



A Service-based Approach to Schema Federation of Distributed Databases

¹Leena Al-Hussaini

²Stratis Viglas

¹Malcolm Atkinson

¹National e-Science Center, University of Edinburgh, UK

²School of Informatics, University of Edinburgh, UK

November 2005

ABSTRACT

In the last few years, we have witnessed a rapid growth in distributed database processing. We consider the question of data integration: how we can integrate distributed schemas into a new one and query just that new schema without losing the ability to retrieve data from the original schemas. The area in which we try to answer that question is federated databases, where the original heterogeneous sources are assumed to be autonomously managed. Our approach is based on schema matching: the identification of semantic correspondences between elements of different schemas. We propose SASF, the Service-based Approach to Schema Federation, which is composed of three services: schema translation, schema matching, and schema mapping that are accessed through a user portal. Our approach exploits both schema-level and instance-level information to discover semantic correspondences, and is able to operate over a full range of matching cardinalities. A demonstration of the usability of SASF in a real-world scenario of federating astronomical databases is presented as a case study. The results, we believe, illustrate the potential of performing data integration through database federation.

**Edinburgh E-Science
Technical Report EES-2006-01**

Available from http://www.nesc.ac.uk/technical_papers/EES-2006-01.pdf

Copyright © 2005 The University of Edinburgh, All rights reserved

A Service-based Approach to Schema Federation of Distributed Databases

Leena Al-Hussaini
National e-Science Center
University of Edinburgh, UK
leena@nesc.ac.uk

Stratis Viglas
School of Informatics
University of Edinburgh, UK
sviglas@inf.ed.ac.uk

Malcolm Atkinson
National e-Science Center
University of Edinburgh, UK
mpa@nesc.ac.uk

ABSTRACT

In the last few years, we have witnessed a rapid growth in distributed database processing. We consider the question of data integration: how we can integrate distributed schemas into a new one and query just that new schema without losing the ability to retrieve data from the original schemas. The area in which we try to answer that question is federated databases, where the original heterogeneous sources are assumed to be autonomously managed. Our approach is based on schema matching: the identification of semantic correspondences between elements of different schemas. We propose SASF, the Service-based Approach to Schema Federation, which is composed of three services: schema translation, schema matching, and schema mapping that are accessed through a user portal. Our approach exploits both schema-level and instance-level information to discover semantic correspondences, and is able to operate over a full range of matching cardinalities. A demonstration of the usability of SASF in a real-world scenario of federating astronomical databases is presented as a case study. The results, we believe, illustrate the potential of performing data integration through database federation.

1. INTRODUCTION

Distributed database research has been fueled recently by data integration applications. Database federation [1] is one of the ways to achieve integration, where specialized middleware is developed to provide a common gateway to heterogeneous schemas. A readily available instance of such a scenario has recently appeared in Data Grid [2] [3] [4] environments, where the nodes of the Data Grid, autonomous as they may be, need to export and be accessed through a *common* schema. As Data Grid implementations are mainly based on services, rather than dedicated servers, it is sensible to look into service-based solutions to database federation; this is the focus of our paper.

Schema federation is a much more difficult task than schema integration. The key differences are summarized in two points: (a) schema integration assumes that source schemas are local, while in schema federation source schemas are distributed; and (b) source schemas are autonomously managed in schema federation, while a centralized management approach is assumed in schema integration. Moreover, the output of schema federation is a global schema where schema elements are mapped to the original data sources, which allows users to pose queries on the global schema and retrieve data from the source databases. This is in contrast to an integrated schema where, after mapping, the data is accessible through a single schema. Data and structural changes need to be propagated to the integrated schema in order for them to be visible to the user or accommodated by the mapping and query

system. These parameters increase the complexity of the problem and naturally a question that arises is one of scale.

At the heart of database federation, and data integration in general, lies schema matching [5, 6, 7, 8, 9]: finding semantic correspondences between elements of different schemas. For example, in Fig. 1, finding that address in schema ShipTo matches custStreet, custCity, and custZip in schema Customer. We will be referring to this example throughout the rest of this paper to highlight various aspects of our proposals.

DB URI: http://321.654.987 Database: PO1 Schema: Customer	DB URI: http://132.465.798 Database: PO2 Schema: ShipTo
custNo INT (3) custName VARCHAR (200) custStreet VARCHAR (200) custCity VARCHAR (200) custZip VARCHAR (20)	poNo INT (2) custNo INT (3) custFname VARCHAR (100) custLname VARCHAR (100) address VARCHAR (500)

Figure 1. An example

Though a number of proposals have addressed different aspects of schema matching (e.g., COMA [6], SemInt [7], Cupid [8], TranSem [9], LSD [10], Artemis [11, 12], and DIKE [13]), the solutions are sometimes too specialized and/or confined, which makes their application to more distributed database federation scenario problematic. Our work which follows a service-based approach addresses the scalability problem.

More specifically, we treat schema matching as a component in a service-based database federation architecture, where we define a service to be a single unit that fulfills a phase in schema federation; e.g. translation is a phase in schema federation, thus Schema Translation service. Besides, we operate in a Grid environment which is itself based on a Service-Oriented Architecture. Our proposed architecture consists of three core services: schema translation, schema matching, and schema mapping. This results in the *Service-based Approach to Schema Federation* (SASF). Our approach exploits both schema-level and instance-level information to discover semantic correspondences, and is able to operate over a wide range of matching cardinalities with demonstrable results in real-world use-cases. The service-based approach has the added bonus that by decomposing the main problem into three core services we effectively “de-couple” them. This aids in simplifying their development, maintenance, and performance. For instance, different matching services can be used seamlessly in the architecture, provided they comply with the interfaces expected by the remaining two services. Our preliminary results, we believe, illustrate the potential of performing data integration through database federation.

The key problems that we address in this work are some of the long-standing issues in schema matching. More specifically, our project features are as follows:

(1) *Exploitation of both schema-level and instance-level information to discover semantic correspondences.* Most of the so far work in the area has concentrated on using schema-level information approach of the two approaches and a hybrid framework combining the two, to the best of our knowledge, needs more attention. For example, in Fig. 1, and for each schema element we can retrieve sample data values that provide additional information on which to base the discovery of semantic correspondences between schema elements. This contribution is fulfilled by our *Schema Matching Service* (see Section 3.2.)

