Support for seamless data exchanges between web services through information mapping analysis using kernel methods

Buhwan Jeong, Daewon Lee, Jaewook Lee*, Hyunbo Cho

Department of Industrial and Management Engineering, Pohang University of Science and Technology (POSTECH), San 31, Hyoja, Pohang 790-784, South Korea

Abstract

A challenging issue to web services interoperability is seamless data exchanges between web services to be composed. A solution to this problem is to establish semantic mappings from an information item to another. To do that, we present an approximate information mapping analysis. We propose a kernel-based structural similarity measure for XML documents. Simulation results with industrial XML data show that the proposed kernel-based measure outperforms other existing methods.

© 2007 Elsevier Ltd. All rights reserved.

Keywords: Data exchange; Information mapping; Kernel method; Schema matching; Structural similarity; Web service

1. Introduction

Web services, which are self-described and self-contained software components over the Internet and use standardized XML messages to exchange data among them, have certainly become the key enabler to develop, integrate, and deploy enterprise applications. More adoptions of web services accordingly result in more interactions and data exchanges among services. Moreover, an application service is often made up of several component services to accomplish a complex task, thereby resulting in more data exchanges among these component services. Both data exchanges between services and within a service potentially cause interoperability problems such as syntax mismatches and semantics conflicts. Hence, assurance of interoperable data exchanges is a prerequisite to successful service composition and deployment.

The present paper proposes an information mapping analysis, which maps an information item in source data to one or more items in destination data. A mapping is established by the degree of similarity between compared information items. For that purpose, we propose a kernel-based similarity measure that exploits XML data by structure. We conduct preliminary simulations with industrial XML data for performance evaluation.

The rest of the paper is organized as follows: Section 2 briefly introduces the web service composition problem and then describes the information mapping analysis. Section 3 proposes the kernel-based similarity measure. Preliminary simulations are conducted in Section 4, and, consequently, Section 5 gives the conclusion remarks.

2. Problem statement

2.1. Web service composition

Web services are coupled via message (or data) exchanges. In other words, to use a service, an application sends a request message to it and, as a result, receives a response message from it. Such interactions are present both between application services and between component services. Here, we concentrate on interactions between component services, which are scaled up interactions between application services. Service composition is dealing with such interactions between component services (Younas, Awan, & Duce, 2006).
Service composition organizes and executes several relevant services to accomplish a complex task, when no single service offers that functionality and/or when a single service has very low quality (e.g., expensive to invoke, time-consuming, less available). A typical approach is to use processes that effectively integrate distributed, heterogeneous, and autonomous services (Zeng, Benatallah, Dumas, Kalagnanam, & Sheng, 2003). In this approach, interaction logics (e.g., control flow, data flow, and transactional dependencies) among services are expressed by a process model. An AND/OR graph, for example, captures sub-tasks in nodes and invocation precedence orders in directed arcs. For a composition model, we need to discover and assign appropriate component service(s) to each task node. At this discovery time as well as that composition time, data interoperability between services must be assured. At a certain moment, in other words, output data from services invoked already must include all the necessary information items of input data for the next service to be invoked. To this end, the information mapping analysis is necessary.

2.2. Information mapping analysis

Unless we use identical data in an agreed-upon manner, mismatches in representations and meanings are inevitable during data transfer from a service to another. Such mismatches depreciate the quality and performance of a composite service, thereby demoting the adoption of such a service. For this reason, assurance of seamless data transfers is a prerequisite to composing open services provided by anonymous developers. The seamless data transfer is assured when all necessary information is ready for service invocation. In other words, sources are sufficient for and correctly mapped to a destination in Fig. 1. In the figure, an information mapping tool merges, fragments, and transforms one or more source data into destination data to be sent into a service to be invoked. The sources come either from external applications or previous services already invoked, or both.

