An integrated heterogeneous web service retrieval via combination of instance- and metadata-based schema matching method

Jie HE* and Wei WANGb

aSchool of Resource and Environment, Ningxia University, Yinchuan 750021, China; bState Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China

(Received 27 December 2014; final version received 29 April 2015)

Schema matching is a critical step in the integration of heterogeneous web service, which contains various types of web services and multi-version services of the same type. Mapping loss or mismatch usually occurs due to schema differences in structure and content and the variety in concept definition and organization. Current instance schema matching methods are not mature enough for heterogeneous web service because they cannot deal with the instance data in web service domain and capture all the semantics, especially metadata semantics. The metadata-based and the instance-based matching methods, in the case of being employed individually, are not efficient to determine the concept relationships, which are crucial for finding high-quality matches between schema attributes. In this paper, we propose an improved schema matching method, based on the combination of instance and metadata (CIM) matcher. The main method of our approach is to utilize schema structure, element labels, and the corresponding instance data information. The matching process is divided into two phases. In the first phase, the metadata-based matchers are used to compute the element label similarity of multi-version open geospatial consortium web service schema, and the generated matching results are raw mappings, which will be reused in the next instance matching phase. In the second phase, the designed instance matching algorithms are employed to the instance data of the raw mappings and fine mappings are generated. Finally, the raw mappings and the fine mappings are combined, and the final mappings are obtained. Our experiments are executed on different versions of web coverage service and web feature service instance data deployed in Geoserver. The results indicate that, the CIM method can obtain more accurate matching results and is flexible enough to handle the web service instance data.

Keywords: schema matching; web coverage service (WCS); metadata-based matcher; instance-based matcher; web feature service (WFS)

1. Introduction

Heterogeneous web service includes different types of services, such as web feature service (WFS) (1), web coverage service (WCS) (2), sensor observation service (SOS) (3), and different versions of the same service, e.g. WFS 1.1.0 and WFS 1.0.0. Heterogeneous web service based on various information models (4) causes differences in schema, which can be classified into three categories: (1) Structure difference. According to structure granularity, structure difference can be divided into class level and attribute level. Class level includes creation and deletion of classes, the class inheritance, and so on. Attribute level mainly refers to the modification of class name, attribute, reference, and constraint, e.g. attribute pulled up to super-type or pushed down to subtype. (2) Content difference. Structure difference will bring tremendous changes in content, for example, the modification of class or attribute may cause the variation of property. (3) Instance value difference. Instance data in spatial information domain are different from those in other domains due to their inherent characteristics. Instance value difference includes difference in temporal and spatial, such as the WCS instance data covering the same spatial range but different temporal range, and difference in spatial reference.

When metadata-based matching is executed alone, mismatching will occur due to different schemas of heterogeneous web service. Chen et al. proposed a fragment-based dynamic syntax schema matching method (extended FRAG-BASE schema matching method) and a node semantic similarity (NSS) schema matching method which were applicable to different versions of the open geospatial consortium (OGC) Web Service schemas (5,6). The result of the NSS method shows that the average recall, precision, and overall of WCS schema matching are 83, 92, and 67%, respectively (6). Therefore, it is difficult to determine the right mappings if we only rely on the instance-based matching method. For example, if a number of different element attributes have the same value, but the element relationships in semantics are unknown, it may be difficult to determine the relationship between two elements.

Schema structure and semantic relationship between element labels must be considered in schema matching. The correspondences of instance pairs are also taken into account. This paper proposes a hybrid schema matching method namely the combination of instance and
metadata-based (CIM) matching methods, which combines the metadata-based matcher (such as Combination of Schema Matching Approach, COMA) with the instance-based matcher (such as iMatch). First, CIM resorts to the metadata-based matcher to perform matching on schema context (schema structure and element label). Then, the designed instance-based matcher is employed to compute the similarity of the element instances. Finally, candidate mappings can be determined by combining the results of metadata-based matcher and instance-based matcher.

There are four main modifications in this study. First, the generic schema representation is extended so that each element contains instance data and instance data patterns. Second, three similarity measure functions are designed to compute the similarity of instance pairs of any type. Third, both quality and efficiency of schema matching are improved. On the one hand, the internal schema representation contains instance and schema fragments. The schema matching is based on the similar fragment pairs which are found from the source and target schemas. By utilizing the metadata-based matcher in advance, the instance-based matcher can find the candidate instance pairs quickly and correctly. On the other hand, when applying the instance-based matcher for further matching, more mappings which cannot be found by metadata-based matcher are generated. Fourth, the metadata-based matcher and the instance-based matcher are combined in a flexible way. For a given matching task, the matching system can choose the suitable matcher or combination of matchers automatically. New matchers can be introduced to the system to enhance capability.

The rest of this paper is organized as follows. Section 2 is an overview of schema matching. Section 3 outlines the architecture of the proposed CIM schema matching system (Section 3.1.) and presents the details of the metadata-based matching (Section 3.2.) and instance-based matching (Section 3.3.). In Section 4, experiments on the COMA, iMatch, and CIM methods are implemented for WCS and WFS. Finally, we summarize and discuss some future work in Section 5.

2. Overview of schema matching

Schema matching is a critical step for the integration of heterogeneous web service (7–11). Currently, regardless of syntax schema matching method or semantic (ontology) matching method, in terms of the types of available data used in matching phase, it can be roughly classified into three types: metadata-based, instance-based, and the combination of both. Metadata-based matching is typically based on label name, description or definition, and schema structure (8–10, 12, 13). The aim of instance-based matching is to detect instance referred to the same real-world object. Instance matching is also crucial in applications such as identity recognition and data integration. However, the existing research mainly focuses on metadata-based matching, and instance-based matching is studied only in domains such as database field. According to the types of available evidence used in matching process, the instance-based matching can be mainly classified into three categories: (14).