(2) *Operating over a wide range of matching cardinalities.* Previous work has focused on a limited set of matching cardinalities; in contrast, we do not pose any restrictions. SASF is powerful enough to identify, $1:1$, $1:n$, $n:1$, or $n:m$ matching cardinalities without losing anything from its overall competitiveness in terms of performance or quality of the produced matches. With reference to Fig. 1, SASF can discover the following cardinalities [5]:

- a) a $1:1$ matching cardinality between `PO1.Customer.custNo` and `PO2.ShipTo.custNo`;
- b) a $1:n$ matching cardinality between `PO1.Customer.custName` and the pair `(PO2.ShipTo.custFname, PO2.ShipTo.custLname)`;
- c) a $n:1$ matching cardinality between the triplet of `(PO1.Customer.custStreet, PO1.Customer.custCity, PO1.Customer.custZip)` and `PO2.ShipTo.address`; and,
- d) a $n:m$ matching cardinality, which is defined as a set of $1:1$ matching cardinality that builds a query.

This contribution is fulfilled by our *Schema Mapping Service* when building the global schema (see Section 3.3.)

(3) *Functioning in a distributed context.* We pose no restriction on the schemas being local. Since the approach is service-based, the only restriction that we pose is that schemas can be identified by a URI and be exported (most likely through a wrapper, which may be hosted in an intermediary service) to some format that can be parsed by SASF. Observe that in Fig. 1, the two databases schemas reside on two different sites (Schema 1 is shown as `Customer @ http://321.654.987`, whereas Schema 2 resides as `ShipTo @ http://132.465.798`). This contribution is fulfilled by our *Schema Translation Service* which is able to connect to distributed databases (see Section 3.1.)

(4) *Selection of the subset of a schema that should participate in the federation process.* Existing work will follow an “all-or-nothing” approach where they use all attributes of a schema to initiate the matching process. The schemas `PO1.Customer` and `PO2.ShipTo` shown in Fig. 1 could in fact contain hundreds of attributes. We give users the option to select only the subset of attributes that they are interested in querying after the schemas have been federated. This feature avoids the waste of time translating and matching unnecessary attributes, which, in turn makes matching large schemas more feasible as matching costs are largely determined by the scale of the integrated target. This contribution is fulfilled by our *User Portal*.

The rest of this paper is organized as follows: the SASF overview is given in Section 2, while the detailed description of the architecture’s components is presented in Section 3. A real-world scenario of federating astronomical databases is presented in Section 4, while a more detailed comparison with existing approaches is the subject of Section 5. Finally, we draw some conclusions and identify future research directions in Section 6.

2. OVERVIEW OF SASF

Our proposed Service-based Approach to Schema Federation (SASF) consists of three main services: schema translation, schema matching, and schema mapping. Schema federation [14, 15] is the process of federating/integrating a number of distributed source schemas into a global schema (global view). We present the overall architecture of our proposed schema federation approach in Fig. 2, which is adopted for the distributed case from the general schema integration methodology outlined in [14]. We partition the general architecture into three modules: schema translation, schema matching, and schema mapping. In this section, we briefly define each module and its sub-modules, if any.

2.1 Schema Translation

Schema translation is the process of translating distributed source schemas into a *common data model*, e.g. SQL DDL or XML Schema. The common data model (CDM) indicates the choice of *one* data model to be used throughout integration. In the literature, a common data model is sometimes referred to as a Meta-Model, a Common Representation or a Common Model. In the rest of the paper, we use the common data model or common model interchangeably.

Translating distributed schemas into a common model is advantageous for two reasons: 1) to abstract the heterogeneity of the source schemas, e.g., relational schemas and XML schemas will be translated into in a common format; 2) to reduce the complexity of the implementation of the subsequent phases. The inputs to the schema translation module are the distributed source schemas, (shown as Step 1 in Fig. 2.) The outputs are the source schemas translated into the common data model, (see Step 2 in Fig. 2.)

The importance of this module stems from the preservation of the semantics of the underlying distributed source schemas. That is, the common data model should be able to capture the major semantics of each distributed source. Semantic preservation facilitates bi-directional transformation as it allows translation from the common model to model used in the source schemas and vice-versa.

2.2 Schema Matching

Schema matching is defined as the process of finding semantic correspondences between a set of schema elements. For example, if element 1 (e_1) in schema 1 (S_1) has the same semantics as element 2 (e_2) in schema 2 (S_2), then this module discovers this correspondence between $S_1.e_1$ and $S_2.e_2$. The role of this module is to find a reliable set of correspondences between the schemas to be integrated. The inputs to this module are the source schemas translated to the common data model (CDM), (i.e., the output of the schema translation module – see Step 2 in Fig. 2.) The output

is an integrated schema, which is the integration of all input source schemas after finding the semantic correspondences between their elements identified as Step 7 in Fig. 2.

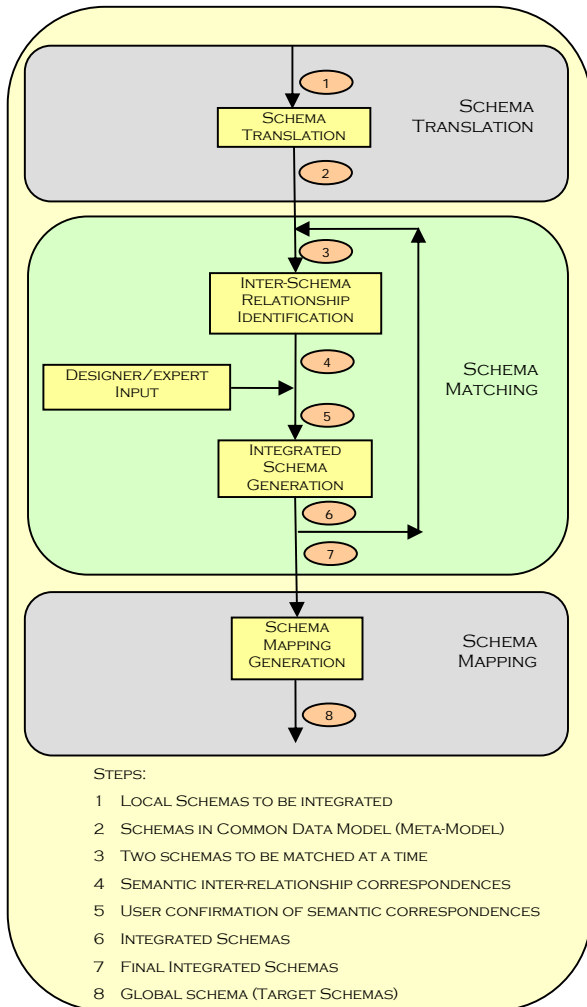


Figure 2. General Architecture of Schema Federation

To achieve the final goal, three sub-modules are iteratively inter-communicating: (a) inter-schema relationship identification; (b) designer/expert input; and (c) integrated schema generation. We briefly explain each sub-module.