Particularly, we introduce the information mapping analysis (IMA) that checks whether the source data are sufficient for the destination data and, if they are, then establishes semantic mappings between them. The information mapping concept first comes from the information mapping test (Kulvatunyou, Ivezic, & Jones, 2005), which verifies that an application correctly processes and transforms data in a proprietary format into those in the standard XML format, and vice versa. Here, IMA uses the same data format – XML data. IMA has two folds: schema matching at design and discovery time and instance mapping at execution time. The former schema matching checks whether the data definitions, in XML Schema, of the destination are covered with those of the sources. A service provider describes and publishes the I/O message meta-types of its service in XML schema via WSDL (Web Service Description Language Christensen, Curbera, Meredith, & Weerawarana, 2001). Then, a service requester subscribes the description and makes its interfaces compatible with the message types. Therefore, this schema matching occurs at composition and/or discovery time. On the other hand, the latter instance mapping actually transforms information items, in XML instances, of the source messages into corresponding ones of the destination message. Hence, the instance mapping is necessary when invoking and integrating services. It is noted that the rest of the paper leaves issues about using ontology for information mapping and transformation. An ontology may explicitly state the relations from source information items to destination items, e.g., the element Facility, or the element MaxQuantity is a sub-type of the element Quantity. In other words, the paper uses approximate matching between schemas or between instances.

Simply, IMA compares every pair of information items, each from the sources and the destination, and selects the most plausible mapping(s). This is an approximate matching, in that the most similar pair of information items is mapped. Measuring similarity of an information pair is the core of IMA, and will be detailed in the next section. Here, we deal with practical issues arising in IMA. The first issue is the relation between sources and a destination, which is beyond the degree of their similarity. The sources must be sufficient for the destination. In other words, the sources must contain all the information items required by the destination. The first case that is ideal is that a source is identical to the destination – exact match. The second case is that a source or several sources collectively cover the destination – general match. In this case, since the sources have more information or different structures from the destination, the information mapping tool merges, segments, and re-organizes the sources to be identical to the destination. The third case is that the sources provide the destination with insufficient information – partial match, thereby requiring additional inputs from external applications.

The second issue is about information essentiality. For a specific information item I, if a source defines it as an optional item, the item may not appear in an instance document. On the other hand, suppose that the destination defines the same item I as mandatory. In this case, even though they are an exact match in the schema level, they can be a partial match in the instance level due to absence of the item I in the source instance. Therefore, a seamless data exchange is granted after such an information essenti-

![Fig. 1. Service connection via an information mapping tool.](image-url)
ality condition is met. A minimal interoperability condition between services is that all the mandatory items in the destination are also defined as mandatory in the sources. To sum up, IMA maps information items when they have identical essentiality as well as the same definition and structure. The next section will detail measurement of similarity between structured data.

3. Kernel-based structural similarity measure

This section entails a comprehensive review on the measurement of similarity for XML documents by structure, and proposes a novel structural similarity measure using kernel methods. We also give a brief introduction to kernel methods for they are the core enabler of the proposed measure.

3.1. Similarity measures for XML documents

An XML document is distinguished by its modular tree structure. In the literature, the tree structure provides the underpinning idea to propose a similarity measure for XML documents. Here are typical tree-based similarity measures for XML documents. The most intuitive measure is to compute the portion of common nodes in two documents, in terms of Jaccard coefficient. An advanced one is to compute the portion of common paths. Such node/path matching approaches appear in Amer-Yahia, Kou- das, and Marian (2005); Buttler (2004); Costa, Manco, Ortale, and Tagarelli (2004); Lee, Lee, and Kim (2001, 2002); Theobald, Schenkel, and Weikum (2003). The second approaches compute the total operation costs to make a tree identical to the other one. This is often called as tree edit distance (TED), and examples include (Dalamagas, Cheng, Winkel, & Sellis, 2003; Lian, Cheung, Mamoulis, & Yiu, 2004; Nierman & Jagadish, 2002; Yang, Kalnis, & Tung, 2005). In addition, minor approaches also appear such as (Yang, Cheung, & Chen, 2005) that extends a vector space model; (Bhavsar, Boley, & Yang, 2003) that uses weighted node matching; and (Flesca, Manco, Masciarei, Pontieri, & Pugliese, 2005) that uses the Fourier transformation. Note that even though we put similar approaches together, each approach uses slightly different representations and/or algorithms to measure similarity.