1. Value matching. It mainly focuses on equivalence between property values of instances. Berlin and Motro proposed an instance-based matching method based on the attribute dictionaries created by domain experts (15), in which, the instance is matched against these dictionaries to generate an individual match score for each attribute. These scores are combined and a mapping is determined by applying a minimum cost and maximum flow graph algorithm. However, the method has potential performance problem because huge efforts are needed to build the dictionaries and especially to calculate the probabilities. In addition, the definition and organization of heterogeneous web service instance data vary with the domains, and the work for experts to build attribute dictionaries is still challenging. Wang et al. proposed an instance-based schema matching method to solve the data matching problem in the network database site and inter-site (16). Unfortunately, its performance is inflected by the selection of instance samples, and the instance data used in matching are different from the web service instance data used in this paper. Bilke and Naumann designed an algorithm to identify properties with different semantic but similar syntax (17). However, the matching result depends on the number of copies in the instance data. If there is no copy, the algorithm will fail.

2. Individual matching. It is mainly executed to detect whether two individuals represent the same real-world object. Chua et al. proposed an instance-based attribute identification method in database integration (18). The mappings are generated by performing statistical analysis of the data in duplicates. The duplicates are assumed to be identified by a common ID attribute, and meanwhile, the author must assume at least one attribute has already been aligned. However, only a few entity identifiers can be consistently employed to all databases and the supported instance data in Ref. (16) is numerical attribute data. Wang and Chen proposed a method to realize the automatic identification of share entities, but the GIS database instance is different from the web service data instance in structure and organization (11).

3. Data-set matching. It takes into account all individuals in the source and target data sets and outputs an optimal alignment between these whole sets of individuals. For example,
Doan et al. proposed an ontology matching method based on machine learning (19). The instance classification is applied to calculate the joint probability distribution of the concept, and then the probability distribution is transformed into conceptual similarity through a similar function. In this method, the instance is required to be the same in the syntax; otherwise it results in poor quality. Additionally, the mapping relationship between concepts is hard to determine in the case of lacking of instance. Based on COMA++ (12), Engmann and Massmann proposed two instance-based matchers as an extension of the existing matcher library (20). In their study, a propagation algorithm is applied to the results of the instance-based matchers to transform similarities from elements to their surrounding elements. Experimental results show that the instance-based matchers further improve the match quality. Unfortunately, the details of the similarity measure used in COMA++ are not shown.

Literature surveys show that the current studies on instance schema matching are not mature enough for heterogeneous web service matching: neither can they deal with the instance data in web service domain, nor can they capture all the semantics, especially the metadata semantics. In this paper, we try to fill this gap by implementing a hybrid schema matching approach, based on the combination of metadata-based matcher and instance-based (CIM) matcher. The matching process is divided into two phases: the metadata matching phase and instance matching phase. In the first matching phase, metadata-based matchers are used to compute the element label similarity of multi-version OGC web service schema, and the generated matching results are raw mappings, which will be reused in the next instance matching phase. In the second matching phase, the designed instance matching algorithms are employed to the instance data of the raw mappings and fine mappings are generated. Finally, the raw mappings and the fine mappings are combined and the final mappings are obtained.

3. Proposed method: CIM

3.1. System architecture

This section describes the overall architecture of the CIM matching system. As shown in Figure 1, the system comprises four core components: schema parsing, schema partition, matcher execution and similarity combination. The details of each component and the specific modification in the context of this architecture are introduced below.

(1) Schema parsing enables CIM method to manage the input data source. The format of the data source schema is converted into an internal representation, such as a tree structure. Then, the instance value of the corresponding element can be extracted from the schema tree.

(2) Schema partition is a two-step process designed to divide the schema (schema tree) into fragments (sub-trees), according to the characteristics of schema structure. The first step is schema tree partition. The input is schema trees and the partition is performed for all the root nodes (each schema tree is replaced by a proxy node, such as a root node). The root nodes represent schema trees, which can be divided into sub-trees. The second step is similar fragment (sub-trees) identification. The goal of this step is to identify the most similar fragments of the source and target schemas. Therefore, these fragments are worth matching in the further processes. The comparisons of fragments are performed on fragment metadata, such as the root node names and fragment contexts.

(3) Matcher execution performs the matching works according to the matcher selected from the matcher library (including metadata-based matcher and instance-based matcher), and outputs the corresponding similarity matrix.

(4) Similarity combination combines the similar value in similarity matrix to generate a total similarity of the corresponding elements.

Finally, according to the predefined threshold and the reference mappings in information management library, the mapping selector generates candidate mappings, which are checked by mapping validator and outputs the final matching results. If the combination of similarity values cannot meet the users’ requirements, the program returns to the matcher execution phase to perform a new iterative process. At the same time, the matching system will automatically choose the suitable matcher or a combination of matchers for a given task, and the information of optimal combination of matchers will be saved to the library. Additionally, in the matcher execution phase and similarity combination phase, the user is usually involved in selecting matchers, determining the combination strategy, running the parameters of matchers, and so on.
3.2. Metadata-based matching

The details of schema matching have been described in many a literature. For example, COMA (21) is composed of four modules: parsing, pre-process, matcher execution and a combination of match results. First, the parsing module aims at parsing the input files and converting them to the internal graph format. Second, the pre-processing module is concerned with pre-processing operation which prepares the data for the matching process. For example, it can determine which element pairs will be compared. Then, the matcher execution module executes multiple independent matchers chosen from the matcher library and each matcher determines an intermediate match result consisting of a similarity value of [0, 1]. Finally, the combination operations are performed on the results of individual matchers to derive the combined match result and apply a selection strategy to choose the match candidates for a schema element, such as selecting elements of the other schema with the best similarity value exceeding a predefined threshold. In this paper, we mainly introduce the instance-based matcher – iMatch, which is one component of the CIM.

3.3. Instance-based matching

3.3.1. Instance similarity definition

Geospatial web service instance data are divided into three types: string, numeric, and the mixture of both.

(1) String-type instance includes the single-string type and the multi-string type. For the single-string value, the examples are the name of the element (Name = “Montain”), identification value (ID = “001”), etc. For the multi-string value, the example is the description of the element attributes (description = “NOAA 15 Advanced Microwave Sounding Unit-A Footprint Data”).