2.2.1 Inter-schema Relationship Identification

This sub-module *examines* the semantics of input schemas and identifies relationships between corresponding elements. Its responsibility is identifying a set of match candidates between schema elements. A match candidate is defined as the discovery that there is a high probability that an element in one schema matches another element in a different schema. Only a set of *proposed* match candidates can be discovered; after the discovery user intervention is required to confirm the *accurate* match candidates. The inter-schema relationship identification sub-module accepts as input a pair of schemas at a time (Step 3 in Fig.

2) and outputs a set of relationships between schema elements (proposed match candidates – see Step 4 in Fig. 2.)

2.2.2 Designer/Expert Input

User intervention throughout the process of matching schemas is important for producing a reliable set of *confirmed* match candidates. The input to this sub-module is a set of schema elements with their proposed match candidates and their suspected relationships. Once a user confirms the accuracy of identified match candidates the sub-module outputs a confirmed set of accurate matches between the input schemas elements (Step 5 in Fig. 2.)

2.2.3 Integrated Schema Generation

This sub-module *integrates* the set of confirmed match candidates between a pair of schemas into an integrated schema. The resulting schema can then be used as input to subsequent iterations, if any. A set of confirmed match candidates are the input into this sub-module; the output is an integrated schema of the confirmed relationships between pairs of elements taken from different schemas (Step 6 in Fig. 2.)

2.3 Schema Mapping

Schema mapping is the module that maps the integrated schema elements to their original data sources. It takes as input an integrated schema (Step 7 in Fig. 2) and outputs a global schema of elements that are mapped to their original data sources (Step 8 in Fig. 2.) Global schema generation is important to handle user-proposed queries in terms of mapped elements.

3. SASF SERVICES

In this section we will provide more details of how each service in our proposed approach fulfils its task towards schema federation. The constituent services share a single *Schema Store*, which acts as their central communication point. The store is a relational data source that maintains the outputs as well as inputs for schema federation services. The user communicates with the services through a user portal.

3.1 Schema Translation Service

The main task of Schema Translation Service is to *translate* the schemas of *external data sources* into a *common representation* held in the Schema Store.

External Data Sources: could be any database management system. For example, a relational data source, or an XML data source could be external data sources. The feature of those external data sources is that they have their data *structured*. That is to say, a *schema* exists for a data source. The only assumption posed by the architecture is that such an exporting facility exists. This may be achieved by using wrappers.

Common Data Model (CDM): putting external data sources into a common model is advantageous as it abstracts over the heterogeneity of the data models of the sources. A common model also reduces the complexity of implementing subsequent phases. These then only have to cope with one format which is the common model format.

Translation Process: translating the schemas of external data sources into a meta-model is usually done manually, or through the aid of a translation tool. Therefore, the translation process could be manual or automatic.

The specific features of the SASF Schema Translation Service are outlined in Table 1.

Table 1. Features of the SASF Schema Translation Service

Feature	SASF Translation Service Features
External Data Sources	Relational Data Sources
Common Data Model	SQL DDL
Translation Process	Automatic

SASF automatically translates the schemas of external relational data sources into SQL DDL (Structured Query Language – Data Definition Language) [16] statements as its common data model. A user can select the set of schema elements that are to be translated into a common data model

3.1.1 Algorithm of SASF Schema Translation Service

Fig. 3 shows the algorithm used by the SASF Translation Service.

The set of source schemas is represented as a set A, where each element of A is a distinct source schema different from the rest of the set (Line 1.) In Line 2, we define B as a set of source schemas translated to the common data model. Each element in B corresponds to an element in A in terms of order, *i.e.*, b_1 is the common model of a_1 .

The translation function signature, shown in Line 3, accepts as input the set A, and outputs the set B. In Line 4, we iterate over the elements in A. We then translate each element of A to an element in B (Line 5.) We insert the translated element into B (Line 6) and, finally, return the set B (Line 7.) The returned translation is inserted into the SASF Schema Store.

```

1- Let A = {a1, a2, ..., an}; set of source schemas
   where a1 is the first source schema, and so on.
2- Let B = {b1, b2, ..., bn}; set of source schemas in
   CDM where b1 is a1 being put in a CDM, and so on.
3- output:B function:translateToCDM (input:A)
4- { for each ai in A
5-   { bi = translate ai -> CDM;
6-     add B <- bi; }
7- return B }
```

Figure 3. Algorithm of SASF Schema Translation

3.2 Schema Matching Service

The main task of Schema Matching Service is to find *semantic correspondences* between a *set of elements in one schema* with a *set of elements in another schema*. We assume that multiple matchers are used to accurately derive the semantic correspondences and increase the quality of the results. Each matcher has its own “logic” that identifies schematic relationships. *User input* is vital to confirm match candidates proposed by the *combined* set of matchers.

Match Granularity (element-level vs. structure-level): Matching a set of elements to another set of elements could be

element-based or structure-based. It is element-based if the matching process considers elements at an element-level for every match. Matching is structure-based if it compares specified relationships between elements within each source schema to infer match probability.

Match Cardinality (1:1, 1:n, n:1, n:m): Matching between elements of different schemas generates the following 4 cases:

1. **1:1** – an element from schema 1 matches one and only one element from schema 2.
2. **1:n** – an element from schema 1 matches more than one element from schema 2.
3. **n:1** – a set of elements in schema 1 match one and only one element from schema 2.
4. **n:m** – a set of elements in schema 1 matches a set of elements in schema 2. This case is considered as a set of 1:1 match cardinalities as in [5]. *n:m* matching cardinality is basically clear when writing a query where the joins in the query are set of 1:1 matching cardinalities.

Note that the Matching Service merely *identifies* the relationships between elements that will be *mapped* to the above four cases by the Mapping Service.

Semantic Correspondences: Deriving a set of match candidates between elements of different schemas requires the exploitation of schema-level information and instance-level information. By schema-level information we mean abstract schema information such as element/attribute name, type, domain of values, etc. While instance-level information means looking at actual data samples to infer matches.

Combination of Matchers: A matcher is a function that satisfies a single criterion for elements to be matched. We need a set of matchers to satisfy a set of criteria to match the chosen elements of different schemas. There are two ways to combine the results from matchers to form final result [5]:

1. **Hybrid:** where all criteria are examined on an element at the same moment.
2. **Composite:** where each criterion is examined on an element at a different time.

The final results of all matchers are then combined into a single result.

User Input: User intervention is vital to derive accurate match candidates. A user input could be used to either:

- to confirm some of the match candidates proposed by a system; or
- to enter an appropriate threshold to select suitable match candidates.

