International Journal of Geographical Information Science

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tgis20

Extended FRAG-BASE schema-matching method for multi-version open GIS Web services retrieval

Nengcheng Chen \textsuperscript{a}, Jie He \textsuperscript{a}, Wei Wang \textsuperscript{a,b} & Zeqiang Chen \textsuperscript{a}

\textsuperscript{a} State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan, 430079, China
\textsuperscript{b} School of Resources and Environment, Ningxia University, Yinchuan, 750021, China

Available online: 08 Aug 2011


To link to this article: http://dx.doi.org/10.1080/13658816.2010.510474

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: http://www.tandfonline.com/page/terms-and-conditions

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan, sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.
Extended FRAG-BASE schema-matching method for multi-version open GIS Web services retrieval

Nengcheng Chena*, Jie Hea, Wei Wanga,b and Zeqiang Chena

aState Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China; bSchool of Resources and Environment, Ningxia University, Yinchuan, 750021 China

(Received 2 February 2010; final version received 12 July 2010)

The OGC Web Service (OWS) schemas have the characteristics of a complex element structure, are distributed and large scale, have differences in element naming, and are available in different versions. Applying conventional matching approaches may lead to not only poor quality, but also bad performance. In this article, the OWS schema file decomposition, fragment presentation, fragment identification, fragment element match, and combination of match results are developed based on the extended FRAG-BASE (fragment-based) schema-matching method. Different versions of Web Feature Service (WFS) and Web Coverage Service (WCS) schema-matching experiments show that the average recall of the extended FRAG-BASE matching for the schemas is above 80%, the average precision reaches 90%, the average overall achieves 85%, and the matching efficiency increases by 50% as compared with that of the COMA and CONTEXT matcher. The multi-version WFS retrieval under the Antarctic Spatial Data Infrastructure (AntSDI) data service environment demonstrates the feasibility and superiority of the extended FRAG-BASE method.

Keywords: OGC Web Service; schema matching; Web Feature Service; Web Coverage Service; AntSDI

1. Introduction

Currently, the Open GIS Consortium (OGC) provides a series of implementation specifications for various spatial data, such as Web Feature Service (WFS) (Vretanos 2005) and Web Coverage Service (WCS) (Whiteside and Evans 2008); they allow users an interoperable way to access spatial data on the Web. As different versions of OGC Web Service (OWS) implementation specifications have their own geospatial information schemas, the heterogeneous schemas of each geographical entity and feature must be transformed into a uniform normative schema to allow spatial information integration. From OWS implementation specifications, we know that service instances and their clients need not support all defined versions, but must obey the negotiation rules. Negotiation is performed using the ‘GetCapabilities’ operation. In response to a ‘GetCapabilities’ request containing a version number, an OWS instance must either respond with output that conforms to that version of the specification or negotiate a mutually agreeable version if the requested version is not implemented on the server. This means that a user only gets a service version nearest to the user demanded version through negotiation rules. To allow the user to obtain the exact
version, different service versions need to be transformed into a specified one. For this purpose, the schema match algorithm should be employed to identify and match associated elements between two input schemas.

The WFS specification, as well as the WCS specification, is updated from time to time and there are many different versions for Web service. Accordingly, different versions of Web service schema files differ in their structure and content. These differences are as follows:

1. Structural conflicts because of alternative modeling methods in different versions of Web service schemas: for example, the operations of WFS are defined in different schema files in WFS 1.0.0 and previous versions, but in the WFS 1.1.0 version all operations are defined in a single monolithic schema;
2. Schemas for identical concepts may have structural and naming differences: the same element label may have different meanings, and different labels may express the same meaning;
3. Differences in user-defined complex types or classes: the same class in the newly released version may add or delete some attributes or sub-elements, or the constraint may have changed; and
4. Difference in data instance: different versions may correspond to different data instance.

In addition, every Web service schema has the following characteristics: (1) a complex element structure: a schema may contain not only simple data types and elements, but also user-defined complex data types and classes; (2) a large-scale distribution: a schema may be distributed over multiple documents and namespaces: the XML Schema definition (XSD) provides different directives (include, redefine, and import) to incorporate component definitions from one document to another; and (3) a diverse set of sub-schemas: a schema file is composed of either independent sub-schemas or a file that can be decomposed into many sub-schemas.

From a comprehensive comparison given in Do et al. (2003), COMA++ (Do and Rahm 2006) is currently the most excellent schema match system. However, how to apply a fragment-based (FRAG-BASE) matching method included in COMA++ for multi-version Open GIS services retrieval is still an issue because of its characteristics of complex element structure, large-scale distribution, and diverse sub-schemas. The schema matching for multi-version open GIS services retrieval encounters the following difficulties: (1) How to apply the FRAG-BASE match process proposed in COMA++ to meet the requirements of OWS schema decomposition and identification? (2) How to evaluate the quality of COMA, CONTEXT, and the FRAG-BASE matching approach for multi-version OWS schema matching? (3) How to demonstrate its feasibility in real-world multi-version GIS service retrieval?

The objective of this article is twofold. First, it proposes and describes the extended FRAG-BASE schema-matching method based on COMA++, including an improved schema decomposition algorithm and schema fragments identification algorithm, which enable COMA++-based support to OWS schema matching. The second objective is to explore the uses that this extended FRAG-BASE schema-matching method can provide to retrieve the different versions of OWS under the Antarctic Spatial Data Infrastructure (AntSDI) (Steffen 2007) environment, showing examples of schema matching functionality improvements.