3.2. Kernel methods for structured data

Kernel methods have been successfully applied to various real-world problems such as pattern recognition, bio-/chem-informatics, stock market analysis, natural language processing, computer vision and control, etc. (Lee & Lee, 2005; Schölkopf & Smola, 2001). They convert a linear algorithm into a non-linear one by mapping the original samples into ones in a higher-dimensional Hilbert space $\mathcal{H}$ so that a linear model in the new space $\mathcal{H}$ is equivalent to a non-linear model in the original space $\mathcal{X}$. However, the transformation often leads the methods to a catastrophe due to highly intensive computational burdens and memory problems (Muller, Mika, Ratsch, Tsuda, & Schölkopf, 2001). Fortunately, this problem can be easily solved by using so-called kernel trick, which is originated from an idea that when an algorithm solely depends on the inner product between vectors, the scalar product can be implicitly computed in $\mathcal{H}$, without explicitly using or even knowing the mapping. That is, we just deal with a kernel function $K: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, not a mapping $\Phi: \mathcal{X} \rightarrow \mathcal{H}$, as $\Phi(x) \cdot \Phi(y) = K(x,y)$. The kernel trick is valid to non-vectorial data as well (Lee & Lee, 2005, 2007a, 2007b). Particularly, the following subsections comprehensively detail some kernel methods for manipulating textual data.

3.2.1. Vector space model

A traditional way to deal with textual data is a vector space model (VSM) or bag-of-words approach. With a long history (since 1970s), VSM is used as a state-of-the-art tool for ranking relevances of documents in the information retrieval field. It transforms a document into a numerical vector, each element of which is an indicator of presence or absence of a certain word in the document. Accordingly, a boolean VSM has only ones or zeros, while a real-valued VSM is filled with each term’s appearance frequency in a set of documents. ‘Term Frequency–Inverse Document Frequency (TF–IDF)’ weighting (Kobayashi & Aono, 2003; Saracoglu, Tutuncu, & Allahverdi, 2007) is widely used, in which the weight of the $i$th term in the $j$th document, denoted as $w_{ij}$, is defined by $w_{ij} = TF_{ij} \times IDF_i = TF_{ij} \times \log(n/DF_i)$, where $TF_{ij}$ is defined as the number of occurrences of the $i$th term within the $j$th document, and $DF_i$ is the number of documents (out of $n$) in which the term appears.

The similarity between two VSMs, denoted by $v_1$ and $v_2$, is simply calculated as the cosine of the angle $\theta$ defined by them as a distance metric, i.e., $d(v_1, v_2) = \cos \theta = \frac{v_1 \cdot v_2}{\|v_1\| \|v_2\|}$. This cosine similarity is a better distance metric than the Euclidean distance for information retrieval (Kobayashi & Aono, 2003). However, real-world data are so massive that the similarity measurement method requires too much computations and comparisons for real-time outputs. To overcome this problem, LSA (or LSI, Latent Semantic Analysis/Indexing) (Landauer, Foltz, & Laham, 1998) was proposed that is a fully automatic statistical technique for extracting and inferring relations of expected contextual usage of words in passages of discourse.