We consider a matching phase to have produced accurate and reliable match candidates if a user has confirmed the accuracy of the proposed set of match candidates.

The semantics of the SASF Schema Matching Service are summarized in Table 2.

Table 2. Features of the SASF Schema Matching Service

Features		Features of SASF Matching Service
Match Granularity		Element-level
Match Cardinality		1:1, 1:n, n:1, n:m
Semantic Correspondences Finding	Schema-level	Name- and constraint-based
	Instance-level	Text-oriented
Combination of Matchers		Composite
User Input		To confirm match candidates or set a threshold value

As described in Table 2, SASF performs element-level schema matching. It is capable of discovering 1:1, 1:n, n:1, and n:m match cardinalities. The semantics between elements are discovered based on both schema- and instance-level information. For schema-level information, we match elements based on their names using string-matching algorithms [17]. We also employ information about existing key constraints. For instance-level information, we extract some sample data for each element in the schema. We then attempt to match elements based on the extracted sample data by employing a set of matching algorithms over their values.

We have a library of available matchers a user can choose from (further explained in Section 3.2.1.) Some matchers implement a string-matching algorithm that examines a single criterion between metadata about the elements. Other matchers examine matches based-on sample data of the elements. Each matcher generates an estimate of the probability that its two inputs have correlated semantics. We use a composite approach to combine the results of the various matchers. Once results are combined, we ask a user to confirm the match candidates generated by SASF. If individual user decisions are not an option, then a user can enter a threshold value above which match candidates are confirmed by default. The threshold value ought to be carefully selected, by expert user, to avoid losing accurate match candidates or accepting candidates that are not accurate matches.

3.2.1 Architecture of SASF Schema Matching Service

The architecture of the SASF Schema Matching Service is depicted in Fig. 4 and consists of the following modules:

- **Matcher Library:** is a collection of matchers users can choose from for their applications. For example, we can have: attribute-name matcher, data-values distribution matcher, etc. Customized matchers for user applications can also be plugged-in.
- **Combinator Library:** is a collection of combinators users can choose from for their applications. For example, we can have: average combinator, max-value combinator, etc. Customized combinators for user applications can also be plugged-in.
- **Similarity Matrix:** a matrix of match probabilities that is built from executing a number of matchers over selected elements of selected schemas.
- **User Feedback:** is sought to confirm the match candidates discovered by the user-selected matchers. It also helps to generate accurate match candidates.

- **Integrated Schema:** is an integration of confirmed match candidates after each iteration. This is in a proper format for the subsequent mapping phase.

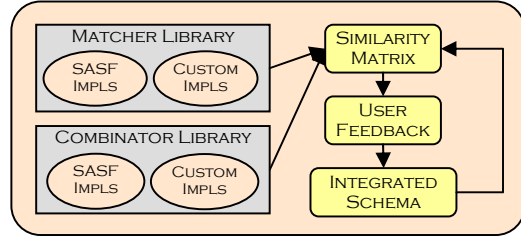


Figure 4. Architecture of SASF Schema Matching Service

The process works as follows: A user selects one or more matchers and only one combinator (also referred to as an aggregator) to start building the *similarity matrix*. A similarity matrix is where two schemas' elements are compared according to a certain criterion, and a similarity value for a pair of elements is derived accordingly. Within the similarity matrix, we have the results of the execution of all the selected matchers. From the similarity matrix, we combine the different matchers' results using the selected combinator and generate a combined result. Once the combination is in place, we seek user feedback to confirm the combined values of the match candidates. The user either confirms the acceptable matches or provides a reliability threshold. Finally the confirmed matches are integrated into a single schema. If there is more than one schema to match, this integrated schema is matched with the third schema and so on.

3.2.2 Algorithm of SASF Schema Matching Service

The algorithm of the matching phase is sketched in Fig. 5. Set B (Line 1) represents the set of source schemas already in CDM; set B is the output of the schema translation service. The resulting integration of the source schemas into one schema is represented by set C (Line 2.)

In Line 3, `userMatchers` indicates the set of selected user matchers to be used in matching the source schemas represented by set B . The chosen combinator is represented as `userCombinator` (Line 4.) `confirm` denotes a user's selected confirmation technique to confirm the proposed match candidates (Line 5.) The integration of confirmed match candidates takes place in function `integrate` (Line 6.)

The matching function signature, shown in Line 7, accepts as input the set B and outputs C . The body of the function consists of: matching two schemas at a time from set B (Line 10); combining the results of matched schemas (Line 15) confirming the results of combined matches (Line 16); and integrating confirmed matches (Line 17.) The algorithms for matching, combining, confirming, and integrating are not discussed due space limitations.

In more detail, in Line 8 we iterate over the set B . We consider two schemas from set B at a time and match those two schemas using a number of selected matchers being executed in parallel (Line 10.) For example, in Line 11 we execute a matcher (um_1) over schema b_i , and schema b_j . The result of the matching process is returned as `MatchResult1`. We then combine the

matchers' results and confirm the combined match results (Line 16); and finally integrate the confirmed match results (Line 17.) We set C to be the final IntegratedResult after iterating over all schemas in B and finally return C (Line 19.) The returned integrated result is inserted into the SASF Schema Store.

```

1- Let B = {b1, b2, ..., bn}; set of source
   schemas in CDM where bi is ai (a source
   schema) being put in a CDM, and so on.
2- Let C be an integrated schema of set B; C =
   integration of {b1, b2, ..., bn}.
3- Let userMatchers = {um1, um2, ..., umn}; set of
   user selected matchers from Matchers
   Library.
4- Let userCombinator be a selected user
   combinator from Combinators Library.
5- Let confirm be the user's choice of
   confirmation methods (manually by user or
   via a threshold).
6- Let integrate be an integration function of
   confirmed matches.

7- output:C function: matching (input:B)
8- { for each bi, bj in B
9-   {Consider bi, bj;
10-   In parallel, execute userMatchers over
      bi, bj
11-   { MatchResult1 = um1 (bi, bj);
12-     MatchResult2 = um2 (bi, bj);
13-     . . .
14-     MatchResultn = umn (bi, bj); }
15-   CombinedMatchResult =
      userCombinator (MatchResult1,
      MatchResult2, ..., MatchResultn);
16-   ConfirmedMatchResult =
      confirm (CombinedMatchResult);
17-   IntegratedResult =
      integrate (ConfirmedMatchResult); }
18- C = IntegratedResult;

```

Figure 5. Algorithm of SASF Schema Matching Service

3.3 Schema Mapping Service

The main task of the Schema Mapping Service is to *map* the integrated schema's elements produced by the Schema Matching service to their original data sources. It then builds a *Global Schema*, which can then be seamlessly queried by the user.