The rest of the article is organized as follows: a motivating example is described in Section 2. The background is introduced in Section 3, with the extended FRAG-BASE schema-matching method proposed in COMA++ for multi-version OWS retrieval in
Section 4. The quality of the COMA, CONTEXT, and extended COMA++ methods for WFS and WCS is tested in Section 5. Section 6 demonstrates the prototype and the application in AntSDI and in Section 7 the results are analyzed. Finally, Section 8 summarizes the conclusions and outlines future work.

2. A motivating example

Antarctica is the coldest, driest, and most inaccessible continent on Earth. The extreme climate, its remoteness, and sparse human population distinguish the white continent from the rest of the world. Antarctica plays a key role in numerous scientific questions, many of which are related to global climate change. In most of these research activities, the spatial component is crucial. King George Island is one of the South Shetland Islands. It is located close to the northern tip of the Antarctic Peninsula. The Great Wall station is a year-round station operated by China, opened in 1985, and located on the south of the King George Island.

AntSDI is sponsored by the Scientific Committee on Antarctic Research (SCAR 2009). Its Standing Committee on Antarctic Geographic Information (SC-AGI) is responsible for maintaining spatial data for Antarctica and sharing it through the application of OGC specifications. AntSDI provides not only all kinds of services but also many standards and specifications for Antarctic data, such as the SCAR Antarctic Digital Database (ADD) (Alexander 2006) WFS, King George Island GIS (KGIS) (Steffen 2000) WFS, and Chinese Antarctica Center of Surveying and Mapping (CACSM) WFS (Chen 2010c).

To illustrate the proposed solution, we use the following example throughout the article. Assume a biologist, Mike, wants to know: ‘What is the terrain situation of the Great Wall station on King George Island?’ The latitude of Great Wall station is $-62.217^\circ$, and the longitude is $-58.962^\circ$. He would go through the following steps to answer this question using the distributed heterogeneous data services:

1. Retrieve the small-scale terrain data by ADD WFS 1.0.0: the ADD is a compilation of small-scale topographic data for the continent of Antarctica. It is derived from a wide variety of sources and aims to provide the best currently available data in all areas. The most detailed information is taken from maps compiled at a scale of 1:1,000,000 (i.e., a resolution of approximately 5 m) and the least detailed from sources with a resolution of at best 5 km. A WFS has been developed to provide access to the ADD. If Mike uses a ‘GetFeature’ request (i.e., ‘http://www.add.scar.org:8080/geoserver/wfs?SERVICE=WFS&REQUEST=GetFeature&typename=add:cnt10_linestring&VERSION=1.0.0&BBOX=-64,-60,-60,-56&CRS=EPSG:6.11.2:4326&FORMAT=GML2’) to the ADD WFS, he can download the contour data of the Great Wall station in Geography Markup Language (GML) 2.0 format at a scale of 1:1,000,000.

2. Retrieve the medium-scale terrain data by KGIS WFS 1.1.0: The SCAR KGIS project provides a topographic database for King George Island, South Shetland Islands. The data sources cover a wide range of topics including topography, near-shore bathymetry, surface hydrology, glaciology, vegetation, cultural features, and human impacts. Today, a GeoServer WFS 1.1.0 server is deployed to provide access to the KGIS. If Mike uses a ‘GetFeature’ request (i.e., ‘http://www.kgis.scar.org:7070/geoserver/wfs?SERVICE=WFS&REQUEST=GetFeature&VERSION=1.1.0&TYPENAME=Contour_025&BBOX=-64,-59,-61,-57&CRS=EPSG:6.11.2:4326&FORMAT=GML2’) to the KGIS WFS, he can download the contour data of the Great Wall station in Geography Markup Language (GML) 2.0 format at a scale of 1:1,000,000.
2:4326&FORMAT=GML3') to the KGIS WFS, he can download the contour data of the Great Wall station in GML 3.0 format at a scale of 1:250,000.

3) Retrieve the large-scale terrain data by CACSM WFS 1.1.0: The CACSM project provides a topographic database for Great Wall station. The data sources cover a wide range of topics including elevation, contour, coast line, lake, land, river, and building. Today, a Deegree WFS 1.1.0 server is deployed to provide access to the CACSM. If Mike uses a ‘GetFeature’ request (i.e., ‘http://swe.whu.edu.cn:9000/deegree-wfs/services?REQUEST=GetFeature&typename=CACSM:Building&namespace=xmlns(CACSM=http://www.deegree.org/app)&Coordinates=-63,-59,-61,-57&version=1.1.0&service=WFS’) to the CACSM WFS, he can download the building data of the Great Wall station in GML 3.0 format at a scale of 1:2000.

4) Schema matching and information extraction between WFS 1.0.0 and WFS 1.1.0: With the ADD WFS KGIS WFS, and CACSM WFS, general users can obtain small-scale feature data in GML 2.0, medium-scale feature data in GML 3.0, and large-scale feature data in GML 3.0. If Mike wants to overlay the different format data on a client which only supports WFS 1.0.0 interface and GML 2.0 information model, schema-matching component and dynamic information extraction component are needed to transform GML 3.0 into GML 2.0 on the fly.

5) Integrate the diverse data on the OWS client: General users can easily manipulate and visualize the different versions of geospatial data of personal interest in the OWS client through the schema matching and information extraction service middleware. The key points of the example are the matching of different schemas and dynamic information extraction service middleware (Castano et al. 2001).

