ArHex: Flexible Composition of Indexes and Similarity Measures for XML

Ismael Sanz, Rafael Berlanga
Universitat Jaume I, Spain
{isanz,berlanga}@uji.es

Marco Mesiti
Università di Milano, Italy
mesiti@dico.unimi.it

Giovanna Guerrini
Università di Genova, Italy
guerrini@disi.unige.it

Abstract

This work-in-progress paper describes the architecture of the ArHeX similarity-oriented XML processing toolkit [10]. ArHeX is designed to assist in the engineering of XML similarity-oriented applications, supporting the design and evaluation of suitable similarity measures and their associated indexes for a particular application.

1. Introduction

There are many XML database applications that require some notions of similarity. For example, in the integration and merging of highly heterogeneous XML databases, in which there is no common schema, exact approaches are impractical due to the great (and unpredictable) structural variations of the diverse sources. Therefore, similarity measures are required to identify similar information modelled using different structures or terminologies. Moreover, in systems handling objects with complex structures (e.g. protein data, music retrieval systems, or shape databases) there is a fundamental need of identify similar objects according a similarity function. These applications usually require carefully tailored similarity functions that consider both structure variations and vocabulary discrepancies occurring in the collection of documents.

A crucial design issue in the development of these kinds of similarity-based systems is that a single notion of similarity that "works best" in any situation does not exist. Different users in different contexts may require different similarity functions; for instance, a biologist may wish to retrieve proteins based on a comparison with a given amino acid sequence, while another one may issue a query asking for "malaria antigen" in the associated textual description, and a third user might combine both kinds of queries. This leads to the notion of *multi-similarity* systems [1], which are designed to support multiple notions of similarity simultaneously. These systems require the possibility to combine different similarity measures depending on the characteristics of the data that need to be handled, coupled with specifically tailored indexing structures that take the heterogenous nature of the data into account.

This latter aspect is crucial when handling heterogenous collections of XML documents because heterogeneity can occur at many different levels – from changes in the vocabulary used in the tag names to complex document-wide structural variations. Moreover, retrieval should be performed efficiently. Thus, similarity measures should be coupled with specifically tailored indexing structures for their efficient computation. Current proposals in the XML context [6] are usually tailored for a particular application and are hardly reusable in other domains.

Starting from these requirements, we are developing the ArHeX similarity-oriented XML processing toolkit. Key features of ArHeX are: (i) its ability to support collections which are heterogeneous at multiple levels of granularity, (ii) its flexible pattern-based query model, and (iii) its component-based architecture. These features allow ArHeX to support multiple user-defined similarity measures on top of efficient indexes. The following section describes the features of ArHeX and the techniques it employs.

2 Measure Definition and Composition

A wide variety of similarity measures, both general purpose and specifically tailored, has been proposed for XML [6] that produces good results in a specific kind of collection. General purpose measures include metric functions such as the Manhattan or Euclidean distances; IR-like matching coefficients such as the cosine with $tf \times idf$ weighting; entropy-based measures such as the Kullback-Leibler divergence; and structure-oriented techniques such as variants of the Tree Edit Distance algorithm. Specific tailored measures include [dire quali sono le misure specifiche per XML + Xdiff + altri].

These similarity measures are usually obtained through the composition of several "atomic" measures at a given *granularity level* of the XML hierarchy; for instance, a measure for complete XML documents is defined by evaluating the similarity of paths, which in turn requires some criterion to compare the elements contained in the path. The follow-

ing levels can be devised: the whole XML document, subtrees (i.e., portions of documents), paths, elements, links, attributes and textual content (of attributes and data content elements).

This indicates that it is possible to build frameworks for the implementation of complex XML similarity measures, based on a library of basic component functions (implementing the atomic measures). This does not exclude the employment of ad-hoc measures if necessary. For example, the designer of a similarity-based application in the domain of genetics may need to combine a generic text-oriented function that matches protein names with a highly specialized function that matches amino acid sequences.

We have thus defined a formal framework for the definition and composition of similarity measures relying on the granularity levels of documents and then specified software components implementing such functions that can be combined to obtain new measures specific for a given context.