Mapping Global Schema: Mapping an integrated schema's elements to their original data sources could be accomplished in one of four ways:

1. **GAV** (Global-as-view): The global schema is constructed in terms of the source schemas elements [18].
2. **LAV** (Local-as-view): A mapping mechanism needs to be used to map the source schemas to the global schema [18].
3. **GLAV** (Global/Local-as-view): The relationships between the source schemas and the global schema are established by making use of both GAV and LAV approaches [18].

4. **BAV** (Both-as-view): BAV [19] provides a much richer integration framework as it allows reversible transformation between GAV and LAV. The BAV approach overcomes the disadvantages of the LAV and the GAV approaches as it allows incremental modification of the global schema when new sources are added (*e.g.*, in the AutoMed [20] data integration project.)

Global Schema (Input/Output): The Global view is used in some systems as an input, while in some other systems it is generated as a result of integrated schema generation.

The semantics of the SASF Schema Mapping Service are shown in Table 3.

Table 3. Features of SASF Schema Mapping Service

Features	Features of SASF Translation Service
Mapping Global View	GAV
Global View	As output

SASF uses the GAV approach to map integrated schema elements to their original data sources. It outputs the final Global Schema that a user can then query.

3.3.1 Algorithm of SASF Schema Mapping Service

The complete algorithm is presented in Fig. 6. The integrated schema is represented as C. Each element in C is of the following form: (e_i, m_{i-n}), which means that e_i has a number of matches represented by m_{i-n}. We assume that G is the global schema, as in (Line 2.)

The mapping function accepts as input C, and outputs G. We iterate over elements in C in preparation to build a global schema, which is of fewer elements that are not redundant (Line 4). We then compact the integrated schema to global schema based on one of four cases:

Case 1: e_i has no matches, (Line 5.) If an element has no matches at all, then we just add that element to the global schema, G, For example, (back to our running example of Fig. 1) `PO2.ShipTo` has no matches.

To consider cases 2, 3 and 4, an element should have matches in which case we iterate over an element's matches to check the following cases:

Case 2: e_i has number of matches EQUAL to the number of matches of all of its matchees, (Line 10.) If the element being considered has a number of matches, say x matches, and each match has a number of x matches, then we add that element to the global schema, G (Line 12.) We then iterate over that element matches and mark them as matched to avoid checking them again (Lines 13 and 14.)

In the running example of Fig. 1, `PO1.Customer.custNo` matches `PO2.ShipTo.custNo`. This represents a 1:1 matching cardinality.

Case 3: e_i is a subset of e_x, where e_x is composed of e_{i-n}, as in Line 15 in Fig. 6. If the element being considered has a match with more matches than the considered element (*i.e.*, the considered element has less matches than one of its matches), then we start an iteration over all matches (Lines 16 and 17.) When we come

across the element that has more matches than the considered element, we add the found element into G. We then iterate over the remaining matches and mark them as matched (Lines 20 to 21.) We also mark the considered element as matched. In Fig. 1's example,

$$\text{PO1.Customer.custStreet} \subset \text{PO2.ShipTo.address},$$

which presents a n:1 matching cardinality,

$$\text{PO1.Customer.custStreet, PO1.Customer.custCity, PO1.Customer.custZip} \subset \text{PO2.ShipTo.address}.$$

```

1- Let C be an integrated schema, where each
   element in C is of the form (ci, ei-n); ci an
   integrated schema element, ei-n the set of
   matches of ci.
2- Let G be Global Schema.
3- output:G function:mapping (input:C)
4- { for each ei in C
5-   /*case 1: ei has no matches */
6-   if (ei has no matches)
7-     Add G <- ei;
8-   else if (ei has x matches)
9-     { Loop over ei matches
10-      /*case 2: ei has number of matches
        EQUAL to the number of matches of all of
        its matchees */
11-      if (all ei matches has x matches)
12-        { Add G <- ei;
13-          Loop over ei matchees
14-          Mark each match as checked }
15-      /*case 3: ei is subset of ex, where ex
        is composition of ei-n*/
16-      else if (ei match has < x matches)
17-        { Loop over ei matches
18-          {if(eimatch has > x matches)
19-            {Add G <- eimatch;
20-              Loop over other ei matches
21-              Mark each match as checked}}
22-      /*case 4: ei is a composed element of
        ej-n */
23-      else if (ei match has > x matches)
24-        { Add G <- ei;
25-          Loop over other ei matches
26-          Mark each match as checked }}
27-   return G;

```

Figure 6. Algorithm of SASF Schema Mapping Service

Case 4: e_i is a composed element of e_{j-n}, (Line 22.) If the element being considered has more matches than one of its match's matches, then we add the considered element into G (Lines 23 and 24.) We then iterate over all of the current element's matches, and mark them as matched (Lines 25 and 26.)

In the running example, PO1.Customer.custName \supset PO2.ShipTo.custFname and PO2.ShipTo.custLname, presents a 1:n matching cardinality.

We finally return the global schema, when all global schema elements are found (Line 27.) The global schema is stored in the SASF Schema Store as well presented on the user portal to help the user write queries.

SASF is able to discover the following semantic correspondences [21] between elements:

- **Equivalence:** the elements of two schemas (A and B) are the same, *i.e.*, $A = B$ (1:1 matching cardinality)
- **Composition:** a schema's elements (A) are a subset of another schema's elements (B), *i.e.*, $A \supset B$ or $B \subset A$ (1:n or n:1 matching cardinalities.)
- **Intersection:** two schemas (A and B) intersect on a subset of their elements, *i.e.*, $A \cap B$
- **Disjointness:** two schemas (A and B) share no elements, *i.e.*, $A \neq B$

The schemas of different applications have different semantics between their elements. We support a reasonable range of semantic correspondences – more than can exist in one application as we have found in practice.