3. Background

This section introduces some concepts and technologies related to our approach.

3.1. OWS specification and schema

The OGC WFS specification uses HTTP as the distributed computing platform to define interfaces for describing data access and manipulation operations for geographic features. Through these interfaces, a Web user or service can insert, update, delete, query, and discover geographic features from different sources. The WFS specification is constantly being developed and improved. There are 0.9.0, 1.0.0, and 1.1.0 versions. WFS specification (version 1.0.0) is the version derived from the OGC Transaction Encoding Specification (Vretanos 2001) and the Spatial Object Transfer Format (SOTF) (Percivall 2002) specification. The current version of the WFS specification (version 1.1.0) is based on GML 3, whereas the WFS specification (version 1.0.0) is based on GML 2. GML 2 is only used to handle simple features, such as linear geometries restricted to one or two dimensions. GML 3 includes complex geometries, spatial and temporal reference systems, topology, units of measurement, metadata, gridded data, and default styles for feature and coverage than GML 2. Typical WFS implementations include Deegree (2010a), GeoServer (2010a), and ArcGIS Server (ESRI 2007).

The OGC WCS supports electronic retrieval of geospatial data as ‘coverages’ – that is, digital geospatial information representing space-varying phenomena. The WCS specification is also in development and is improving. There are 0.7, 0.9.0, 1.0, 1.1.0, and 1.1.2 versions. From version 1.0, WCS supports transaction operation. The latest WCS version is
1.1.2. The typical WCS implementations include Deegree (2010b), GeoServer (2010b), ArcGIS Server (ESRI 2007), and CSISS (2010) WCS.

3.2. **Schema-matching approach**

Schema matching is the task of finding semantic correspondences between elements of two schemas (Milo and Zohar 1998, Mitra et al. 1999, Ge 2002). Various systems and approaches have recently been developed to determine whether schema matches automatically or semi-automatically.

The learning source descriptions (LSD) system (Doan et al. 2000) uses machine learning techniques to match a new data source against a previously determined global scheme, and forms a 1:1 element-level mapping, which represents a powerful composite approach with an automatic combination of match results. LSD is primarily instance oriented and, to improve match accuracy, LSD has to consider user-supplied domain constraints on the global schema to eliminate some of the previously determined match candidates; the only schema type supported by LSD is XML (Doan et al. 2001).

Cupid (Madhavan et al. 2001) uses a sophisticated hybrid match approach combining a name matcher with a structural match algorithm, which derives the similarity of elements based on the similarity of their components, thereby emphasizing the name and data type similarities present at the finest level of granularity (leaf level). The Cupid matching algorithm consists of three phases: linguistic matching, structure matching, and combination matching. In the structure-matching phase, the similarity of leaf context is used in a lower match quality.

SemInt (Li and Clifton 2000, Li et al. 2000) creates a mapping between individual attributes of two schemas. It is a powerful and flexible approach for hybrid matching; multiple match criteria can be selected and evaluated together. However, it does not support name-based matching or graph matching and is difficult to determine a useful mapping to the [0,1] interval. It uses neural networks to determine match candidates for large match tasks and has substantial performance problems (Clifton et al. 1997).

CONTEXT (Aviv and Eran 2009) is a simple match approach which constitutes only one PATH matcher. It matches two schemas from the root element, finds out all paths from the root node to its leaves, and identifies all paths by a shared element. It uses Name matcher to compute name similarities between the nodes on the given two paths and a combined value to derive the path similarity. Compared to Name matcher, CONTEXT considers all contexts of the matching nodes and the whole structure of schema, so it can achieve a better quality. Moreover, the idea of path-level matching can help instance-based matchers to distinguish between schema elements with similar instances.

COMA (Do et al. 2003) is a generic match system to combine multiple matchers flexibly and supports different ways of combining match results. In COMA system, the supported matchers include simple matchers, hybrid matchers, and COMA combination matcher. The simple matchers are Affix, N-gram, EditDistance, Soundex, Synonym, and DataType, whereas the hybrid matchers are Name, NamePath, TypeName, Children, and Leaves; combination matcher is the combination of Name, Path, and Leaves matcher. New match algorithms can be added in COMA and used in combination with other matchers. COMA is also a platform to evaluate the effectiveness of different matchers.

COMA++ (Do and Rahm 2002, Rahm et al. 2004, Aumueller et al. 2005) is an extension of COMA and with many improvements, such as a rich graphic user interface, new matchers, and a FRAG-BASE match method to deal with very large schemas. There are more than 15 matchers included in COMA++ matcher library, and NameState, Parent, and Siblings
matchers are not included in COMA. COMA++ uses FilteredContext strategies to support iterative refinement filter for relevant elements before matching and the FRAG-BASE method to achieve high quality and better performance for large schemas.

In Section 4, we will show how our work borrows ideas from COMA++ and how to extend the schema decomposition algorithm and schema fragments identification algorithm based on COMA++ to meet the requirements of multi-version OWS retrieval.

4. The extended FRAG-BASE schema-matching method

The FRAG-BASE schema-matching method, grounded on the idea of divide-and-conquer as far as a big and complicated schema match problem is concerned, decomposes schema files into small fragments and achieves accurate matching of two schema files through fragments matching.

