, 1989), similarity-based organization (Schwanke, 1991), analogy-based retrieval (Lee and Harandi, 1993) and hybrid organizations (Constantopoulos et al., 1994).

*This research was supported by the Information Technology Research Center of Ontario. Author is indebted to J. Glasgow, J. Mylopoulos and D. Lauzon for helpful comments and suggestions.

The most appealing feature of similarity-based retrieval is that it allows for retrieving relevant information even if the user cannot specify the query completely and precisely. It also allows for iterative browsing through the repository. This becomes especially useful when a repository is large and created by many user over longer time period. In general, it is impossible to predict what and when will be required to be retrieved and also, in which form. This problem can be decreased by allowing for flexible and iterative similarity-based retrieval.

The CAReT system (Lee and Harandi, 1993) is similar to our work in the sense that it supports flexible retrieval of software components. CAReT is an analogy-based retrieval system applied to software design reuse. It also supports incomplete query specification as well as similar and exact matching of components. Its knowledge base consists of background knowledge and a design library. The system uses *is-a* and *part-of* links to navigate through the library. The retrieval mechanism uses a three-phase procedure: locate-evaluate-select. Flexible retrieval is supported by allowing modification of the level at which attribute matches occur. The system uses two levels of matching: object-based and attribute-based. The first level compares object classes, object type and attribute properties and their relationships to other object types to support retrieval. The second level compares attribute data types and properties.

AIRS (AI-base Reuse System) (Ostertag et al., 1992) is a AI-based library system for software reuse. AIRS is a hybrid system that integrates faceted indexing and a hierarchical frame system. Similarity between two components is based on a distance measure of their respective descriptions. The distance measure is based either on the closeness or the subsumptions relation. The closeness relation is intended to capture that new components can be constructed by modifying of existing constructs. The subsumption relation is defined among the components of the system and is intended to capture the idea that certain components can be built by composing several other components. Based on this measure, two components are considered similar if the overall difference between their corresponding terms is small. The main restriction of the system is its inflexibility. Because the components must be described in terms of all features, adding new features would involve modifying all components which are already stored in the library.

Another SBR system is presented in (Daudjee and Toptsis, 1994). Authors present a heuristic clustering scheme used to organize software repository automatically. By conducting a limited evaluation (on a repository of 150 items with a simple structure and by using only two test-queries) the average performance was 53% recall and 98% precision.

In (Cunningham, Bonzano and Smyth, 1995) an incremental retrieval system I-CBR is presented. In case-based reasoning a user specifies a case skeleton and wants to retrieve cases similar to it. I-CBR is a case-based reasoning system which partitions attributes describing a case into two groups - the ones which are known immediately and the ones that need some extra effort for specifying them. During the retrieval process, user specifies known attributes and the system returns a pool of possible case candidates. From the set of unknown attributes the most discriminating one is selected and the user is asked about its value. Thus, the initial pool of retrieved cases is processed to decrease its size.

SBR is used in databases as well (Jagadish, Mendelzon and Milo, 1995). This interest stems from the need to manage large amount of federated databases, such as available on an information highway. Users need to be able to retrieve not only exact matches but also similar and relevant items.

Many of the mentioned tools meet requirements of retrieval tools for software repositories - managability of large and complex data structures and flexible retrieval. Our system $\mathcal{TA}$**3** not only supports flexible retrieval of complex data structures, it also allows the user to control the retrieval process and relevance.

3 Context-Based Similarity Retrieval

In (Jurišica, 1994a; Jurišica and Glasgow, 1995) we defined a new theory for similarity-based retrieval which allows for more flexibility. In short, context is a parameter of a similarity relation which maps an information base onto a set of relevant (in terms of the context) items.

An item $\mathcal{I}$ is a representation of a real world object in a particular representation scheme. It is represented as a finite set of attribute-value pairs: $\mathcal{I} = \{< a_0, V_0 >, < a_1, V_1 >, \cdots, < a_n, V_n >\}$, where a_i is an attribute name and V_i is the corresponding value. A repository (Δ) is a finite set of items: $\Delta = \{I_1, I_2, \cdots, I_n\}$.