4. APPLICATION USE CASE

We demonstrate the usability of SASF in a real-world scenario of federating astronomical databases. Our goals in this demonstration are:

1. Show the applicability of our approach.
2. Present an end-to-end real-world astronomical example execution through all SASF phases: translation, matching, and mapping. We also show how querying takes place.
3. Provide a discussion on combining matchers evaluation and schema evolution

4.1 An Astronomical Application

Astronomers have tried to aid manual schema federation by defining Unified Content Descriptors (UCDs)¹, which are "semantic type" labels assigned to columns in database tables to indicate what physical quantity it stores, independent of the column name. Different astronomical data centers implement UCDs differently: in some cases they are queryable via the data center's SQL interface, whereas, in other cases, they are only provided in non-queryable documentation and/or used to tag columns in output files generated from querying the databases. The differences in the implementation of UCDs across data centers, makes astronomical datasets an ideal candidate for federation. The same semantical information is effectively captured under different schema elements and needs to be extracted and matched in a semi-automatic way. SASF is somewhat in position to semi-automate astronomers' manual schema federation which uses UCDs. SASF in its current implementation supports the federation of relational schemas. We choose to demonstrate SASF execution using the relational Sloan Digital Sky Survey for the Early Data Release (SDSS-EDR)² and the SuperCOSMOS Science Archive (SSA)³ servers. Astronomers are in need of federating the schemas of their distributed databases to allow seamless querying of the federated global view. In Table 4, we present some details of the servers. As

¹ <http://www.ivoa.net/Documents/latest/UCDlist.html>

² <http://skyserver.sdss.org>

³ <http://surveys.roe.ac.uk/ssa/>

shown, the SDSS-EDR server contains 37 schemas and the SSA server contains 15 schemas. We choose to use the term ‘schemas’ rather than ‘tables’ to emphasize our target in federating the distributed schemas.

4.2 An Astronomical End-to-End Real-World Example

With the help of astronomers, we were provided with a real-world query that needs to be evaluated over a federated global view. The SQL syntax of the query is shown in Figure 9. The textual description of the query is: *find galaxies for which there exist near-infrared detections in the SuperCOSMOS and Sloan Digital Sky Survey servers.*

Table 4. Application use case data sources used

Servers	Total schemas	Chosen schemas for experiment	Total attributes	Used attributes for query	Attributes % used
SDSS-EDR	37	photoobj	369	3	0.813
		field	409	2	0.489
SSA	15	crossneighboursedr	5	3	60
		source	57	3	5.263
		detection	46	2	4.348
		plate	152	2	1.316
Total			1038	15	1.445

We use OGSA-DAI [22, 23, 24] as our distributed database wrapper. SASF has been implemented in Java on top of the OGSA-DAI Grid implementation; the user portal has been implemented using Java Servlets [25].

SASF provides the user with selective control of what table schemas to federate, as well as what set of attributes within a table should be chosen to build the global view. With reference to Table 4, we can see that the user has chosen two tables out of the available 37 from the SDSS-EDR SQL server and four tables out of the available 15 from the SSA SQL server. In addition, the user has chosen to federate three attributes out of the 369 available in the photoobj table from SDSS-EDR. That is, only 0.813% of the total available attributes are needed to participate in building a global view. The rest of the chosen attributes from the remaining tables are shown in Table 4 and sketched in Figure 7, which is discussed below.

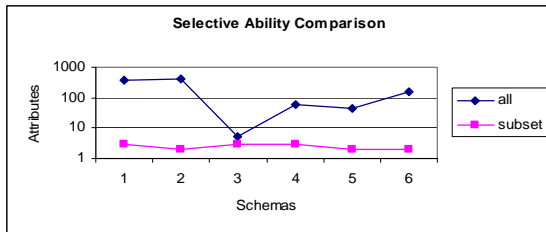


Figure 7. Selective Ability Comparison

Figure 7 shows the ability SASF provides for its users which can significantly save time on matching “unnecessary” attributes. It presents the big difference between matching all schema attributes to just a required subset of the schema attributes. The term *all* indicates all attributes in a schema, where *subset* indicates a user-selected subset used in the query.

Observe that the selective and control of SASF in terms of what to use in the federation process eliminate unnecessary translation and matching. Thus, performance is increased as the time that is “wasted” on un-necessary translation and matching is being factored out. This selectivity feature makes matching large schemas relatively simple without degrading performance. Note that existing approaches do not provide such flexibility: they will have to access and match all participating attributes from all schemas, even though these will never be used or queried by the user. In contrast, SASF only operates on what is for the correct use. The end-to-end example is explained throughout Step 1-9 below.

Step 1 – Schemas to Federate

We federate two schemas from SDSS-EDR and four schemas from SSA. The schema attributes, after user selection, are presented in Figure 8.

Table 1: skyservv3. dbo.photoobj	Table 2: skyservv3. dbo.field	Table 3: PSC.personalSSA. crossneighboursedr
type	fieldid	ssaid
objid	mjd_i	sdssid
fieldid		distancemins

Table 4: PSC.personalSSA. source	Table 5: PSC.personalSSA. detection	Table 6: PSC.personalSSA. plate
objidi	objid	plateid
meanclass	plateid	mjd
objid		
ra		
dec		

Figure 8. User selected schemas and attributes

Translation Phase

Step 2 – Translation to CDM

In SASF, we maintain a *Schema Store* where different federation phases store their output. It also acts as a central point for the federation phases to communicate. The translation phase causes the creation of three tables in the SASF Schema Store: `schema_index`, `schema_nodes`, and `schema_nodes_samples`.

In `schema_index` we maintain table references such as their original server URI, the database name, and the table name. On inserting such information in the `schema_index`, a unique id is generated which identifies a certain table (schema). In `schema_nodes`, we store user-selected attributes from each schema along with each attribute’s information such as name, data type, domain value, and key constraints. The unique id that is generated from `schema_index` is used with every entry in `schema_nodes` to identify the origin of each attribute. In `schema_nodes_samples` we maintain sample data instances for each attribute. Again, the unique id which is generated in `schema_index` is used with every entry.

Matching Phase

Throughout the matching phase, almost all tables are created in memory. The final integrated schema is stored in SASF Schema Store.

Step 3: We prepare the schemas that are translated to a common model with reference to their `schema_index` ids. For example, with reference to Fig. 8, the preparation of Tables 1 and 2 is as follows:

id	attribute	id	attribute
1	type	2	fieldid
1	objid	2	mjd_i
1	fieldid		

Step 4: For the first iteration, we consider a pair of schemas to be matched. For all sub-sequent iterations, we consider the integrated schema of the previous pair of schemas and a new schema. For example, the matching results after executing two matchers over the first pair of schemas are:

- matcher₁ result:

id1	schema1	id2	schema2	similarity_value
1	type	2	fieldid	0.0
1	objid	2	fieldid	0.4
1	fieldid	2	fieldid	1.0
...

- matcher₂ result:

id1	schema1	id2	schema2	similarity_value
1	type	2	fieldid	0.0
1	objid	2	fieldid	0.3
1	fieldid	2	fieldid	1.0
...

A matcher result constitutes the *similarity matrix*. The similarity value in cell (i, j) of the matrix indicates the degree of similarity between an element in schema_i and an element in schema_j. An element in each matcher's result is referenced to its original source. Executing matchers that examine various matching criteria between schema elements produce the similarity matrix.