3.2.2. String kernels

VSM suffers from ignorance of the order appear in a document. Alternatively, string kernels accommodate the order of words, based on the similarity of two strings on the number of common subsequences. These subsequences need not be contiguous in the strings but the more gaps in the occurrence of the subsequence, the less weight is given to it in the kernel function. For example, take two strings
paper and article. Clearly the common subsequences are ‘a’, ‘e’, ‘r’, ‘ae’ and ‘ar’, each of which is represented in an exponential decay penalty – ‘a’: ( 1 \lambda ) , ‘e’: ( 2 \lambda ) , ‘r’: ( \lambda^2 ) , ‘ae’: ( \lambda^3 ) , and ‘ar’: ( \lambda^4 ) , that is, ( \lambda^1 , \lambda^2 , \lambda^3 , \lambda^4 ) vs. ( \lambda, \lambda^2, \lambda^3, \lambda^4, \lambda^5 ) – after applying penalties based on the gaps in the occurrence of a subsequence (i.e., the total length of a subsequence in the two strings) with a constant decay factor \lambda \leq 1. The kernel function is then simply the sum over these penalties, i.e., \( K(\text{paper}, \text{article}) = \lambda^1 \cdot \lambda^1 + \lambda \cdot \lambda^1 \cdot \lambda^1 + \lambda^1 \cdot \lambda + \lambda^1 \cdot \lambda + \lambda^3 \cdot \lambda^2 = \lambda^{10} + \lambda^6 + 3\lambda^3 \). A formal definition of the basic string kernel, a.k.a., string subsequence kernel (SSK), is provided as follows:

**Definition 1 (String Subsequence Kernel)**

Let \( S \) be a finite alphabet. A string is a finite sequence of characters from \( S \), including the empty sequence. For string \( s \) and \( t \), we denote by \(|s|\) the length of the string \( s = s_1 \ldots s_{|s|} \), and with \( s \) the string obtained by concatenating the strings \( s \) and \( t \). The string \( s[i:j] \) is the substring \( s_i \ldots s_j \) of \( s \). We say that \( u \) is a subsequence of \( s \), if there exist indices \( i = (i_1, \ldots, i_{|u|}) \), with \( 1 \leq i_j < \ldots < i_{|u|} \leq |s| \), such that \( u_j = s_{i_j} \), for \( j = 1, \ldots, |u| \), or \( u = s[i] \) for short. The length \( l(i) \) of the subsequence in \( s \) is \( i_{|u|} - i_1 + 1 \). We denote by \( \Sigma^* \) the set of all finite strings of length \( n \), and by \( \Sigma^* \) the set of all strings, i.e., \( \Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \). We now define feature spaces \( F_n = \mathbb{R}^{\Sigma^n} \). The feature mapping \( \phi \) of a string \( s \) is given by defining the \( n \) coordinate \( \phi_{n}(s) \) for each \( u \in \Sigma^* \). We define \( \phi_{n}(s) = \sum_{u\in\Sigma^n} \lambda^{l(i)}, \) for some \( \lambda \leq 1 \). These measures capture the number of occurrences of subsequences in the string \( s \) weighting them according to their length. Hence, the inner product of the feature vectors for two strings \( s \) and \( t \) gives a sum over all common subsequences weighted according to their frequency of occurrence and lengths

\[
K_{n}(s, t) = \sum_{u \in \Sigma^*} \langle \phi_{u}(s) \cdot \phi_{u}(t) \rangle = \sum_{u \in \Sigma^*} \sum_{\lambda_{i:j}} \lambda^{l(i)+l(j)}. \tag{1}
\]

Since a direct computation of these features would involve \( O(|\Sigma|^n) \) time and space, a recursive computation in \( O(n|s||t|) \) is provided in Lodhi et al. (2002). The \( K(s, t) \), i.e., the inner product of the feature vectors, is the similarity between the string \( s \) and \( t \) (Vert, Tsuda, & Schölkopf, 2004).

Alternatives to SSK have been used in Leslie, Eskin, and Noble (2002) Paas, Leopold, Larson, Kindermann, and Eickelker (2002) where only contiguous substrings of a given string are considered. A string is then represented by the number of times each unique substring of (up to) \( n \) symbols occurs in the sequence. This representation is known as the spectrum of a string or its \( n \)-gram representation. Extensions of those basic string kernels are found in Cancetta, Gaussier, Goutte, and Rendels (2003) Saunders, Tschach, and Shawe-Taylor (2002) where the characters replaced with words or syllables – word-sequence kernel – as well as soft matching is allowed. These extensions yield a significant improvement in computation efficiency for large documents.