Figure 1 shows the overview of the OWS schema-matching method using COMA++, which consists of three core components: schema decomposer, fragment identifier, and a schema-matching component. The schema decomposer decomposes the input schema and utilizes the directed graph to represent the fragments. The fragment identifier identifies all similar fragments between the source and target schemas. The schema-matching component matches elements between fragments and combines all match results between fragments of the source and target schemas. The extended FRAG-BASE matching method requires four steps in all: schema files decomposition and fragments representation, similar fragments identification, fragments elements matching, and combination of match results.

4.1. Schema decomposition and fragment representation

Schema file decomposition is based on a schema graph (or tree). There has been much work done on fragment construction (Hu and Qu 2006). Seidenberg and Rector (2006) designed a way to construct standalone fragments by exhaustively traversing various links.
(e.g., properties) from a specified entity. They applied the detailed semantics captured with ontology described by the W3C Web Ontology Language (OWL) (W3C 2009) to produce highly relevant segments. Stuckenschmidt and Klain (2007) defined a framework for modular ontology based on distributed description logics, but how to manipulate the sizes of the modules was not given. Tu et al. (2005) presented a novel ontology visualization approach, which produced a holistic imaging of the ontology that contains a semantic layout of the ontology classes, with the visualization of instances distribution and relations. It uses a data clustering algorithm (the force-directed placement algorithm) to partition a large ontology. The basic idea of the approach in Tu et al. (2005) was to transform ontology into graphs and to use clustering algorithms to partition the graphs. But the method does not provide any mechanism to preserve the triples that contain blank nodes. In this article, we decompose schema files in the following two steps:

**Step 1.** Transform the schema files into trees. As we know, a Web service schema contains a series of distributed external schemas by the ‘include’ or ‘import’ tag. In this article, we download and parse the external schemas dynamically according to the URL address, schema of specified OWS is represented by a tree. We take ‘wfs.xsd’ schema file of WFS 1.1.0 and ‘WFS_basic.xsd’ of WFS 1.0.0 as an example to explain the procedure in detail. As shown in Figure 2, three distributed ‘gml.xsd’, ‘filter.xsd’, and ‘owsAll.xsd’ schema files have been imported into ‘wfs.xsd’ schema file, two distributed ‘feature.xsd’ and ‘filter.xsd’ schema files have been imported into ‘WFS_basic.xsd’ schema file. All the schema files are in different namespace. The two imported ‘filter.xsd’ schema files have same name; however, one is version 1.1.0 and the other is version 1.0.0. The two schemas contain diverse simple and complex elements, the two schema files are represented in trees after transformation (shown in Figure 2).

**Step 2.** Decompose the schema tree into fragments. In this article, we divide the schema tree based on the similarity measure between the source schema fragments and the target schema fragments; the source schema is the old version, the target schema is the new version, and we assume that the new version is updated from the old one, and most of fragments can find an image in the target schema tree. Figure 3 is the pseudo-code of schema decomposition. ‘SchemaDecomposition’ takes source and target roots as input; each source or target root is the representation of a schema or schema fragment. For each source root, if the target roots are numerous, the algorithm will first find a similar root in the target roots (line 3); if the target is found, then the similar root in the target roots is deleted (line 4). If there is no similar root in the target roots, the selected source root is deleted; if the target roots have only one root, the target tree is first divided into many sub-trees according to its root. If a similar root is found between the source roots and target sub-trees roots, the similar target root is deleted; if there is not a similar root in the target roots, the selected source root is deleted (lines 6–10).

If the source root is null, all the similar root pairs are found between source roots and target roots, then the decomposition is ended and returns to the target fragments (lines 11–13); or else, if the recursive times are larger than 2, the decomposition is ended (lines 14–16), because too many recursive iterations may lead not only to a lower performance but will also make the algorithm more complicated. If the recursive times are less than 2, then the algorithm is recalled continually (lines 17–19). In line 3, the function ‘hasSimilarRootPair’
computes the most similar fragments using the algorithm described in Section 4.2. ‘DivideTree’ takes tree roots as input and divides the tree as follows: it deletes the tree’s root, deletes all the connection lines from the root to its direct children, and divides the tree into many sub-trees, each child node becoming a new sub-tree root, and generates the sub-tree root.

The fragments obtained from the decomposition of the schema file fall into two types: sub-schemas and internal schemas. Sub-schemas refer to those independent fragments, including some complex data types defined in the schema file (e.g., Capabilities in wcsGetCapabilities.xsd) or some independent operation types (e.g., GetCapabilities in wcsGetCapabilities.xsd). The internal schemas indicate the fragments obtained when using the decompose technique.

Figure 4 shows the decomposed sub-schema fragments after ‘decompose’ operation; the ‘wfs.xsd’ schema file is decomposed into 10 independent fragments, viz. Transaction, LockFeature, DescribeFeatureType, TransactionResponse, FeatureCollection, GetGMLObject,
Algorithm: schemaDecomposition(srcRoots, targRoots)

Input: the target roots and source roots, each source root represent a source fragment, each target root represent a target fragment, the target tree is divided according to the source fragment root

Output: a set of target fragments

1. foreach(root in srcRoots)
2. if(srcRoots.size>1)
3. if(hasSimilarRootPair(root, targRoots))
4. removeRoot(root, targRoots[i]);
5. else continue; //next root in srcRoots
6. else //only one tree
7. targRoots = divideTree(targRoots); //divide the tree into many subtrees, the first divide
8. if(hasSimilarRootPair(root, targRoots))
9. removeRoot(root, targRoots[i]);
10. else continue; //next root in srcRoots
11. if(srcRoots is null)
12. Fragments = cloneTree(labeled root in targRoots);
13. Return Fragments;
14. else if(schemaDecompositionNumber>2)
15. Fragments = cloneTree(labeled root in targRoots);
16. return Fragments;
17. else
18. schemaDecompositionNumber++;
19. schemaDecomposition(srcRoots, targetRoots);

Figure 3. The pseudo-code of schema decomposition algorithm.