Step 5: We combine the results of the two matchers used in Step 4. We use an average function, as a combinator, to generate the following combined values:

id1	schema1	id2	schema2	combined_value
1	type	2	fieldid	0.0
1	objid	2	fieldid	0.35
1	fieldid	2	fieldid	1.0
...

Note that when we combine the similarity values of different matchers, the combined value emphasizes the relationship between schema attributes. In our results we can see that `fieldid` in schema 1 matches `fieldid` in schema 2 with a combined value of 1.0. On the other hand, the combined value of `objid` in schema 1 and `fieldid` in schema 2 is decreased to indicate dissimilarity. In any case a user or a threshold function can confirm the accuracy of matching candidates.

Step 6: We confirm the combined results either via user feedback or a threshold function. The confirmed result contains the *accurate* match candidates between the pair of schemas:

id1	schema1	id2	schema2
1	type	NULL	NULL
1	objid	NULL	NULL
1	fieldid	2	fieldid
...

The results indicate an accurate and reliable set of match candidates between the matched schemas. For example, the confirmed results say: `type` in schema 1 has no matches; `fieldid` in schema 1 matches `fieldid` in schema 2; etc.

Step 7: We integrate the confirmed results in preparation for other iterations:

id1	attributes	id2	matches
1	type	NULL	NULL
1	objid	NULL	NULL
1	fieldid	2	fieldid
2	fieldid	1	fieldid
...

To generate the integration, we simply glue the two schemas together on one side (left), and present their match candidates on the other side (right).

Step 8: After going through the same process of matching, combining, and confirming the matches of the remaining schemas (schemas 3-6 in Fig. 8), we generate the final integrated schema which is the output of the schema matching phase after the completion of all iterations that have matched all schemas. The final results look like the results obtained in Step 7 with all schemas attributes on the left-hand side along with their matches on the right-hand side. This final integrated schema is stored in the SASF Schema Store, which will be retrieved by the Mapping Phase.

Mapping Phase

In the mapping phase, we map the attributes of the integrated schema to their original data sources. Note that we are using the GAV approach, which means that the elements are *already* mapped to their original data sources as they are used without any modification. In this final step we build the global schema by eliminating redundancies.

Step 9: We build the final global schema from the integrated schema. In this step, we emphasize the matching cardinalities between attributes. In our example, the only matching cardinalities that exist are: 1:1, and n:m.

id1	attributes	id2	matches
1	type	4	meanclass
1	objid	3	sdssid
1	fieldid	2	fieldid
2	mjd_i	6	mjd
3	ssaid	4	objid
3	distancemins	NULL	NULL
4	objidi	5	objid
4	ra	NULL	NULL
4	dec	NULL	NULL
5	plateid	6	plateid

Note that the “attributes” column contains global schema elements that a user can choose to query, where “matches” column shows each global schema element matches.

Query Execution: In SASF we do not provide a query engine. We actually use OGSA-DQP [26], which is a distributed query engine that uses OQL language to write the queries, or depend on application-specific query analyzer. For this example, we depend on the SSA query analyzer which is available on: SuperCOSMOS Science Archive at <http://surveys.roe.ac.uk/ssa/sql.html>. A typical astronomical query that was created from the global view is given in Figure 9.

```
SELECT s.ra,s.dec
FROM skyservv3.dbo.photoobj as p, skyservv3.dbo.field as f,
crossneighbourseidr as c, source as s, detection as d, plate as pl
WHERE d.objid=s.objid and s.meanclass=1 and p.type=3 and
c.ssid=s.objid and c.ssid=p.objid and
c.distancemin < 1.0/60.0 and p.fieldid=f.fieldid and
d.plateid=pl.plateid and (f.mjd_i between 47891 and 51543) and
(pl.mjd between 47891 and 51543)
```

Figure 9. A typical query example

The exact meaning of the attributes used in the query given in Figure 9 is not our focus. What needs to be noted, however, is that underlined attributes are discovered by SASF in the global view that was the result of Step 9 above. Providing the correspondences between the table attributes helps the user to express the necessary relational joins that relate attributes that are different at the schema level, but semantically similar. Finally, if this query executed on the query analyzer mentioned above, a total of 7560 rows are in the result set, when run on the “personal” versions of the SDSS-EDR and SSA: these are small, spatial subsets of the much larger full databases.

4.3 Discussion and Comparison to Existing Approaches

In comparison to existing matching systems (*e.g.*, COMA [6], SemInt [7], Cupid [8], TranScm [9], LSD [10], Artemis [11, 12], and DIKE [13]), their focus is only on the schema matching part. They do not: (1) consider handling user queries, (2) generate a global schema which is mapped to original data sources, and (3) operate in distributed environment. SASF, by being service-based rather than centralized, addresses all three points.

4.3.1 Combining Matchers Evaluation

Our approach of combining multiple matchers is inspired by COMA [6] and [27]. In [6], a detailed evaluation shows the advantage of combining multiple matchers. In SASF, users are given the option of selecting the set of matchers for their applications. In practice, different applications require different types of criteria to be examined. Therefore, we also support “plugging in” application-specific matchers for increased matching accuracy, *e.g.* to handle change of format or unit.

4.3.2 Global Schema Maintenance

We assume that schemas do not evolve. In case a source schema that is a constituent of a global schema has evolved, we need to re-generate the global schema. The re-generation is expensive for

large schemas, but with the selective features of SASF, re-generation is not as expensive. The re-generation is not expensive as the user is selecting only the attributes that s/he would like to query. Besides, a re-generation could be expensive especially if re-generation consists of redundant attributes. We hope to solve the redundancy in re-generation by introducing reusability of previous discovered matches; which is a future work.

In practice, it is rarely the case that a user writes queries accessing all the attributes of a table, especially in a data integration scenario. In our example, the user needed to use a total of seventeen attributes from a total of six schemas and 1038 attributes. The alternative, and the one that to the best of our knowledge has so far been followed in the literature, is accessing and matching all attributes from all tables of the schema. In large and highly distributed schemas the cost of doing something like that is prohibitive.

In addition, many global schemas can be created from different distributed sources. A user has the option of deleting an outdated global schema and building another in its place.

5. RELATED WORK

There are three axis in which existing work can be compared to our.

5.1 Schema Translation and Matching

In the **SemInt** [7], **TranScm** [9], **LSD** [10], **Artemis** [11, 12], and **DIKE** [13] schema matching systems only a 1:1 match cardinality is addressed. In reference to the example of Fig. 2, only *custNo* in PO1 can be discovered to match *custNo* in PO2. In **COMA** [6], the discovery of 1:1 and n:m is supported, while in **Cupid** [8], 1:1 and n:1 match cardinalities are discovered. With **SASF**, we support 1:1, 1:n, n:1, and n:m match cardinalities in a single framework.