**3.2.3. Tree kernels**

Real-world data are often more structured such as in forms of labeled ordered trees than sequential strings. The key idea to capture such structural information about the trees in the kernel function is to consider all sub-trees occurring in a parse tree, i.e., parse tree kernel (Collins & Duffy, 2002). Here, a subtree is defined as a connected sub-tree of a tree such that either all children or no child of a vertex is in the subgraph. The parse tree kernel for two trees \( T_1 \) and \( T_2 \) is defined as \( K(T_1, T_2) = \sum_{i=1}^{T_1} h_i(T_1) h_i(T_2) \), where \( h_i(T) \) is the number of times the \( i \)th subtree occurs in tree \( T \). In addition, for sets of vertices \( v_1 \) and \( v_2 \) of the trees \( T_1 \) and \( T_2 \), let \( \mathcal{S}(v_1, v_2) \) be the number of subtrees rooted at vertex \( v_1 \) and \( v_2 \) that are isomorphic. Then, the kernel can be computed as \( K(T_1, T_2) = \sum_{i=1}^{v_1 \in \mathcal{S}} h_i(T_1) h_i(T_2). \) This computation has time complexity \( O(|v_1||v_2|) \). A recent tree kernel is an extension to string kernels, in which a sequence of node labels in the order of a depth-first traversal is constructed and each node label is identified as a symbol (Vishwanathan & Smola, 2003). The original trie ought to be ordered.

**3.2.4. \( \lambda \)-Weighting**

One of the most critical factors to determine the kernels’ performance is the choice of the decay factor (i.e., \( \lambda \)), regardless the use of the exponential function. Compared with the original string kernel using an identical \( \lambda \) for every character, Saunders et al. (2002) introduces a different \( \lambda \)-weighting strategy that assigns a different weight \( \lambda_c \) to each character \( c \in \Sigma \). In addition, Cancedda et al. (2003) applied the same concept to the word-level. For example, the subsequence ‘cat’ from a string ‘cart’ would receive the weight \( \lambda_c \lambda_o \lambda_r \lambda_s \). Therefore, the weighted string kernel \( K^\omega \) of two strings \( s \) and \( t \) is defined as

\[
K^\omega_n(s, t) = \sum_{u \in \Sigma^n} \langle \phi^\omega_{u}(s) \cdot \phi^\omega_{u}(t) \rangle \tag{2}
\]

Since a direct computation of these features would involve \( O((|\Sigma|^n)^2) \) time and space, a recursive computation in \( O(|s||t|) \) is provided in Lodhi et al. (2002). The \( K(s, t) \), i.e., the inner product of the feature vectors, is the similarity between the string \( s \) and \( t \) (Vert, Tsuda, & Schölkopf, 2004).

The evaluation of \( K^\omega \) can be computed in a similar way to that of the original string kernel (Saunders et al., 2002). The use of different decay factors is one way of incorporating prior knowledge into the string kernel.

**3.3. Proposed structural similarity**

Next, we deploy the ideas how to use the kernels method to measure structural similarity between XML documents. To this end, we first present an interface representation that immediately captures the information about an element, including its label, attributes, related elements, etc. Based on this representation, we apply the string kernel-oriented tree kernel with modifications to compute the structural
similarity. The key idea is a combination approach of translation of an XML DOM tree into a normalized string (Vishwanathan & Smola, 2003), followed by an application of a word-sequence kernel to the normalized strings (Cancedda et al., 2003).