A context (Ω) is defined as a finite set of attributes with associated constraints on the attribute values: $\Omega = \{< a_1, C_1 >, \cdots, < a_k, C_k >\}$, where a_i is an attribute name and constraint C_i specifies the set of "allowable" values for attribute a_i. Our definition of context permits us to specify a variety of situations, ranging from completely ignoring an attribute or its value, through considering only certain values for an attribute, to considering only values that are not prohibited. In the following sections we demonstrate that this provides for a flexible retrieval specification and for a simple method of fine-tuning queries.

An item $\mathcal{I}$ satisfies a context Ω ($sat(\mathcal{I}, W)$) if and only if for all pairs $< a_i, C_i > \in \Omega$, there exists a pair $< a_i, V_i > \in \mathcal{I}$ such that $V_i \in C_i$ (i.e., the value of the attribute in the item satisfies the constraint). This definition can be relaxed so that an item may satisfy a context if it matches a given n number of attributes rather than requiring all the attribute values in the context be satisfied.

In the proposed theory, similarity is considered as a relation among a set of items, SI, a context Ω and a repository Δ. The relation $sim(SI, \Omega, \Delta)$ states that the items contained in set are considered similar for context Ω. We say that such a similarity relation holds when SI is defined as the set of items that satisfy the given context. Given a set of items SI, a repository Δ and a context Ω, similarity, $sim(SI, \Omega, \Delta)$, holds if and only if: $\forall X \in \Delta,\ X \in SI \leftrightarrow sat(X, \Omega)$.

The complexity of the retrieval task is characterized by the number of required comparisons which is, in the worst case, $\mid \Omega \mid \times \mid \Delta \mid$. Classification of items may improve the efficiency of performing the task by limiting the search space of the repository.

In (Jurišica, 1994a) we show that context-based similarity is monotonic, transitive and symmetric - features which have an obvious use for information retrieval. Monotonicity implies that if the user is not satisfied with the number of items that have been retrieved, then the context may be relaxed to retrieve some "less relevant" items or restricted to reduce the number of items. Additional motivation for monotonicity in retrieval has been presented in (Lauzon and Rose, 1994; Rissland et al., 1993).

For information retrieval, we express a query as a similarity relation and a list of matching criteria:

$$QUERY \ = \ [sim(SI, \Omega, \Delta),\ Criteria],$$

where sim is a similarity relation, SI is a variable corresponding to the set of items to be retrieved, Ω is an initial context, Δ is a search space and $Criteria$ specifies conditions for matching items to contexts.

If no matching criteria are specified then the repository Δ is searched for all items that satisfy the context Ω. Retrieved items will be placed in the variable SI and are considered relevant in a given context. If the number of retrieved items does not satisfy the user then the context can be adapted using relaxation or restriction. Relaxation involves extending the constraint set for one or more of the attribute values or removing attributes from the context. Restriction involves removing values from constraint sets or adding new attributes to the context (Jurišica, 1994a).

The retrieval process can be affected by specifying matching criteria in the query. First, for a cardinality criteria, a parameter $Some(n)$ states that in matching an item with a context at least n, rather than all, attributes of a context must be satisfied. Schema criteria allow for matching of attribute names and values or values alone. It is possible to place matching criteria only on subsets (or categories) of attributes, and thus treating different attributes differently. The process of applying matching criteria preserves the property of monotonicity. As n increases, the set of retrieved items is a subset of the less restrictive context. Similarly, allowing attribute and value matching or value matching only is a form of context relaxation.

The $\mathcal{TA}$**3** system has been implemented that exploits the monotonicity of context-based retrieval, restricting or relaxing a context in order to guide a retrieval process. Context relaxation can be executed by the user or by the system - cardinality and schema criteria or attribute values can be relaxed. Context restriction can be performed by the user or a limited automatic restriction can be applied - namely, cardinality or schema criteria can be automatically restricted.