A Formal Specification of Similarity Measures. Let $\mathcal{DOC}$ be a set of XML documents, and $G = \{\texttt{DOC}, \texttt{ELEMENT}, \texttt{ATTRIBUTE}, \texttt{PATH}, \texttt{TREE}, \texttt{CONTENT}, \texttt{LINK}\}$ the granularity levels at which documents in $\mathcal{DOC}$ can be compared. Given a granularity level $\gamma \in G$, a mapping function m_γ can be defined for extracting from a collection $C \subseteq \mathcal{DOC}$ the portions of documents at that granularity level. For example, the mapping functions $m_{ELEMENT}$ and m_{PATH} applied on a collection $C \subseteq \mathcal{DOC}$ return the set of elements (denoted $m_{\texttt{ELEMENT}}(C)$) and paths (denoted $m_{\texttt{PATH}}(C)$) occurring in C, respectively. A partial order relation $\prec_G$ between granularity levels of G can be defined, representing the containment relationship between granularity levels; it can be read as "is lower-level than". For instance, $\texttt{ELEMENT} \prec_G \texttt{PATH}$ because the paths in $m_{\texttt{PATH}}(C)$ are defined in terms of elements in $m_{\texttt{ELEMENT}}(C)$.

A similarity measure at a given granularity level γ can be defined as a function $f_\gamma : m_\gamma(C) \times m_\gamma(C) \rightarrow [0, 1]$. Given two similarity functions $f_{\gamma_1}$and f_{γ_2}, f_{γ_2} may be expressed in terms of f_{γ_1} if $\gamma_1 \prec_G \gamma_2$. For example, an instance of similarity function for XML paths is

$$f_{PATH}(p_1,p_2) = \frac{\sum_{i,j} f_{ELEMENT}(e_i, e_j)}{|p_1||p_2|}$$

where in the simplest case $f_{ELEMENT}(e_i, e_j) = 1$ if $e_i = e_j$ and 0 otherwise, and $|p|$ denotes the length of path p.

Measures as Components. In ArHeX, a *measure component* is an implementation of a similarity function at a given granularity level. According to the definitions above, a component can depend on one or more lower-level functions, but it is irrelevant *which* concrete lower-level function is used, as long as it belongs to the right granularity level. In addition, every component can be parameterized; for instance, a component that computes similarity at the textual level may allow the user to choose whether common words ("stop words") must be considered.

Using this framework, a large number of functions with different requirements can be defined for each granularity level. In order to characterize them the partial ordering of levels $\prec_G$ is extended into a *provides/requires hierarchy* typically used in software component engineering [9]. Each measure component is thus tagged with two extra properties, whose values are chosen from a predefined set of *features*: the *provides* property indicates the granularity level[1] at which the function operates (e.g., all node-level functions provide the feature *nodeMatch*), while the *requires* property indicates the features that must be provided by the lower-level functions on which the function relies. For instance, let C_{PATH} be a component that implements the f_{PATH} function defined above. Then, $provides(C_{PATH}) = \{pathMatch\}$ and $requires(C_{PATH}) = \{elementMatch\}$. For the component to be usable, another component that provides $elementMatch$ must be available.

The provides/requires hierarchy separates the representation of the components from the actual similarity function implemented. This allows us to implement components for generic operations (usually called "tie components" [9]) by computing an aggregated value out of the results of other components. For instance, consider the "weighted sum" tie component C_{WSUM}, that can be defined for a set of n components $\{C_i, \ldots, C_n\}$ at the same granularity level γ and a set of n weights $\{w_1 \ldots w_n | w_i \in \mathbb{R}\}$:

$$\begin{aligned}
parameters(C_{WSUM}) = &\quad \{\{C_1 \ldots C_n\}, \{w_1 \ldots w_n\}\} \\
provides(C_{WSUM}) = &\quad \textstyle\bigcap_i provides(C_i) \\
requires(C_{WSUM}) = &\quad \textstyle\bigcup_i requires(C_i) \\
f_{C_{WSUM}} : &\quad G \times G \rightarrow [0, 1] \\
f_{C_{WSUM}}(o_1,, o_2) = &\quad \sum_i w_i \times f_{C_i}(o_1, o_2)
\end{aligned}$$