3.3.1. Interface representation

Before describing the interface tree, we introduce some assumptions and notations. An XML document (X) is equivalently represented in a DOM tree (D), in which a node (n) represents either an element (e) or an attribute (a), therefore the label of a node is assigned by that of the element or attribute, and no other auxiliary information, e.g., content (i.e., text value of an element or attribute), instruction comment, or annotation, is expressed. The exclusion of contents is because more semantics which an XML document (not an HTML document) ultimately intends to capture are encapsulated in elements rather than the contents (Lee et al., 2001), as well as the paper concentrates only on the structural similarity. For generality, we assume that the element nodes to be compared are inner nodes, such that they have a parent node and one or more child nodes. If a node has no parent node, then it becomes the root node (i.e., an XML document itself, n1). On the other hand, a node becomes a leaf node when no child node is attached. In addition, attribute nodes are ignored except those belonging to the element node of interest.

An interface representation contains three types of information about a node: self-description of the node (i.e., node label and its attributes), its context information by its ancestor nodes (i.e., a path from the root node to its parent node), and its semantics1 by its descendant nodes (i.e., a sub-tree whose root is the concerned node and no attribute node is allowed). Therefore, the information required is represented in a tuple 1 = (A, T, D), where A is a sequence of ancestor node labels; T is the node label and a list of its attribute labels; and D is a sequence of its descendant node labels in the order of a depth-first traversal. Note that the third field is applied the same treatment used by the tree kernel of Vishwanathan and Smola (2003). The root (or a leaf) node has no data in the first (third) field, i.e., A = φ (or D = φ). More specifically, the tuple is serialized to a sequence of words in the order, and then we get a long string. The reason behind this interpretation is that the context of a node is defined by its ancestor nodes, i.e., A, while its precise semantics is realized in its descendant nodes, i.e., D.

Before using this string, we need a normalization process that makes the string have only words in their basic form (e.g., ‘Id’ → ‘Identifier’). The normalization recursively consists of (1) tokenization, which separates a compound word into atomic dictionary words; (2) lemmatization, which analyzes these words morphologically in order to find all their possible basic forms; (3) elimination, which discards meaningless stop words such as article, preposition, and conjunction; and (4) stemming; which finds a stem form of a given inflected word (Jeong, 2006; Shvaiko & Euzenat, 2005). Accordingly, the final interface representation is a normalized string, i.e., a sequence of basic formed words. Take a simple example as shown in Fig. 2. The root node QuantityOnHand, for example, is serialized into {QuantityOnHand, Item, SiteId, Identifier, ContactUrl, AvailableQuantity, . . . }, and further normalized into {Quantity, Hand, Item, Site, . . . }. QuantityOnHand, for example, is separated into {Quantity, On, Hand} and then On is eliminated. In the same way, the node AvailableQuantity is represented as {Quantity, Hand, Available, Quantity, metric}.

3.3.2. Similarity measurement

Now, we are ready to compute structural similarity only by applying kernels to the transformed interface representations (i.e., normalized strings). One intuitive and traditional way is to construct a VSM/LSA matrix from the strings and then to compute the cosine similarity. However, this computation ignores the effect of the word order in the normalized strings as well as is biased to the selection of index words. Instead, we address a kernel-based approach (i.e., a word-sequence kernel Cancedda et al., 2003) that reads two strings as sequences of words, generates two feature vectors, and then calculates their inner product ⟨·, ·⟩, i.e., a kernel function in Eq. (1). The final inner product is the similarity between two XML documents (Vert et al., 2004). The new similarity measure incorporates the order of words as well as the frequency of each word.

With Eq. (2), an extension occurs at assignment of different weights (i.e., decay factor, λ) to different nodes. The use of different weights (to different words) is new in tree kernels to our knowledge although it is not new in string kernels (Cancedda et al., 2003; Saunders et al., 2004).