User relaxation proves to be useful in domains where the user might have more information than is already encoded into the system, or where the user may be unsure of what attributes/constraints are the best for defining relevancy (i.e., when there is a need for iterative browsing through the repository). For example, an expert may relax a context by changing value constraints from a value $Heap\_sort\_algorithm$ to a less restricted set defined by the *Sort_algorithm*. This degree of fine-tuning is supported by the automatic restriction/relaxation implementation which uses hierarchies of terms.

4 System Evaluation

In order to assess usability and scalability of a new system one needs relevant measures, appropriate test-domain and performance results for systems performing the same task.

There are two main characteristics to be measured - *task-performance* and *time-performance*. The first one reveals if the system can perform the specified task (e.g., retrieve all relevant items from the repository). The second one measures how long does it take for the system to perform the specified task. In both cases scalability should be assessed, i.e., how these measures depend on the size and complexity of the repository.

Usually, task-performance of retrieval systems is measured by evaluating *precision* and *recall*, which are defined as follows:

$$Precision = \frac{\#\_of\_relevant\_retrieved\_items}{\#\_of\_retrieved\_items}, \quad Recall = \frac{\#\_of\_relevant\_retrieved\_items}{\#\_of\_relevant\_items}.$$

Even though these formulas seem to be straightforward, some discussion is necessary prior to using them for system evaluation. The following issues must be considered:

- **Relevance.** There are many possible definitions of relevance.
- **Goal of retrieval.** Users might prefer precision-oriented retrieval or recall-oriented retrieval.
- **Domain dependency.** System's performance is domain dependent.

4.1 Relevance

In different domains and for different systems there are distinct definitions of relevance (Greiner, 1994). Moreover, what is relevant in one situation (or for one user) may or may not be relevant in another situation (or for another user) (Jurišica, 1994b). To handle both issues, the retrieval system should be flexible enough to allow for changing relevance definition. If we assume that relevant item is a useful item then context-based similarity retrieval allows for the needed flexibility, as mentioned in Section 3.

Even with the flexibility allowed by explicitly stating the context, it may not be straightforward to assess the relevance (Jurišica and Glasgow, 1995) because one could measure relevance with respect to:

- user's belief;
- user submitted query;
- automatically relaxed or constrained query (i.e., the query actually used for retrieval).

It is apparent that the first point is the trickiest one since user's intentions might not be expressible by the available query language or in a used schema. Thus, two repository items might be believed to be distinct, yet their representation could be equal. This poses a problem for retrieval since both items would be retrieved at the same time and there could be situation when one of them would be irrelevant.

The second point is straightforward - user submits a query and the system retrieves items. If no relaxation/constraining of the query is performed, the system should retrieve only relevant items and should retrieve all of them. Note, that if the retrieval is iterative and the schema is sufficient, then submitted query should converge into user's belief.

However, as the third point suggests, if an automatic relaxation is used then the system might retrieve irrelevant items when measured with respect to the original query. This should not be considered as a problem due to a lowered precision, since there was a reason for relaxation, e.g., too few items (or none) were retrieved by using the original query.

Thus, as suggested in (Jurišica, 1995), one should not use only one precision and recall; definitions with respect to the user belief, user query and actual query used for retrieval should be used instead. It is apparent that the most subjective measure is based on user belief.

The other possibility to overcome the subjectivity of relevance definition is to use simple cross-validation. Even though double cross-validation is a better estimate of an actual performance of the system on the real problems, simple cross-validation is appealing since the relevance is defined – retrieved items can be compared to the set of items that should be retrieved.

4.2 Goal of Retrieval

As presented in (Gravano, García-Molina and Tomasic, 1994) in some situations a *precision-oriented retrieval* might be of interest, i.e., it would not be considered as a system's flaw if some relevant items are missed, if and only if only relevant items are retrieved. If *recall-oriented retrieval* is important, one would not be unhappy if several irrelevant items are retrieved, provided that all relevant ones are retrieved.