***marco: ismael controlli le formule messe sopra?? Mi sarei aspettato che voglio definire una funzione a livello di path e utilizzo funzioni di similarita' a livello di element per cui il require di PATH dovrebbe essere uguale (o contenuto) nel provide di ELEMENT (o viceversa). Ma credo di sbagliare io da qualche parte.... ***

Fig. 1 shows an instantiation of a multilevel similarity measure that uses *WeightedSum*, the ArHeX implementation of C_{WSUM}. It combines components at the node, label, and text granularity levels. Note how the components

[1]marco: mi sembra di capire che il concetto di "granularity level" che c'e' qui e' un po' diverso da quello presentato prima sulla struttura dei documenti XML. E' giusto? Se si bisogna specificare un po' di piu' come e' fatta questa granularita'.

at the node similarity level are computed using a weighted sum. Edge labels represent the requires/provides hierarchy. *** mettere maggiori dettagli sulla figura ***

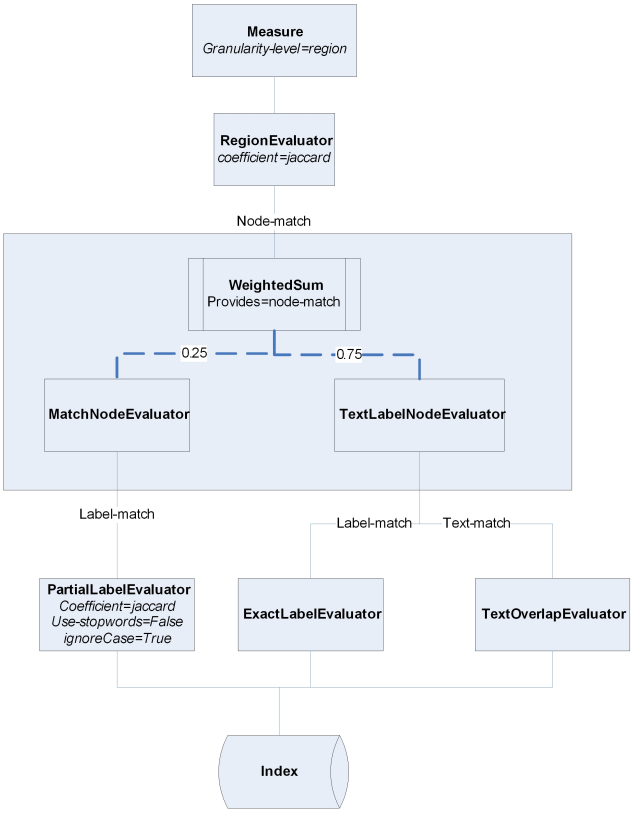

Fig. 1. Component structure of a similarity measure

A Formal Model of Components. We have just shown how to create components that can be glued together to form flexible similarity measures. However, the data engineer needs to answer higher-level questions. For instance: given a particular component, which other components are available to fulfill its requirements? Or, is this component sound (i.e. all of its requirements are correctly fulfilled)? This is particularly important in collaborative methodologies, in which sharing components among independently-working engineers is crucial.

Our approach is to encapsulate all of the required consistency rules in a declarative formalism, using a suitable Description Logic (DL). DLs provide a set of reasoning tasks (subsumption, instance checking, relation checking, concept consistency and knowledge base consistency) exploited in our context to automate many of the tasks that must be performed by the data engineer when designing a multi-similarity system. In addition, sufficiently expressive DLs provide "inverse functional" roles, which are exactly equivalent to candidate keys in a database. This is useful to support the semi-automatic specification of indexes as a combination of index components.

The mapping between the model previously outlined and a DL-based representation is straightforward. The rules (*A-Box* in DL parlance) can be expressed in terms of the concepts *Component* and *Feature*, and the roles *provides* and *requires*. The terminological knowledge (*T-Box*) reflects the current state of the system. An example of an instance of $RegionEvaluatorComponent$ used in Fig. 1 is:

$$\begin{aligned} RegionEvaluator &: Component \\ (RegionEvaluator, regionMatch) &: provides \\ (RegionEvaluator, nodeMatch) &: requires \end{aligned}$$

Complex concepts and terminological rules are built on top of these instances. For example, the set of all components at the "region" granularity level is expressed as

$$RegionEvaluatorComponent \doteq Component \sqcap \exists provides.regionMatch$$

The DL reasoner automatically classifies the $RegionEvaluator$ component as an instance of $RegionEvaluatorComponent$. Similar rules can be defined for consistency checking as outlined above.