WFS_Capabilities, GetFeature, GetCapabilities, and GetFeatureWithLock; the ‘WFS_basic.xsd’ schema file is decomposed into four independent fragments, viz. FeatureCollection, DescribeFeatureType, GetFeature, and GetCapabilities.

4.2. Schema fragment identification algorithm

The goal of this step is to identify fragments generated from Section 4.1 that are sufficiently similar to be worth matching in detail. The schema fragment metadata contains fragment name, contents, statistical data, etc. The schema fragment identification is based on the similar root pairs in Section 4.1 (the match cardinality is n:1). For each pair of schema fragments to be matched, the element-level match is achieved by comparing the corresponding root element names and the fragment’s structures. According to the combination value of name and structure similarity, those fragment pairs with the largest combination similarity are supposed to be similar. The fragment identification algorithm is shown as follows:

Algorithm 1: Schema fragment identification algorithm.
Input: Source schema fragment S and target schema fragment T.
Output: Similar schema fragment pairs.
Identify Fragments (S, T).
(1) Input all source schema fragments \( S \) and target schema fragments \( T \).

(2) For each target schema fragment \( T_i \), Formula (1) is employed to compute the similarities between its root elements and all source schema fragments \( S_j \) root element names, and the one with the greatest similarity is chosen as the name similarity \( E_s \) between root elements.

\[
E_{i,j} = \text{NameSimilarity}(\text{root}_i, \text{root}_j) \quad (0 < i < m, \ 0 < j < n, \ m, n \text{ are, respectively, the number of target and source schema fragments})
\]

where \( E_{i,j} \) is the similarity between the \( i \)th target fragment root node and the \( j \)th source fragment root node; \text{root}_i \) and \text{root}_j \), respectively, refer to the \( i \)th target root node and the \( j \)th source root node.

The calculation of \( \text{NameSimilarity} \) (\text{root}_i, \text{root}_j) \) is as follows:

(i) Translate the string of \text{root}_i \) and \text{root}_j \) to lowercase, if \( \text{root}_i = \text{root}_j \), then \( E_{i,j} = 1.0 \), the similarity computation is ended, or else go to step ii.

(ii) Compute the constituents of \text{root}_i \) and \text{root}_j \); if \text{root}_i \) and \text{root}_j \) are constituted by only one token, then compute the edit distance between \text{root}_i \) and \text{root}_j \), and get a similarity \( \text{sim}_{\text{editDist}} \), then \( E_{i,j} = \text{sim}_{\text{editDist}} \), the computation is ended, or else go to step iii.

(iii) Compute \( \text{sim}_{\text{editDist}} \) among all constituents of \text{root}_i \) and \text{root}_j \); combine all \( \text{sim}_{\text{editDist}} \), then \( E_{i,j} \) equals to the overall similarity, the computation is ended.

Figure 4. The decomposed fragments of ‘\textit{wfs.xsd}’ and ‘\textit{WFS\_basic.xsd}’ schema trees.
(3) Structurally, compare the number and length of paths in sub-graphs between the target schema fragments and source schema fragments. If every path in the target schema graph can find a corresponding path in the source schema graph, the similarity is 1; or else, schema similarity $S_s$ refers to the ratio between the number of matched paths (between target schema fragments and source schema fragments) and the total number of target schema fragment paths. The formula for computing the similarity is

$$S_s = \frac{\text{paths}_{(s,t)}}{\text{paths}_t}$$

(2)

where $\text{paths}_{(s,t)}$ is the number of matched paths and $\text{paths}_t$ is the total number of paths for target schema fragments.

(4) For every target schema fragment, compute the combination value of element similarity and structure similarity between them and the fragments of the source schema. The schema fragment pair with the greatest similarity is regarded as the similar fragment pair to output. The formula for computing the combination similarity is as follows:

$$\text{Sim}_{(s,t)} = zE_s + (1 - z)S_s \quad 0 < z < 1$$

(3)

where $E_s$ is the element similarity, $S_s$ the structure similarity, and $z$ the weighting factor, the value of which is decided by domain experts. Figure 5 shows the similar fragments between 'wfs.xsd' and 'WFS_basic.xsd', the similar fragments include FeatureCollection, DescribeFeatureType, GetFeature, and GetCapabilities.

4.3. Fragment match and combination of 2match results

After having identified the source and target schema fragments, various match algorithms can be used on elements and attributes in these fragments to implement fragment match. Figure 6 shows the procedure of fragment match, which takes four steps:

Step 1: Schema fragment selection. The fragment pairs identified in Section 4.2 are stored in the fragment manager. Each time, a pair of fragments (e.g., $S_i \leftrightarrow T_j$) is selected from the fragment manager to match.