A truly flexible system should support both scenarios since different users might be using the same repository or a single user might change preferences while solving different tasks. In the $\mathcal{TA}$**3** system, we allow for goal flexibility by supporting two relaxation strategies - cardinality and attribute-value (as discussed in Section 3 and explained in more detail in (Jurišica and Glasgow, 1995)).

4.3 Domain Dependency

From our experience with complex medical domain, simpler technical domain (Jurišica, 1995) and simple software repository, system's performance depends on the domain. If the schema for the domain has sufficient details, i.e., individual items are separable, then the system's performance would measure its actual capabilities; otherwise the problem with representation will negatively affect retrieval.

This brings up an issue of how to compare individual systems. Unless they are evaluated on the same domain, using the same queries, the comparison is not an exact measure of their respective performances. Usually, because of security reasons, sharing a real repository by various parties (which are not working on the same project or for the same company) is impossible. This is also a reason why comparative evaluations are quite rare. One of a few exceptions is a field of machine learning, where researchers share some of their repositories (such as a technical domain mentioned earlier) allowing them to compare performance of systems on the same problems.

We believe that for software repositories the main interest is to have an easy-to-use but flexible system which is able to retrieve relevant items. The system should work even in a crude form, i.e., without any information classification. Even though time-performance is a significant measure, it has a secondary importance to task-performance.

Scalability issue is also domain dependent - if the domain is small and items are simple, then the task and time performance might not be meaningful. The system should be evaluated on a domain which is comparable to the size and complexity of an actual domain.

5 $\mathcal{TA}$3 performance

$\mathcal{TA}$**3** was evaluated on a medical domain, data on a rise time of a servomechanism (Quinlan, 1993) and a small software repository (Daudjee and Toptsis, 1994). The main objective of our evaluation was to test task performance. Time performance was evaluated only subjectively.

In the first two domains, retrieval system was a front-end for a reasoning system, namely, retrieved items were used for predicting unknown values for attributes. Thus, simple cross-validation was used and results were compared to actual items to avoid subjective assessing of relevance. For the lack of space we present only a summary of evaluation (more details are presented in (Jurišica, 1995)). Prediction could be successful only if relevant items were used. Thus, accuracy of prediction was a measure of retrieval capability of $\mathcal{TA}$**3** . Medical domain did not have sufficient representation which lowered the accuracy to a relative error between 0.15 ± 0.09 and 0.07 ± 0.04 with 95% confidence interval. Technical domain was more complete and for the 95% confidence interval the prediction accuracy reached relative error of 0.056 ± 0.009. It should be stressed that medical domain has sufficient complexity and reasonable size for being considered as a test-domain for a retrieval tool for software repositories. Technical domain was smaller and with a simpler schema but eight other systems were evaluated on it and thus could serve as a domain for performance comparison.

Performance on a small software repository is subjective due to relevance measure. It proved, however, that $\mathcal{TA}$**3** can use various relevance definitions and that it's performance can be altered from precision to recall-oriented.

We conclude that if the user has a strong idea what to retrieve, the value relaxation is the most suitable to use. If, however, the user does not know what to search for, then iterative retrieval is the fastest way to retrieve relevant items. First, by using cardinality relaxation, system retrieves relevant items with user and query recall equal to 100% but with lower user precision. Second, after having a better idea what is relevant and what attributes should be used, value relaxation can increase the precision while preserving recall.

6 Conclusion

Similarity-based retrieval tools can advantageously be used in building flexible software repositories. This claim is supported by a fact that in a real world it might be difficult or impossible to specify query request precisely and completely due to domain size and complexity. By the very nature of our reasoning, it is easier for us to say that we want to retrieve "something like a particular item, but with listed changes" than to specify required item directly and completely.