For the single-string value, such as name = “Coverage” and title = “Coverage”, the similarity between these two strings can be defined in edit distance. Comparing with other algorithms, edit distance is easier and more effective, especially for short strings. Before computing the similarity, we first classify the string label and analyze morphology to identify the various possible forms, for example, (Coverage, Coverages) → coverage, then we have: \( \text{sim}_{\text{single-str}}(\text{name}, \text{title}) = \text{sim}_{\text{edit}}(\text{“coverage”}, \text{“coverage”}) = 1 \), where \( \text{sim}_{\text{single-str}}(\text{name}, \text{title}) \) represents the similarity between string name and title, \( \text{sim}_{\text{edit}}(\text{“coverage”}, \text{“coverage”}) \) denotes the Levenshtein distance between the two strings.

For the multi-string value, the instance similarity calculation is defined as follows.

**Definition 1:** The similarity of string-type instance. Suppose that we have two string sets: \( A = (a_1, a_2, a_3, ..., a_n) \) and \( B = (b_1, b_2, b_3, ..., b_m) \), where \( a_1, a_2, a_3, ..., a_n, b_1, b_2, b_3, ..., b_m \) are the sub-strings (the original string is divided into sub-strings according to the stop words, such as a space, semicolon, etc.) of \( A \) and \( B \). Then, the inner product of string \( A \) and \( B \) is represented as:

\[
(A, B) = \sum_{i=1}^{n} \sum_{j=1}^{m} \text{sim}_{\text{edit}}(a_i, b_j)
\]

where \( (A, B) \) represents the inner product of string \( A \) and \( B \); \( \text{sim}_{\text{edit}}(a_i, b_j) \) is used to calculate the similarity between \( a_i \) and \( b_j \). The inner product of string \( A \) and \( B \) is the sum of the dot product of all the sub-strings similarities. The normal number of the strings and the definition of similarity between strings can be derived according to the inner product of the string sets.

**Definition 2:** The normal number and similarity. Supposing that we have a string \( A \), the norm is defined as:

\[
||A|| = \sqrt{(A, A)}
\]

where \( ||A|| \) represents the norm of \( A \), which is the square root of inner product of \( A \).

The similarity between the two string sets is defined as:

\[
\text{sim}_{\text{str}}(A, B) = \frac{(A, B)}{||A|| \cdot ||B||} = \frac{(A, B)}{\sqrt{(A, A)} \cdot (B, B)}
\]

where \( \text{sim}_{\text{str}}(A, B) \) represents the similarity between \( A \) and \( B \). It is the quotient of inner product and norm of \( A \) and \( B \), and also represents the cosine similarity measure between \( A \) and \( B \). The cosine similarity measure (22) is the most widely used one between two document vectors (features or the similarity of features), and the complexity can be even transformed to linear, which makes it perfectly applicable to real-world tasks.

(2) Numerical-type instance also includes single-value and multi-value type. Single-value means have only one instance value, such as the resolution of the observational data (Resolution = “0.0001”). Multi-value is the combination of single-values, such as the range of observation data (lower corner = “−90.0, −180.0”, upper corner = “90.0, 180.0”).

For the single-value instance, the similarity is computed directly by a numerical comparison function, as shown in Equation (4). The similarity is 1 when the two values are equal, otherwise it is 0.

\[
\text{sim}_{\text{val}}(a, b) = \text{IsEqual}(a, b) = \begin{cases} 
1 & a = b \\
0 & a \neq b 
\end{cases}
\]

where \( \text{sim}_{\text{val}}(a, b) \) represents two single-value instance similarity, \( \text{IsEqual}(a, b) \) function is used to judge whether their values are equal. Different attributes (such as latitude, longitude, altitude, etc.) may have the same value. The compared two data must be the same attribute data, and have the same coordinate reference, scale, and units. Therefore, the data should meet the above
constraints before the similarity calculation. Otherwise, the suitable conversion is needed, such as the coordinate system conversion. For example, there are two single-value data with different units: \( \text{timeInterval} = "1" \), in which the unit is “hour”; \( \text{intervalTime} = "60" \), in which the unit is “minute”. Then, transformation between the two units must be taken before similarity calculation by applying Equation (4). Data with different attributes cannot be calculated using the above formula. For example, \( \text{coverages} = "6" \), \( \text{dimision} = "6" \), there is no sense to compare them if the data attributes are different.

For the comparison of multi-value-type instance, we first determine the properties and location in the value collection of each single value, and then determine the corresponding value to be compared. In order to facilitate the calculation, we ensure that the indexes of the two compared values are equal. The similarity of the two multi-value-type instances is defined as follows.

**Definition 3:** The similarity of two multi-value-type instances. Suppose we have two value collections \( U = \{p_1, p_2, p_3, \ldots, p_m\} \), \( V = \{v_1, v_2, v_3, \ldots, v_n\} \), where \( p_1, p_2, p_3, \ldots, p_m \) and \( v_1, v_2, v_3, \ldots, v_n \) are sub-values (the original value string is divided into sub-string according to the stop words, such as space, semicolon, etc., and sub-string is transformed into sub-value) of \( U \) and \( V \). Then, the similarity of value collections \( U \) and \( V \) is represented as

\[
\text{sim}_{\text{dig}}(U, V) = \frac{1}{n} \sum_{i=1}^{n} \text{sim}_{\text{val}}(x_i, y_i)
\]

where \( \text{sim}_{\text{dig}}(U, V) \) represents the similarity of value collections \( U \) and \( V \), which is the arithmetic mean of the sum of all the sub-value similarities. Additionally, all the similarity calculation must be under the same condition (with the same attribute, the same coordinate system, the same scale and units, etc.). If not, some transformations should be done before the calculation. For example, there are two instances: \( \text{envelope} = (-90, -180, 90, 180) \), with the coordinate system of EPSG: 4329, and \( \text{BoundingBox} = (44.67, -128.67, 30.35, -110.26) \), with the coordinate system of EPSG: 4326. The transformation of these two coordinate systems must be conducted before the similarity calculation.