Step 2: Fragment match. Every similar fragment pair represents an independent match problem. For the match between each pair of fragments, we use Algorithm 2 to compute the mapping relationship between the two fragment elements. According to the complexity of fragments, simple or complex matchers are used for matching. In other words, if the fragments only contain leaf-level simple elements, we select a string-based matcher, for example, name matcher, for matching; if the depth of the fragment tree (or graph) is larger than 2, we use element- and structure-based combination matchers, for example, Cupid or COMA for matching. The match results of each pair of fragments are stored in a temp mapping file.

Step 3: Combination of match results. After matching all fragment pairs, all mappings stored in the mapping file are combined. The combination technique for mappings is selected according to the types of fragment pairs. If both fragments in a pair are independent sub-schema fragments, all mappings are
directly combined during the combination; if the fragments are internal schema fragments, then before combination, the paths from the schema root node are added to the internal schema root node before the mapping path of internal schema fragments, and then the mappings are combined. The combination results are a series of mappings containing the correspondences between schema elements.

**Algorithm 2:** Schema fragment-based match algorithm.

Input: Similar schema fragment pairs (Si, Tj).

Output: Mappings between schema elements.

**MatchFragment** (Si, Tj).

1. Generate schema graphs $G_s$ and $G_t$ for the corresponding fragments of the source schema and target schema, $S_i$ and $T_j$.
2. Compute the depth of fragment schema graph according to Formula (4) and then determine the type of matcher to be selected according to the computed depth:

$$L_s = \text{Depth}(G_s, G_t)$$ (4)

where $L_s$ is the maximum depth of schema graphs $G_s$, $G_t$.
3. Match two fragment schema graphs with the selected matcher.

Figure 5. The similar fragments between ‘wfs.xsd’ and ‘WFS_basic.xsd’.
5. Experiments

5.1. Schema fragment identification experiments

5.1.1. WFS schema fragment identification experiments

Figure 7 shows the match results between fragments in ‘wfs.xsd’ and ‘WFS_basic.xsd’; there are 21 mapping items of corresponding elements.
5.1.2. WCS schema fragment identification experiments

Taking two versions of schema files `getCoverage.xsd` and `wcsGetCoverage.xsd` as examples, and using the same algorithm as described in Section 4.2, the root node name similarities are shown in Table 4. The results in Table 5 are the structure similarities. Table 6 reflects the combination similarities; the correspondences of similar fragments are shown in Figure 8.

Table 1. Similarity between schema fragment root nodes for different WFSs.

<table>
<thead>
<tr>
<th></th>
<th>GetCapabilities</th>
<th>DescribeFeatureType</th>
<th>GetFeature</th>
<th>FeatureCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeatureCollection</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>GetCapabilities</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>GetFeature</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Transaction</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>DescribeFeatureType</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Query</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Lock</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
5.2. Match quality experiments

Precision, recall, and overall (Rahm and Bernstein 2001) are employed to assess the match quality of the schema-matching technique based on fragments.

1. Precision reflects the share of real correspondences among all found ones:

\[
\text{Precision} = \frac{T}{P} = \frac{T}{T + F}
\]  

(5)

2. Recall specifies the share of real correspondences that is found:

Table 2. Structure similarity between schema fragments for different WFSs.

<table>
<thead>
<tr>
<th></th>
<th>GetCapabilities</th>
<th>DescribeFeatureType</th>
<th>GetFeature</th>
<th>FeatureCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeatureCollection</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td>GetCapabilities</td>
<td>1.0</td>
<td>0.25</td>
<td>0.25</td>
<td>0.0</td>
</tr>
<tr>
<td>GetFeature</td>
<td>0.67</td>
<td>0.33</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Transaction</td>
<td>0.33</td>
<td>0.33</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>DescribeFeatureType</td>
<td>0.33</td>
<td>0.67</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>Query</td>
<td>0.0</td>
<td>0.05</td>
<td>0.12</td>
<td>0.0</td>
</tr>
<tr>
<td>Lock</td>
<td>0.0</td>
<td>0.25</td>
<td>0.06</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 3. Combining similarity between schema fragments for different WFSs.

<table>
<thead>
<tr>
<th></th>
<th>GetCapabilities</th>
<th>DescribeFeatureType</th>
<th>GetFeature</th>
<th>FeatureCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeatureCollection</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.8</td>
</tr>
<tr>
<td>GetCapabilities</td>
<td>1.0</td>
<td>0.01</td>
<td>0.01</td>
<td>0.0</td>
</tr>
<tr>
<td>GetFeature</td>
<td>0.27</td>
<td>0.13</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Transaction</td>
<td>0.13</td>
<td>0.13</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>DescribeFeatureType</td>
<td>0.13</td>
<td>0.87</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>Query</td>
<td>0.0</td>
<td>0.02</td>
<td>0.05</td>
<td>0.0</td>
</tr>
<tr>
<td>Lock</td>
<td>0.0</td>
<td>0.01</td>
<td>0.02</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4. Similarity between schema fragment root nodes for different WCSs.