In this paper we presented a context-based similarity retrieval system applicable as a tool for software repositories. $\mathcal{TA}$**3** was evaluated on several domains (complex medical domain, simple technical domain and a small software repository) with promising results. The task performance of a retrieval component is domain independent whereas time performance is. The work on full-scale integration of the system as a supporting tool for a software repository is ongoing.

References

Constantopoulos, P., Jarke, M., Mylopoulos, J., and Vassiliou, Y. (1994). The software information base: A server for reuse. Technical Report DKBS-TR-94-1, University of Toronto, Department of Computer Science, Toronto, ONT.

Cunningham, P., Bonzano, A., and Smyth, B. (1995). An incremental case retrieval mechanism for diagnosis. Technical Report TCD-CS-95-01, Trinity College, Dublin, Ireland.

Daudjee, K. S. and Toptsis, A. A. (1994). Automatic organization of reusable software components in a multidimensional space. In De, P. and Woo, C., editors, *4^{th} Annual Workshop on Information Technologies and Systems*, pages 11–20, Vancouver, BC.

Díaz, R. P. and Freeman, P. (1987). Classifying software for reusability. *IEEE Software*, 4(16):6–16.

Garg, P. and Scacchi, W. (1989). ISHYS: Designing an intelligent software hypertext system. *IEEE Expert*, pages 52–82.

Gravano, L., García-Molina, H., and Tomasic, A. (1994). Precision and recall of *gloss* estimators for database discovery. Technical Report CS-TN-94-10, Stanford University, Department of Computer Science, Stanford, CA.

Greiner, R. (1994). *AAAI Fall Symposium Series on Relevance.* AAAI Press, Menlo Park, CA.

Jagadish, H. V., Mendelzon, A. O., and Milo, T. (1995). Similarity-based queries. In *Proc. of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, CA.

Jurišica, I. (1994a). Context-based similarity applied to retrieval of relevant cases. Technical Report DKBS-TR-94-5, University of Toronto, Department of Computer Science, Toronto, Ontario.

Jurišica, I. (1994b). How to retrieve relevant information? In Greiner, R., editor, *Proc. of the AAAI Fall Symposium Series on Relevance*, pages 101–104, New Orleans, Louisiana.

Jurišica, I. (1995). Retrieval and evaluation similarity functions applied to in-vitro fertilization and servo domains. Technical report, University of Toronto, Department of Computer Science, Toronto, Ontario. In preparation.

Jurišica, I. and Glasgow, J. (1995). Context-based similarity for case-based reasoning. Submitted.

Lauzon, D. and Rose, T. (1994). Task-oriented and similarity-based retrieval. In *Conference on Knowledge-Based Software Engineering*.

Lee, H.-Y. and Harandi, M. T. (1993). An analogy-based retrieval mechanism for software design reuse. In *Proc. of the Eight Knowledge-Based Software Engineering Conference*, pages 152–159, Chicago, IL.

Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. (1990). Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4):325–362.

Ostertag, E., Hendler, J., Díaz, R. P., and Braun, C. (1992). Computing similarity in a reuse library system: An AI-based approach. *ACM Transactions on Software Engineering and Methodology*, 3(1):205–228.

Quinlan, J. R. (1993). Combining instance-based and model-based learning. In *Proc. of the 10th International Conference on Machine Learning*, Amherst, MA.

Rissland, E. L., Daniels, J. J., Rubinstein, Z. B., and Skalak, D. B. (1993). Case-based diagnostic analysis in a blackboard architecture. In *Proc. of AAAI-93*, Washington, DC.

Rittri, M. (1991). Using types as search keys in functional libraries. *Journal of Functional Programming*, 1(1).

Schwanke, R. W. (1991). An intelligent tool for re-engineering software modularity. In *Proc. 14th Conference on Software Engineering*, pages 83–92, Austin, TX.

Tedjini, M., Thomas, I., Benoliel, G., Gallo, F., and Minot, R. (1990). A query service for a software engineering database system. In *Proc. Fourth Symposium on Software Development Environments*, pages 238–248, Irvine.