(A) XML Schema document

```xml
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="QuantityOnHand" type="QuantityOnHandType"/>
  <xsd:complexType name="QuantityOnHandType">
    <xsd:sequence>
      <xsd:element ref="Item"/>
      <xsd:element ref="SiteId" minOccurs="0"/>
      <xsd:element ref="AvailableQuantity"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

(B) Abstract DOM Tree

![Fig. 2. An XML document and its DOM tree.](image)

1 The leaf (or lower) nodes represent the atomic data that an XML document ultimately describes (Madhavan, Bernstein, & Rahm, 2001).
Various factors can affect the determination of a decay factor $\lambda_n$ for a node $n$ such as the depth of the node, the number of its child nodes, its context (i.e., ancestor nodes), and the lexical semantics of its label (e.g., synonym) (Zhang, Li, Cao, & Zhu, 2003). Among them, the context information is already captured in the interface representation and the lexical semantics is treated as soft matching in Cancedda et al. (2003) and Saunders et al. (2002). Here, we adopt a depth-dependent decay factor $\lambda_n = \frac{k_n}{\text{depth}(n)}$, where depth($n$) is the depth of the node $n$, (depth(root) = 1), and $r \geq 1$ is a relevant factor. Since, as shown in the example below, the size of inputs (i.e., length of strings) is usually not a constant, the kernel value is sometimes normalized in $[0, 1]$ by applying $\tilde{K}(s, t) = \frac{K(s, t)}{\sqrt{K(s, s)K(t, t) \cdot K^*(s, t)}}$

$\tilde{K}(s, t) = 1$ if and only if strings $s$ and $t$ are identical.

Fig. 3 illustrates an example to compute the structural similarity between the node QuantityOnHand in Fig. 2 and a similar node Inventory, which is represented as {Inventory, Item, Site, Name, Identifier, Contact, Point, Inventory, Quantity, Inventory, Capacity}. For simplicity, we use following symbols to each word in the normalized documents: A(vailable), C(ontact), H(and), I(dentifier), L(ocator), M(inimum), N(ame), P(oint), Q(uantity), R(esource), S(ite), T(Item), U(niversal), V(Inventory), X(Maximum), and Y(Capacity). After serialization and normalization, we get the string ‘QHTSIICUAQMQXQ’ for the Inventory node. The common subsequences are \{C, T, \}^{1}, Q^{4}, S, T, CQ^{3}, IC^{2}, \ldots, TSICQ^{6}\}. Note that numbers in parentheses indicate the number of possible occurrences. Accordingly, as detailed in Fig. 3, their similarity is easily computed via Eq. (2) as $K'' \simeq 2.1399$ and $K'' \simeq 0.6149$ with respect to $r = 1$ and $2$, and $\lambda_0 = 1$, whereas computed via Eq. (1) with with $\lambda = 0.5$ yields $K = 2.3699$.

4. Experimental analysis

To evaluate the proposed method, we performed an experiment with XML schema documents from OAGIS.\(^2\) The OAGIS BOD (Business Object Document) schemas are open and standard specifications for supporting interoperable data exchange by providing enterprise/domain-neutral data structures and definitions. The particular use of OAGIS BODs is very relevant to this research because they are used as the standard data exchange definitions among B2B web services. We designed three types of experiments. The first design is just to show that the proposed method is well suited to human judgment, while the second one is to verify the proposed similarity measures well discriminate valuable information from irrelevant information in the perspective of information retrieval. And, the third experiment shows the proposed kernel-based similarity measure is well suited to an application of XML documents clustering.

For the first experiment, we randomly select 200 pairs of CC’s (Core Components)\(^3\) and then four human experts verify every pair to assign their degree of relatedness. The averaged scores over the four operators are used to compare the performance of the proposed methods. We implemented five algorithms – TED (Tree Edit Distance); VSMs by means of cosine similarity, and LSA; kernels both with fixed penalty (i.e., $\lambda_n = c$) and with variant penalty (i.e., $\lambda_n = f(\lambda_0, \text{depth}(n), r)$). The experiment result is depicted in Fig. 4 in terms of correlation with the operators’ score. ‘KN.1’ and ‘KN.2’ implement the fixed $\lambda$-weighting with respect to $\lambda = 1$ and 1/2, while ‘KN.3’ and ‘KN.4’ implement the variant $\lambda$-weighting with respect to relevant