<table>
<thead>
<tr>
<th></th>
<th>Identifier</th>
<th>DomainSubset</th>
<th>RangeSubset</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceCoverage</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>domainSubset</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>rangeSubset</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>interpolationMethod</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>output</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Recall $= \frac{T}{R}$ \hspace{1cm} (6)

(3) Overall represents a combined measure for match quality, taking into account the post-match effort needed both for removing false matches and adding missed matches:

$$\text{Overall} = \text{Recall} \times 2 - \frac{1}{\text{Precision}} \hspace{1cm} (7)$$

where $T$ represents the true matches, $P$ all matches, $F$ false matches, and $R$ all true matches.
All schema-matching tests and time measurements were uniformly achieved using Sun Java 1.6.0 libraries on a Windows machine equipped with a 3.0 GHz Intel Xeon processor and 2.0 GB RAM. For the CONTEXT match approach, the constituent matcher is path, the similarity cube aggregation strategy is average. The directional strategy is both. The similarity value combining strategy is average. The match candidate selection method is multi-factor, that is multiple (0, 0.01, 0.5), which is customized in COMA++. For the COMA match approach, the constituent matchers include name, path, leaves, parent, etc. The similarity cube aggregation strategy and the directional strategy, and the similarity value combining strategy are the same as for CONTEXT. The match candidate selection method is also multi-factor, that is multiple (0, 0.008, 0.5), which is defined in COMA.

5.2.1. WFS schema match quality experiment

The experiment chooses wfs.xsd, the version 1.1.0 WFS, as the source schema file, and version 1.0.0 WFS-basic, WFS-capabilities, and WFS-transaction.xsd as the target schema files. Three match approaches, COMA, CONTEXT, and FRAG-BASE, are applied to match the two versions of schemas. The basic string-matching approach is selected for matching fragment elements. From Figures 9–11, it can be observed that, compared to complete schema matching, for example, WFS-capabilities.xsd, its recall and overall are significantly lower than the match results based on the sub-schema fragments.

5.2.2. WCS schema match quality experiment

The match schemas are divided into three groups: wcsGetCapabilities between wcsGetCapabilities.xsd and wcsCapabilities.xsd; DescribeCoverage between wcsDescribeCoverage.xsd and describeCoverage.xsd; and GetCoverage between wcsGetCoverage.xsd and getCoverage.xsd. Three match prototypes, COMA, CONTEXT, and FRAG-BASE, are used to match the two versions of schemas. From the match results shown in Figures 9–11, we know that for wcsGetCapabilities and DescribeCoverage, the FRAG-BASE method yields the best recall, precision, and overall, and recall and precision are up to 80%; for getFeature, because it is a complete schema matching, and did not decompose the schemas, the match quality is lower than wcsGetCapabilities or DescribeCoverage.

Figure 9. Recall of three match methods for WFS and WCS.
5.3. Schema-matching efficiency experiment

By comparing the matching efficiencies of WCS schemas and WFS schemas in Figure 12, it is clear that, for Web service schemas, the extended FRAG-BASE matching technique greatly improves the matching performance. For the same service schema, the extended FRAG-BASE matching technique takes about 10%–20% of the time needed by COMA and saves 50%–70% of time as compared with CONTEXT.

6. Prototype

6.1. Implementation

A prototype system composed of the following components using Java Web service technology has been developed (Figure 13):

1. GeoMatcher component: Its main function is to generate the style sheet language file of the matching result through schema files decomposition, fragments representation, similar fragments identification, fragments elements matching, and match results combination.
GeoExtractor component: Its main function is to make use of the style sheet language file generated by the GeoMatcher component and XSL processor to convert the XML file to a new XML file that meets the user requirements.

Uniform OWSs proxy service (Chen 2010a): It explores two standard interfaces to other clients. They are WCS and WFS. The proxy service is designed to be capable of retrieving data from different sources and to meet the different versions’ requests. The proxy service dynamically supports the following conversions through GeoMatcher and GeoExtractor components: WCS server from a different version and WFS server from a different version. Different plug-in Web services are developed to support the retrieval of multi-version OWSs. A proxy service enables the integrated retrieval of data from internal services and external services.

GeoMPGC Client (Chen 2010b): GeoMPGC provides an interoperable way of accessing geospatial Web services, especially those from the Open Geospatial Consortium (OGC), for integrating and analyzing distributed heterogeneous Earth science data. GeoMPGC conforms to the OGC Catalogue Service for Web (CSW) specification to play a ‘directory’ role that permits the registry, discovery, and access of geospatial information resources that are distributed on the Internet. By
implementing the latest protocols of the OGC Web Feature Service (WFS), Web Map Service (WMS), and Web Coverage Service (WCS), GeoMPGC provides a single point of entry for access to OGC-compliant data services around the world, to request any subsets of multi-dimensional and multi-temporal geospatial data for a specific geographic region.

6.2. Use case

In the example presented in Section 2, there are ADD WFS, KGIS WFS, and CACSM WFS. If we use WFS 1.0.0 as the request, we must have a uniform WFS. In this example, we first apply the GeoMatcher component to perform schema matching between WFS 1.1.0 and 1.0.0 schemas, the results of schema matching can be accessed using the URL http://swe.whu.edu.cn:9000/geosurf/fragment; second, the GeoExtractor component is used to extract the information into WHU WFS 1.0.0 (http://swe.whu.edu.cn:9000/geosurf/wfs) from ADD WFS (Geoserver 1.0.0), KGIS WFS (Geoserver 1.1.0), and CACSM WFS (Deegree 1.1.0); finally, the GeoMPGC (Chen 2010b) can retrieve the spatial information from the WHU WFS 1.0.0. The GML 2.0 data from WHU WFS 1.0.0 can be integrated and shown in GeoMPGC (Figure 14). There are three feature types of ADD (station, road, and contour100), three feature types of KGIS (contour050, contour025, and contour010), and four feature types of CACSM (contour005, building, coast line, and ice). Thus, the user can manipulate, query, and visualize the ‘Great Wall station’ data from diverse data source served by different version WFSs using GeoMPGC. The detailed interaction of GeoMPGC can be accessed using the URL http://swe.whu.edu.cn:9000/geompgc/about.htm.