(3) **Mixture-type instance** means value comprising string value and numerical value, such as the weather observation data, containing both the string-type instances, the observation time value, and the numerical-type instance, the observation data consisting of time, temperature, pressure, windSpeed, and windDirection. The data type of time is date, while the other ones are the value types.

Figure 2 shows the mixture-type instance of weather observation. In this case, the calculation of the instance similarity can be executed. First, each sub-instance should be determined according to the patterns of the instance data text block (<swe:TextBlock>). For example, according to the block separator (<swe:TextBlock blockSeparator = “;”/>), we can split the observation values text block (<swe:values/>) into sub-block, and each sub-block is a sub-instance. Then, each sub-instance structure can be obtained according to the fields defined in data record (<swe:DataRecord>). For example, based on the elements DataRecord (<swe:DataRecord> +<swe:field name = “time”/> +<swe:field name = “temperature”> +<swe:field name = “pressure”> +<swe:field name = “windSpeed”> +<swe:field name = “windDirection”></swe:DataRecord>) in Figure 2, we know that each sub-instance data fields of weather observation is composed of time, temperature, pressure, windSpeed, and windDirection. After that, we retrieve each data field value by the token separator (<swe:TextBlock tokenSeparator = “;”/>). For each sub-instance, we determine its instance type and instance value, apply Equations (3) and (5) to calculate the string-type and the numerical-type instance similarities, and then calculate their average value as the final similarity of the mixture type. The formula is defined as follows:

**Definition 4:** The similarity of mixture-type instances. Suppose that we have two mixture values \( H = (H_{str}, H_{dig}) \) and \( F = (F_{str}, F_{dig}) \), where \( H_{str} \), \( F_{str} \), \( H_{dig} \), and \( F_{dig} \) are the string-type instance of \( H \) and \( F \), and the numerical-type instance of \( H \) and \( F \), respectively. Then, the similarity of mixture instance \( H \) and \( F \) is represented as:

\[
\text{sim}_{\text{sub}}(H, F) = \frac{1}{2} \left( \text{sim}_{\text{str}}(H_{str}, F_{str}) + \text{sim}_{\text{dig}}(H_{dig}, F_{dig}) \right)
\]

where \( \text{sim}_{\text{str}}(H, F) \) represents the similarity of the mixture instance \( H \) and \( F \). It is the arithmetic mean of the string-type instance similarity \( \text{sim}_{\text{str}}(H_{str}, F_{str}) \) and the numerical-type instance similarity \( \text{sim}_{\text{dig}}(H_{dig}, F_{dig}) \).

For some special instance values, such as sensor observation service instance value, we usually use date or time as keywords. Therefore, if we use date or time as keywords to carry out web service request or query, the instances to be compared must have the same time.
or date, or within a specified date or time range; otherwise there is no need to take compare.

3.3.2. Matching implementation

Figure 3 is the matching flowchart of iMatch. The matching process includes instance extraction, pre-match, and matching. The main task of instance extraction is to retrieve the instance data of the leaf nodes of the schema trees. Engmann and Massmann extended the schema representation to make sure that each element can contain instance data (20). It is known that, for a given web service schema representation, not all the elements but only the leaf elements contain instance data. Therefore, based on COMA++, element name and ID are introduced to represent the elements without instance data. Each instance element is represented by a five-tuple: <id, name, type, instance, patterns>. First, id represents the position of the element in the schema tree. For example, id = "7" refers to the seventh element node of the sub-tree. Traversing the schema graph in depth-first order, the element number can be calculated: id of the root node ID is “1” and so forth. Second, name is the identifier of the element. Third, type refers to the instance data type, such as string, integer, double, etc. Fourth, instance refers to the element instance. Fifth, patterns is used to describe the structure of element instance by regular expressions. In order to identify the interested strings in certain texts more easily, different patterns are used to describe instance data. For example, email has the pattern *@*.* and website has the pattern http://
/*.; date is expressed as [d–]{4}–[d–]{2}–[d–]{2}; and time as [d–]{2}:[d–]{2}:[d–]{2}, where the symbol "\[" denotes the character or number, “\]” denotes the digit and “\{" denotes the character or digit. Patterns C.* and C\[w–\]{n} match the strings starting with “C”, following any character or number and with the total length of n. [w–]{1, 5} denotes that the expression in the squared brackets has to occur 1 time minimum and 5 times maximum.

Figure 4 shows the import of WCS XML instance source into internal schema representation. The left part of Figure 4 is some input WCS XML fragments and the right part is the corresponding representation in the CIM internal schema graph. For each element in XML fragment, there is a corresponding instance element represented by a five-tuple. For example, the first element in Figure 4 is wcs:CoverageSummary. ows:Title, and its corresponding instance element is represented by id:2 name:Title type:string Instance:{A sample ArcGrid file, North America sample imagery} patterns:[w–]{n}.

In the internal schema graph of Figure 4, there are not only the element names and instance elements, but also the element IDs and instance data patterns. For the duplicate instance data, such as "<ows:keyword=WCS</ows:keyword>", only one of them can be saved.

The imported internal schema graph is managed by the schema manager, and the instance matching is based on the instance data. Different elements may have the same instance value. For example, "<ows:Constraint name = "LocalTraverseXLinkScope"/><ows:value>2<br/</ows:value>
</ows:Constraint>"; "<gml:MultiSurface><srsDimension>2</srsDimension><gml:MultiSurface>"”, these elements are with the equal instance value of 2, but there is no corresponding relationship between the two elements. Therefore, a pre-match on schema instance is needed before the matcher execution in order to reduce false matching and improve matching efficiency. The main work of the pre-match is to find all the candidate instance pairs by the calculation of the instance data similarity, which is a combination of element semantic similarity and instance pattern similarity. If the combined similarity value is greater than the predefined threshold (set to 0.6), the two instance elements can be considered as a candidate instance pair. In this paper, the method proposed in Ref. (5) is employed to calculate the element semantic similarity, and the edit distance algorithm is applied to calculate the instance patterns similarity. Finally, the two similarities are combined and final similarity is generated. The calculation of the combined similarity is defined as follows:

Definition 5: The similarity of candidate instance pairs. Suppose that we have two elements: E_i = (id_i, name_i, type_i, instance_i, patterns_i), E_j = (id_j, name_j, type_j, instance_j, patterns_j). The combined similarity of E_i and E_j is represented as

![Figure 3. The flowchart of the instance matching.](image-url)
\[
\sim(E_s, E_t) = \alpha \times \sim_{\text{meta}}(\text{name}_1, \text{name}_2) + (1 - \alpha) \\
\times \sim_{\text{edit}}(\text{patterns}_1, \text{patterns}_2)
\]  

(7)

where \(\sim_{\text{meta}}(\text{name}_1, \text{name}_2)\) is used to compute the element metadata similarity and \(\sim_{\text{edit}}(\text{patterns}_1, \text{patterns}_2)\) is used to compute instance pattern similarity. The two candidate instance data must be of the same type before the computation of similarity. For different instance data type, the value is set as follows:

- \(\text{type} = 0\): When the instance data type is mixture type.
- \(\text{type} = 1\): When the instance data type is string type.
- \(\text{type} = 2\): When the instance data type is numerical type.

Because the same instance value may refer to the different elements in Equation (7), the weight of element metadata similarity is greater than that of instance pattern similarity, and the value of factor \(\alpha\) is set to 0.7. For example, there are two instance elements of different versions of service of WFS, \(E_s = ("4", "fid", "states.3", "s[\{w-\}8\}, "1")\), \(E_t = ("4", "id", "states.3", "s[\{w-\}8\}, "1\)\). According to Equation (7), we have:

\[
\begin{align*}
\text{Sim}(E_s, E_t) &= 0.7 \times \sim_{\text{meta}}("fid", "id") + (1 - 0.7) \\
&\times \sim_{\text{edit}}(s[\{w-\}8\}, s[\{w-\}8\])
\end{align*}
\]

\[
= 0.7 \times 0.67 + 1.0 \times 0.3 = 0.79
\]

The similarity of \(E_s\) and \(E_t\) exceeds the threshold; therefore, \(E_s\) and \(E_t\) are the candidate instance pairs.

If all candidate instance pair-wises are determined, the instance matching can be performed on these instance pairs. As shown in Figure 3, in the instance matching phase, three different similarity measure functions are designed for different types of instance pairs, respectively, such as string matcher, numerical matcher, and hybrid matcher.

The first algorithm is the instance pairs matching algorithm. The input is candidate instance pairs, and the output is instance mappings. Each output instance mapping is represented as a triple: \(\text{Mapping} = (\text{name}_1, \text{name}_2, \text{sim}_{\text{instance}})\), where \(\text{name}_1\) and \(\text{name}_2\) are the names of instance elements and \(\text{sim}_{\text{instance}}\) is the similarity of the two instance elements with the value of \([0, 1]\). The details of instance pairs matching algorithm are as follows:

**Algorithm 1: Instance pairs matching algorithm.**

**Input:** Candidate instance pairs \((E_s, E_t)\).

**Output:** Instance mappings.

**Instance Matching \((E_s, E_t)\).**

1. Choose an instance pair \((e_s, e_t)\) from the model manager.
2. Determine the instance data type. For string type, Equation (3) is used to calculate the instance string similarity. For numerical type, Equation (5) is applied to compute the instance data similarity. For hybrid type, each instance data structure must be determined first according to the instance data patterns and the fields of data type. Then the hybrid instance data are split into string-type instance data and numerical-type instance data. Finally, Equation (6) is used to calculate hybrid instance data similarity.
3. Process the intermediate matching results. The intermediate matching results can be saved to information management library, and at the same time, the previous intermediate matching results need to be updated in terms of the added
matching results. For example, multiple different similarities of the same instance pair can be combined into an overall similarity.

(4) If the instance pair matching is finished, go to step (5). Otherwise, return to step (1) and repeat the above steps.

(5) Output the instance pairs, matching results, and end the algorithm.

3.4. Matching combination

The whole schema matching is performed at two levels: metadata matching and instance matching. Metadata matching is independent of instance matching, while instance matching is performed based on metadata matching. First, a metadata-based matcher included in the CIM is chosen to implement metadata matching and output mappings\(^1\), which is also the input of the second level – instance matching. Second, for each mapping of mappings\(^1\) with the instance similarity of \(sim_{\text{meta}}\), the instance data are retrieved and the similarity measure functions are employed to compute the instance similarity of \(sim_{\text{instance}}\) between two instance data. Finally, an average similarity is defined as \(sim_{\text{tot}} = (sim_{\text{meta}} + sim_{\text{instance}})/2\). If the value of \(sim_{\text{tot}}\) is higher than the predefined threshold, the mapping can be obtained.

4. Experiment and discussion

We evaluate the proposed CIM method by comparing with the existing matching methods. The experiment results on six data sets will be given. All of these experiments and time measurements are uniformly implemented using Sun Java 1.6.0 libraries on a Windows device with an Intel Core 2, 2.5 GHz Quad processor, and 2.0 GB RAM.

4.1. Experiment design

4.1.1. Evaluation metrics

The evaluation metrics consisting of Precision, Recall, Overall, and F-Measure is used to evaluate the matching results (23). Precision (\(P\)) is the percentage of correctly discovered mappings in all discovered mappings. Recall (\(R\)) is the percentage of correctly discovered mappings in all correct mappings. Overall (\(O\)) represents a combined measure for match quality, taking into account the post-match effort which removes false matches and adds missed matches. F-Measure (\(FM\)) represents the harmonic mean of Precision and Recall. The four parameters can be represented as

\[P = \frac{|m_a \cap m_m|}{|m_a|}, \quad R = \frac{|m_a \cap m_m|}{|m_m|}\]

\[O = R \times (2 - 1/p), \quad FM = 2 \times \frac{P \times R}{(P + R)} \quad (8)\]

where \(m_a\) is mapping discovered by the matcher, and \(m_m\) is mapping assigned manually.