\(^2\) http://www.openapplications.org.

\(^3\) Highly reusable primitive BOD definitions in OAGIS 9.0.
factors $r = 1$ and 2. As shown in the figure, the kernel methods (including VSMs) outperform TED, the state-of-the-art structural similarity measure. Moreover, the kernel methods that preserve the word order give better performance than VSMs do.

The second experiment is a mapping test that evaluates whether mappings established by an algorithm is correct or not compared with true mappings. We configured four experiment sets, each of which consists of two data sets having 10 CC’s and 20 CC’s. Between the two data sets, an algorithm (and human operators as well for evaluation) select plausible mappings (the number of mapping is less than 10). Then, we evaluate how many mappings by an algorithm are the same ones in true mappings by human operators. For this experiment, we adopt standard performance metrics, i.e., Precision, Recall, and F-Measure, widely used in the information retrieval field. Let $A$ be a set of alignments mapped by an algorithm, and $T$ be a set of true mappings by human experts. Then the metrics are defined as follows:

\[
\text{Precision} = \frac{|A \cap T|}{|A|} \\
\text{Recall} = \frac{|A \cap T|}{|T|} \\
F\text{-Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

The experiment results are depicted in Fig. 5 and show that our proposed kernel method for structured data (i.e., KN) give better performance than the traditional TED and VSMs do as does for the first experiment.

The third experiment is XML document clustering. For this experiment, we prepared 118 CC’s roughly belonging to six groups – time and date, contact information, reference, business role, location, and order reference. We used $k$-medoids clustering because we have a pair-wise similarity table only. We evaluate if the clustering algorithm identifies six clusters well and if samples in each cluster are from the same group. The clustering results are summarized in Table 1, from which we can conclude that the proposed kernel method more precisely and accurately groups samples.

### 5. Conclusion

The interoperable data exchanges among services to be composed are a prerequisite to successful web services composition. A couple of services to be connected must meet their exchange data definitions at a design time and have to be capable of mapping data at a run-time. To this end, we proposed an information mapping analysis that establishes semantic mappings based on an approximate analysis. As part of the analysis, we proposed a novel kernel-based measure to compute structural similarity between tree-structured XML documents. In particular, after a comprehensive review of kernel methods, we presented an interface representation of XML documents to be fed into the kernel computation and also introduced a depth-oriented penalty assessment. The results on experimental simulations show that the proposed kernel-based measurement of structural similarity outperforms the conventional VSMs as well as the tree edit distance method. Studies are still needed in the direction of consolidating the information mapping analysis such as semantics-based reasoning.

### Table 1

<table>
<thead>
<tr>
<th>Method</th>
<th>TED</th>
<th>VSM</th>
<th>LSA</th>
<th>KN.1</th>
<th>KN.2</th>
<th>KN.3</th>
<th>KN.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoCa $a$</td>
<td>4</td>
<td>4.8</td>
<td>4.3</td>
<td>5.1</td>
<td>5</td>
<td>5.3</td>
<td>5.2</td>
</tr>
<tr>
<td>NoD$^b$ (%)</td>
<td>23.3 (19.7)</td>
<td>15.2 (12.9)</td>
<td>24 (20.3)</td>
<td>8.7 (7.4)</td>
<td>7.8 (6.7)</td>
<td>11.8 (10.0)</td>
<td>6.9 (5.85)</td>
</tr>
</tbody>
</table>

$^a$ NoCa indicates the average number of clusters.

$^b$ NoD indicates the total number (%) of different samples within each cluster.
Acknowledgement

This work was supported by the Korea Research Foundation under the Grant Number KRF-2006-311-D00956.

References