3 Index Composition

Composition-based system have already been applied in the context of schema matching (e.g. COMA++ [4]) However, a purely functional approach where components operate directly over the actual data is clearly not appropriate in our large-scale Web data context. To compute these functions efficiently, it is necessary to exploit suitable indexes on the base XML data. For instance, consider a measure component for word similarity that uses $tf \times idf$ weights. The only way to compute such a measure efficiently in a large collection is to add the global frequency of each word to the index information. In ArHeX, this index information is called *index components*. ArHeX indexes are built as a composition of a base XML index plus a set of such components. This leads us to a second feature-based provides/requires hierarchy that associates measures to index components.

*** fare un esempio rispetto a figura 1 ***

4 Issues in Query Processing

The dual hierarchy *** specificare *** leads naturally to two levels of algebraic optimization. First, note that the tree obtained by the composition of measure components

is just a variant of an expression tree, and therefore familiar expression optimization techniques adapted from compiler technology, such as common subexpression elimination, can be applied on it. Then, using the mapping of measure components to index components, the resulting expression is translated to a physical algebra, on which cost-based optimizations can be performed to obtain an execution plan.

Our flexible measure definition model seems naturally fitted to a top-k query processor. There has been a lot of recent work describing approaches for optimizing top-k queries in an XML context; see for example [7, 8, 11]. Many of these approaches use variants of Fagin's threshold algorithm, which is suitable for monotonic aggregation functions [5]. In our context, however, there is no guarantee that the user-defined functions will have this property; even a simple case like a weighted sum with negative coefficients (used, for example, to penalize certain features) is non-monotonic.

We are currently evaluating techniques to perform approximate top-k queries in the presence of non-monotonic aggregation functions. To avoid evaluating all candidate results, we are studying sampling techniques to find promising subsets of the collection in which "good" results are more probable. Analogous techniques have been used for related problems in different contexts; see e.g. [2]. Our preliminary experiments show promising results, but many issues still remain open.

5 Summary

In this work-in-progress paper we have addressed the ArHeX approach to multi-similarity systems. ArHeX supports a flexible, component-oriented way to define measures, and Description Logic-based metadata facilities that help support multiple measures simultaneously. We also raise the open problem of efficiently evaluating top-k queries in non-monotonic aggregation functions, of practical importance in our context.

References

[1] S. Adalı, P. Bonatti, M. L. Sapino, and V. S. Subrahmanian. A multi-similarity algebra. *SIGMOD*, 402–413, 1998.

[2] D. Berleant, L. Xie, and J. Zhang. Statool: A tool for distribution envelope determination (DEnv), an interval-based algorithm for arithmetic on random variables. *Reliable Computing*, 9(2):91–108, 2003.

[3] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis. A methodology for clustering XML documents by structure. *Information Systems*, 31(3):187–228, 2006.

[4] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. *VLDB*, 610–621, 2002.

[5] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *J. Computer and System Sciences*, 66:614–656, 2003.

[6] G. Guerrini, M. Mesiti, and I. Sanz. An Overview of Similarity Measures for Clustering XML Documents. *Web Data Management Practices: Emerging Techniques and Technologies*, 56–78. Idea Group, 2006.

[7] R. Kaushik, R. Krishnamurthy, J. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. *SIGMOD*, 779-790, 2004.

[8] A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive Processing of Top-k Queries in XML. *ICDE*, 162–173, 2005.

[9] F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *IEEE Trans. Softw. Eng.*, 28(11):1056–1076, 2002.

[10] I. Sanz, M. Mesiti, G. Guerrini, and R. Berlanga. ArHeX: An approximate Retrieval System for Highly Heterogeneous XML Document Collections. Demo at *EDBT*, *LNCS*(3896), 1186–1189, 2006.

[11] M. Theobald, G. Weikum, and R. Schenkel. Top-k Query Evaluation with Probabilistic Guarantees. *VLDB*, 648–659, 2004.