4.1.2. Data sets

The experiment data include XSD schema file and XML instance document. The XSD schema file includes version 1.0.0 and version 1.1.0 schema file of both WCS and WFS. There are seven schema files in version 1.0.0 of WCS: describeCoverage.xsd, getCoverage.xsd, gml4wfs.xsd, OGC-exception.xsd, owsBase.xsd, values.xsd, and wcsCapabilities.xsd. There are 17 schema files included in WCS schema document of version 1.1.0 and 3 schema files (wcsGetCapabilities.xsd, wcsDescribeCoverage.xsd, wcsGetCoverage.xsd) have covered all the relevant contents in the 17 schema files. Therefore, all the schema matching experiments for WCS are implemented between wcsCapabilities.xsd and wcsGetCapabilities.xsd, describeCoverage.xsd and wcsDescribeCoverage.xsd, and getCoverage.xsd and wcsGetCoverage.xsd, respectively. There are 5 schema files in the WFS version 1.0.0 schema document: OGC-exception.xsd, wfs.xsd, WFS-basic.xsd, WFS-capabilities.xsd, and WFS-transaction.xsd. Only one schema file (wfs.xsd) is included in the version 1.1.0 schema document. Therefore, the schema matching experiments for WFS are implemented between wfs.xsd and WFS-basic.xsd, wfs.xsd and WFS-capabilities.xsd, and wfs.xsd and WFS-transaction.xsd, respectively.

The XML instance data in the experiment are supplied by the open-source WebGIS realization product – Geoserver (http://geoserver.org/display/GEOS/Welcome). It provides two different versions of web service instances of WFS and WCS. The selected XML files include wfsGetCapabilities1.1.0_response.xml, wfsGetCapabilities1.0.0_response.xml, describeFeatureType1.1.0_response.xml, describeFeatureType1.0.0_response.xml, getFeature1.1.0_response.xml, getFeature1.0.0_response.xml, wcsGetCapabilities1.1.0_response.xml, wcsGetCapabilities1.0.0_response.xml, wcsDescribeCoverage1.1.0_response.xml, wcsDescribeCoverage1.0.0_response.xml, getCoverage1.1.0_response.xml, and getCoverage1.0.0_response.xml.

When the XML files are phased in the experiment, the statistics is set for instance elements. Tables 1 and 2 show the statistics on WFS and WCS service instance elements, respectively. The columns represent XML files of two versions service requests, such as getCapabilities, describeFeatureType, and getFeature in Table 1, and wcsGetCapabilities, describeCoverage, and getCoverage in Table 2. The rows represent statistics on numerical type, string type, mixture type, non-instance elements, and the total elements. Statistical data show that the number of getCapabilities operation are significantly different for the two versions of WFS service instance.
elements, for example, the total numbers of the elements are 492 for version 1.1.0 and 348 for version 1.0.0. Furthermore, the element number of individual types is also different between the two versions of service instance. It means that every version of web service is different from each other, which enhances the difficulty in finding the “correct” mapping. Moreover, there is no instance element in response to XML files of the two versions of describeFeatureType request.

### 4.1.3. Experiment setting

Three matching methods (COMA, iMatch, and CIM) are employed in the matching experiments. COMA is used to find mappings between elements in XSD schema files. iMatch performs an instance matching and finds the correspondent instance pairs. CIM combines the results of the two matchers and outputs the final results. For COMA, the best combination strategies are applied in the test: Average for aggregation, Both for direction, Average for computing combined similarity, and use the combination of threshold and MaxDelta for selection.

In the XML instance file, the element characteristics are determined by both the element name and the element instance, and the same instance data generally describe the similar characteristics. The same value in a schema instance file may appear many times, but the same value appearing in different elements may have different semantics. Additionally, in order to test the robustness of the CIM system for matching numerical-type instance data, we take the response XML file of WFS getCapabilities request as experiment data, and add 30 false candidate mapping elements to the matching system manually (including 15 false numerical-type instance pairs). For the other methods (COMA or iMatch), the match quality and performance are evaluated as well.

### 4.2. Results

#### 4.2.1. Precision

Figure 5 shows that the CIM method yields the best precision for getCapabilities, describeFeatureType, getFeature of WFS, and wcsGetCapabilities, describeCoverage, getCoverage of WCS. The precision for WFS getCapabilities, describeFeatureType, getFeature, wcsGetCapabilities, describeCoverage, and getCoverage are 97, 100, 94, 87, 85, and 80%, respectively. The average precision of CIM is as high as 90.5%, while that of COMA and iMatch are about 79 and 63%, respectively.

#### 4.2.2. Recall

Figure 6 shows that the CIM method yields the best recall for getCapabilities, describeFeatureType, getFeature of WFS, and wcsGetCapabilities, describeCoverage, getCoverage of WCS. The recall for WFS getCapabilities, describeFeatureType, getFeature, wcsGetCapabilities, describeCoverage, and getCoverage are 97, 100, 94, 87, 85, and 81%, respectively. The average recall of CIM is as high as 91%, while that of COMA and iMatch are about 81 and 61%, respectively.

#### 4.2.3. Overall

Figure 7 shows that the CIM method yields the best overall for getCapabilities, describeFeatureType, getFeature of WFS, and wcsGetCapabilities, describeCoverage, getCoverage of WCS. The overall for WFS getCapabilities, describeFeatureType, getFeature, wcsGetCapabilities, describeCoverage, and getCoverage are 95, 100, 93, 72, 73, and 61%, respectively. The average overall of CIM is as high as 82%, while that of COMA and iMatch are about 59 and 39%, respectively.
4.2.4. F-measure

Figure 8 shows that the CIM method yields the best F-Measure for getCapabilities, describeFeatureType, getFeature of WFS, and wcsGetCapabilities, describeCoverage, getCoverage of WCS. The F-Measure for WFS getCapabilities, describeFeatureType, getFeature, wcsGetCapabilities, describeCoverage, and getCoverage are 97, 100, 96, 85, 86, and 80%, respectively. The average F-Measure of CIM is as high as 91%, while that of COMA and iMatch are about 79 and 61%, respectively.