Figure 14. ‘Great Wall station’ data from different WFSs (ADD, KGIS, CACSM) in GeoMPGC.
7. Discussions

In this article, we have extended the schema decomposition algorithm and schema fragments identification algorithm based on COMA++. For these tests, schemas are composed of independent sub-schemas; the decomposition algorithm searches the similar fragments (by searching similar roots) in the high-version schema file according to the low-version schema fragments. These independent sub-schema fragments are taken directly as the schema fragments to be decomposed (e.g., WCS group wcsGetCapabilities and group DescribeCoverage schema files, WFS schema files wfs-basic.xsd and wfs-transaction.xsd). For these schemas comprised of only one schema fragment, such as WCS group getCoverage schema files, it is comprised of one schema fragment getCoverage. The decomposition algorithm first divides the source schema getCoverage.xsd into five fragments: sourceCoverage, domainSubset, rangeSubset, interpolationMethod, and output. The sourceCoverage and interpolationMethod fragments consist of a single element. The target schema wcsGetCoverage.xsd is decomposed into four fragments: Identifier, RangeSubset, DomainSubset, and Output; the fragment Identifier contains only one element. And the decomposition algorithm can easily find that domainSubset and DomainSubset, rangeSubset and RangeSubset, and output and Output are similar fragments. The fragments pair Identifier and sourceCoverage may need further testing. From the results of fragments identification in Tables 1–6, and Figures 7 and 8, it can be observed that Algorithm 1 fully identifies all the similar fragments between the source schemas and the target schemas.

As for the quality of fragments match, the significant reduction of the match schema size and lower complexity of schema files improve the match quality of the FRAG-BASE matching technique. From Figures 9–11, we see that for the matches of WCS schema pair wcsGetCapabilities and DescribeCoverage, the recall, precision, and overall of FRAG-BASE matching are better than those of COMA and CONTEXT. Moreover, the experimental results reveal that the recall (Figure 9), precision (Figure 10), and overall (Figure 11) of matching WCS schema pair getCoverage are 20% worse than those of the match results of wcsGetCapabilities and DescribeCoverage schema files. The getCoverage schema file, as an independent schema fragment, is much more complex than other schema files; therefore, even with the same matching technique, it needs more matching time and has a higher probability of generating false matches. Besides, the information models of the two versions of the getCoverage schema files are quite different from each other and need to be compared more semantically. Thus, the match quality of getCoverage schema files is worse than the matches of the other two types of schema files. Also, the recall (Figure 9) and overall (Figure 11) of matches for WFS schema are 20% worse than the match results of the other two types of schema files.

As to the fragment-matching performance, the matching time is mainly spent on the identification of schema fragments and fragment matches, and because not all the elements in the source and target schemas are tested, the performance of the FRAG-BASE matching is the best among these generic matching techniques. From Figure 12, we can see that for the same service schema, the FRAG-BASE matching technique takes about 10%–20% of the time needed by COMA and saves 50%–70% of time as compared with CONTEXT.

8. Conclusions and future work

This article proposes an extended FRAG-BASE schema-matching method based on COMA++ which, for a problem with large schema to be matched, decomposes the schema files into small fragments and then matches the fragments. As the OWS schemas have the
characteristics of complex element structure, are distributed and large scale, and have
differences in element naming and version differences, we have extended the schema
decomposition algorithm, schema fragments identification algorithm, and schema-matching
combination algorithm. The algorithms are tested by different versions of WCS and
WFS. The prototype system containing GeoMatcher, GeoExtractor, Uniform OWSs, and
GeoMPGC components is implemented using Java Web Service technology. An example of
KGIS data retrieval under AntSDI is presented. The results show the average recall
of the extended FRAG-BASE matching is above 80%, the average precision reaches 90%,
the average overall achieves 85%, and the matching efficiency increases by 50%.

Our future research will be directed to developing a semantic-based schema-matching
approach to improve the quality of OWS retrieval and study the message exchange of more
dynamic Web services, such as the Web Processing Service (WPS) (Schut 2007).

Acknowledgements
This work was supported in part by the National Basic Research Program of China (973 Program)
under Grant 2011CB707101, by the 863 Program under Grant 2007AA12Z230 and 2011AA010500,
and by the National Natural Science Foundation of China under Grant 41023001 and 41021061. We
also sincerely thank Dr. David A. Tait, for proofreading the manuscript. The authors would like to thank
the anonymous reviewers for their valuable comments and insightful ideas.

References
COMA++. In: Proceedings of the 24th ACM international conference on management of data,
Castano, S., De Antonellis, V., and De Capitani Di Vimercati, S., 2001. Global viewing of hetero-
geneous data sources. IEEE Transactions on Knowledge and Data Engineering, 13 (2), 277–297.
Chen, N., 2010b. Sensor Web Team GeoMPGC Website [online]. Wuhan University. Available from:
matching across heterogeneous databases. In: Proceedings of the 17th IFIP working conference on
testing.deegree.org/degree-wfs/[Accessed 24 March 2010].
testing.deegree.org/degree-wcs/[Accessed 24 March 2010].
Doan, A.H., Domingos, P. and Halevy, A., 2001. Reconciling schemas of disparate data sources:
York: ACM, 509–520.