4.2.5. Robustness

Figure 9 shows the robustness of CIM method with 30 false candidate mappings. The matching recall and precision are decreasing rapidly with the increase in false candidate mappings, especially with the increase in numerical candidate instance pairs. When the number of the false candidate mappings increases from 8 to 28, the precision drops from 98 to 65% accordingly.

In summary, Figures 5–8 illustrate that match quality of the CIM is the best of the three matching methods. Figure 9 indicates that the right candidate instance pairs in pre-match phase should be determined when the import of matching is numerical-type instance data. Otherwise, the rate of false matching will increase and the match quality will decrease.

4.2.6. Performance

Figure 10 shows the run time of the three matching methods for a specific matching task. The run time will increase with the data volume of the matching task. When the matching instance data size increases from 0 to 30 KB, the run time varies from 0 to 4 s. The running results show that, the iMatch performs the best, COMA comes the second, while CIM is the worst.

The CIM method has notable advantages over COMA and iMatch for outputting the matching results, but it has the disadvantage in the degraded matching performance (as shown in Figure 10). On the one hand, CIM needs to execute COMA and iMatch for metadata matching and instance matching respectively, and the running time of CIM is much longer than that of COMA or iMatch. On the other hand, when we execute some match tasks by COMA and iMatch in parallel, the total running time of COMA and iMatch is more than that of CIM.

4.3. Discussion

Analyzing the experiment data and the results shown in Figures 5 and 6, we can find that: (1) For WFS getCapabilities instance files, after the processes of automatic pre-processing and manual check, 319 candidate mappings can be found between the two versions instance files, including 248 candidate instance mappings. Additionally, the similarities of all the 248 candidate mappings are 1 after the instance matching. While the COMA method is employed for matching, 305 candidate mappings are found. Because there are more than 100 elements without instance data, the instance matching method is not employed and the recall and precision of iMatch are lower than that of COMA. (2) For describeFeatureType instance files, the recall and precision of iMatch are both 0 because all the elements are without instance data. While the COMA method is employed to all the 26 candidate mappings, the recall and precision achieve 100% as desired. (3) For getFeature instance files, there are 31 candidate mappings (including 25 instance mappings) found by iMatch method and 29 mappings found by the COMA method. The recall and precision of COMA are higher than that of iMatch for lack of instance data. (4) For WCS getCapabilities instance files, there are 86 candidate mappings (including 53 instance mappings) found by the iMatch method and 53 mappings found by the COMA method. The matching recall and precision of iMatch are lower than that of COMA because, more than 90 elements are without instance data. (5) For describeCoverage instance files, there are 34 candidate mappings (including 25 instance mappings) found by the iMatch method and 22 mappings found by the COMA method. The matching recall and precision of iMatch are higher than that of COMA. (6) For getCoverage instance files, there are 9 candidate mappings (including 6 instance mappings) found by the iMatch method and 5 mappings found by the COMA method. The recall and precision of COMA are lower than that of iMatch. We also find that COMA is effective in the non-instance element matching while iMatch performs well in the instance element matching. For example, for candidate mapping posList*=coordinates in getFeature instance file, not COMA but iMatch can determine it. Therefore, the CIM-based matchers can achieve the satisfied results.

4.3.1. Advantages

Compared with the metadata-based matching methods such as COMA and Cupid (24), the CIM method performs not only metadata matching but also instance matching. COMA provides a large spectrum of matchers that can be combined flexibly and supports various ways for the combination of match results. The matchers exploit schema information, such as element and structural property, but the element instance is not included. Therefore, COMA is powerless for the instance data. In contrast to COMA, CIM defines three different similarity measure functions to execute a pair-wise comparison of instance values. With the help of instance matching, the whole match quality is greatly improved. For example, the average overall of CIM is as high as 82%, whereas that of COMA and iMatch are about 59 and 39%, respectively.
Compared with the traditional instance-based matching method, CIM divides the internal schema graph into many suitable fragments in the pre-match phase. Based on these fragments, the metadata match results and instance pattern comparison are used to determine the candidate instance pair-wise. As a result, the matching performance is improved and the rate of false mapping is decreased. Finally, CIM applies different combination strategies to combine the results of metadata matcher and instance matcher for a mapping. The recall and precision of CIM is obviously better than that of COMA and iMatch.
4.3.2. Disadvantages
The experimental results show that CIM is more effective in matching instance data than metadata. If there is no instance data or the instance matcher does not work, CIM will become metadata matcher. For example, when the instance matcher – iMatch is employed to describe FeatureType instance schema files, all the values of match quality are zero. In addition, the robustness of instance matcher is influenced by the instance data type, especially by numerical instance data. Therefore, more efforts should be made to determine the candidate instance pair-wise before the execution of matching.

Compared with metadata matcher and instance matcher, CIM performs schema matching not only on schema element and structural property, but also on element instance. In particular, for large amounts of instance data and complex data structure, much time will be spent on dealing with these instance data and the matching performance will somewhat worsen.

5. Conclusions and future work
Currently, the traditional metadata matcher does not exploit the element instance data, while the instance matcher cannot take full advantage of the results of metadata-based matching. Neither metadata matcher nor instance matcher can solve the problem of mapping loss. In order to further improve the match quality, we propose a combination instance matching method – CIM which combines metadata matcher with instance matcher. With the help of metadata matcher, CIM outputs the right candidate instance pair-wise, and for different types of instance data, such as numerical-type, string-type, and mixture-type instances data, CIM designs three different similarity measure functions for different types of instance pair-wise respectively. By making full use of the advantages of the two matchers, CIM has a better match quality. The results for different WFS and WCS versions of web service schemas show that the average recall, precision, and overall of the CIM matching are 91, 90.5, and 82%, respectively. The proposed method achieves the precise matching for different instances data, identifies those mappings that the metadata matcher loses, and improves the recall and precision. With accurate matching results, the integration of heterogeneous web services will be well achieved.

Instance matching between heterogeneous SOS and other types of web service, such as WFS or WCS, will be the focus of the future research. Meanwhile, due to the difference in data formats, representations and semantics of the observation service instances, especially the difference in the instance types of web service, more serious challenges in matching will be introduced. Therefore, in the future, we will make use of great number of web service instances to test the proposed method, and design new algorithms to deal with those instances with complex structure and big size. Another possibility is to package the matching algorithms into web process service (WPS) and apply the third-party software, such as Hadoop (http://hadoop.apache.org/) to optimize the system’s performance (25).

Funding
This work was supported by the National Natural Science Foundation of China [grant number 41201393]; the Open Research Fund of State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing of Wuhan University [grant number 14I03].

Notes on contributors
Jie HE, PhD, is an associate professor of Ningxia University, China. His research interests include geospatial web service matching and integration, geographical information system software, geospatial data sharing and interoperability, and sensor web.

Wei WANG, PhD, is a professor of the State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing of Wuhan University. His research interests include application of computer technology, the theory and method of GIS, cloud computing, and smart city.

References
(1) Vretanos P.A., Ed. OGC Web Feature Service Implementation Specification (Version 1.0.0). In Open Geospatial Consortium (OGC) Document Number: 02-058, Wayland, MA, USA, 2002; pp 105.
(2) Whiteside, A; Evans, J.D., Ed. OGC Web Coverage Service Implementation Standard (Version 1.1.2) [online] 2008, Open Geospatial Consortium. http://www.opengeospatial.org/standards/wcs (accessed Sep 1, 2012).

(3) Na A.; Priest M., Eds. OpenGIS Sensor Observation Service Implementation Specification (Version 0.1.5). In Open Geospatial Consortium, OGC; Document Number: 06-009r1, Wayland, MA, USA, 2006; 187p.

(4) Wang, H.; Akinci, B.; Garrett, J.H. Formalism for Detecting Version Differences in Data Models. J. Comput. Civil Eng. 2007, 21 (5), 321–330.

(5) Chen, N.; He, J.; Wang, W.; Chen, Z. Extended FRAG-BASE Schema-matching Method for Multi-version Open GIS Web Services Retrieval. Int. J. Geog. Inf. Sci. 2011, 25 (7), 1045–1068.

(6) Chen, N.; He, J.; Wang, C.; Yang, C. A node Semantic Similarity Schema-matching Method for Multi-version Web Coverage Service Retrieval. Int. J. Geog. Inf. Sci. 2012, 26 (6), 1051–1072.

(7) Shvaiko, P.; Euzenat, J. A Survey of Schema-based Matching Approaches. J. Data Semant. IV (Lecture Notes in Computer Science) 2005, 3730, 146–171.

(8) Giunchiglia, F.; Shvaiko, P.; Yatskevich, M. Semantic Matching. In Encyclopedia of Database Systems; Ling, L., Özsu, M.T., Eds. Springer: New York, 2009; pp 2561–2566.

(9) Giunchiglia, F.; Shvaiko, P.; Yatskevich, M. S-match: An Algorithm and an Implementation of Semantic Matching. Semant. Web: Res. Appl. (Lecture Notes in Computer Science) 2005, 3053, 61–75.

(10) Giunchiglia, F.; Yatskevich, M.; Giunchiglia, E. Efficient Semantic Matching. Proceedings of ESWC, Heraklion, Greece, 2005; pp 272–289.

(11) Wang, Y.; Chen, J. An Instance-based Approach for Schema Matching between GIS Databases. Geomatics Inf. Sci. Wuhan Univ. 2008, 33 (1), 46–50.

(12) Aumüller, D.; Do, H.; Maßmann, S.; Rahm, E. Schema and Ontology Matching with COMA++. Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. ACM Press: New York, 2005; pp 906–908.

(13) Noy, N.; Musen, M. The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. Int. J. Human-Comput. Stud. 2003, 59 (6), 983–1024.

(14) Ferrara, A.; Nikolov, A.; Scharffe, F. Data Linking for the Semantic Web. Int. J. Semant. Web Inf. Sys. 2011, 7 (3), 46–76.

(15) Berlin, J.; Motro, A. Database Schema Matching Using Machine Learning with Feature Selection. Advanced Information Systems Engineering, the 14th International Conference, CAiSE, Toronto, Canada, 2002; pp 452–466.

(16) Wang, J.; Wen, J.; Lofchovsky, F.H.; Ma, W. Instance-based Schema Matching for Web Databases by Domain-specific Query Probing. Proceedings of the 30th International Conference on Very Large Data Bases, Toronto, Canada, 2004; pp 408–419.

(17) Bilke, A.; Naumann, F. Schema Matching Using Duplicates. Proceedings of the 21st International Conference on Data Engineering, Tokyo, Japan, 2005; pp 69–80.

(18) Chua, C.E.H.; Chiang, R.H.L.; Lim, E.P. Instance-based Attribute Identification in Database Integration. VLDB J. 2003, 12 (3), 228–243.

(19) Doan, A.H.; Madhavan, J.; Domingos, P., Halevy A. Ontology Matching: A Machine Learning Approach. Handbook on Ontologies in Information Systems, Springer: Berlin, Heidelberg, 2004; pp 385–403.

(20) Engmann, D.; Massmann, S. Instance Matching with COMA++, Proceedings of the Workshop on BTW Workshops, Aachen, Germany, 2007; pp 28–37.

(21) Do, H.H.; Rahm, E. COMA: A SYSTEM for Flexible Combination of Schema Matching Approaches. Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002; pp 610–621.

(22) Salton, G.; McGill, M.J. Introduction to Modern Information Retrieval; McGraw-Hill: New York, 1983.

(23) Do, H.H.; Melnik, S.; Rahm, E. Comparison of schema matching evaluations. Proceedings of the Workshop on Web and Databases, Springer: Berlin Heidelberg, 2003; pp 221–237.

(24) Madhavan, J.; Bernstein, P.A.; Rahm, E. Generic schema matching with Cupid. Proceedings of the 27th International Conference on Very Large Data Bases, Roma, Italy, 2001; pp 49–58.

(25) Schudt P. OpenGIS Web Processing Service implementation specification (Version 1.0.0). In Open Geospatial Consortium, OGC, Document Number: 05-007r7, 2007. http://www.opengisgeospatial.org/standards/wps (accessed May 10, 2012).