Almost all pieces of work mentioned above do not support the examination of instance-level information to derive semantic correspondences, *except* **SemInt** [7] and **LSD** [10]. In **SASF**, we examine both schema-level and instance-level information to increase the accuracy of discovering matches between schemas.

5.2 Ontology Matching

In most ontology matching projects like **OLA** (OWL Lite Aligner) [28], **Anchor-PROMPT** [29], **COMA++** [30], and **SKAT** (Semantic Knowledge Articulation Tool) [31], only ontology (schema) level information is exploited for matching ontologies. Examining instance data is vital to increase the accuracy of discovering match candidates. **NOM** (Naïve Ontology Mapping) [32] and its successor **QOM** (Quick Ontology Mapping) [33] exploit instance-level information to perform matching. The **HICAL** system [34] also exploits data instances in the case of overlapping ontologies to infer mappings. Details about these ontology projects can be found in [35, 36]. With **SASF**, we exploit both schema and data instance information.

5.3 Schema Mapping

Generally, the output of schema matching is an integrated schema. Integrated schema elements need to be mapped to their

original data sources in preparation to handling user queries. **GridMiner** [37] builds a mediated global schema to be used for answering queries. The mediated global schema in **GridMiner** is hard-coded into the system configuration. That is to say, when a user would like to form a new global schema from new data sources, the user has to write code for all the mappings that form the new configuration of the mediated schema. In contrast, **SASF** gives the user the flexibility to create on-the-fly multiple global schemas from as many data sources as he/she desires.

Clio [38, 39], employs a correspondence engine in its architecture. The discovered semantic correspondences by **Clio** can be replaced by **SASF**'s schema matching service. The set of correspondences can then be fed to **Clio**'s mapping engine. On the other hand, the correspondences discovered by **Clio**'s correspondence engine can be fed to **SASF** Mapping service in preparation to user posed queries. This duality allows for the two frameworks to be used complementary.

6. CONCLUSION AND FUTURE WORK

We have described in this paper a service-based approach to schema federation that is consisted from three main services: schema translation, schema matching, and schema mapping. Our approach features are: (1) exploits schema and data information to drive semantic correspondences between schemas' elements; (2) operates over a wide range of match cardinalities (e.g. 1:1, 1:n, n:1, n:m); (3) functions in a distributed context in a scalable fashion; (4) provides a selection and control ability of databases' schemas and schemas' attributes that are to be federated which increases performance by not wasting time on "unnecessary" translation and matching.

The overall goal of **SASF** is to build a *generic solution to schema federation* in a distributed context and in an extensible fashion. More appropriately, **SASF** is a solution to distributed schema federation, where federation implies schema autonomy and structural heterogeneity.

We believe that the decomposition of schema federation services aids in simplifying their development, maintenance, and performance. For instance, different matching services can be used seamlessly in the architecture, provided they comply with the interfaces expected by the remaining two services. Besides, we use a customizable schema matching architecture inspired by **COMA** which allows plugging in new and application-specific matchers as well as newly developed matching algorithms can be exercised in this matching framework.

In the future, we aim to federate both XML schemas and flat files in addition to relational schemas, as our distributed database wrapper, **OGSA-DAI**, provides uniform access to heterogeneous data sources. So, we hope to reach a scenario where a user is querying a global view that is the federation of a number relational, XML, and files data sources in a distributed context in an extensible, scalable, and reliable fashion. Finally, we also hope to support schema evolution as in **Tess** project [40].

ACKNOWLEDGEMENTS

We would like to thank Bob Mann (Institute of Astronomy, University of Edinburgh, UK) and Dave Berry (National e-Science Center, University of Edinburgh, UK) for their invaluable

comments and feedback on this paper. Your efforts are highly appreciated.

7. REFERENCES

- [1] Haas, L.M., Lin, E.T, and Roth, M.A. Data integration through database federation. *IBM Systems Journal*, Dec 2002.
- [2] Foster, I. and Kesselman, C. The Grid 2: Blueprint for a new computing infrastructure. *Morgan Kaufmann*, 2004.
- [3] Chervenak, A., Foster, I., and Kesselman, C. et al. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. *In online proceedings of NetStore'99*, 1999.
- [4] Hoschek, W., Jaén-Martínez, F., Samar, A. et al. Data Management in an International Data Grid Project. *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, 77-90, 2000.
- [5] Rahm, E. and Bernstein, P. A. A survey of approaches to automatic schema matching. *VLDB*, 334-350, 2001.
- [6] Do, H. and Rahm, E. **COMA** - A system for flexible combination of schema matching approaches. *VLDB*, 2002.
- [7] Li, W. and Clifton, C. **Semint**: A system Prototype for Semantic Integration in Heterogeneous Databases. *ACM SIGMOD*, 1995.
- [8] Madhavan, J., Bernstein, P., and Rahm, E. Generic Schema Matching with **Cupid**. *VLDB*, 2001.
- [9] Milo, T. and Zohar, S. Using Schema Matching to Simplify Heterogeneous Data Translation. *VLDB*, 1998.
- [10] Doan, A., Domingos, P., and Halevy, A. Learning to Match the Schemas of Data Sources: A Multistrategy Approach. *Machine Learning Journal*, 50:3, 2003, 279-301.
- [11] Castano, S., Antonellis, V., and Vimercati, S. Global Viewing of Heterogeneous Data Sources. *IEEE Transactions on Knowledge and Data Engineering*, V. 13, 2001.
- [12] Bergamaschi, S., Castano, S., Beneventano, D. et al. Semantic Integration of Heterogeneous Information Sources. *Special Issue on Intelligent Information Integration, Data & Knowledge Engineering*, 36:1, 215-249, 2001.
- [13] Palopoli, L., Terracina, G., and Ursino, D. **DIKE**: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Software - Practice and Experience*, V. 33, 847-884, 2003.
- [14] Ram, S. and Ramesh, V. Schema Integration: Past, Present, and Future. Management of Heterogeneous and Autonomous Database Systems, *Morgan Kaufmann*, 119-155, 1999.
- [15] Bouguettaya, A., Benatallah, B., and Elmagarmid, A. An Overview of Multidatabase Systems: Past and Present. Management of Heterogeneous and Autonomous Database Systems, *Morgan Kaufmann*, 1-24, 1999.
- [16] Date, C. and Darwen, H. A Guide to the SQL Standard. *Addison-Wesley*, 1997.
- [17] Hall, P. and Dowling, G. Approximate String Matching. *Computing Surveys*, 12:4, 381-402, 1980.

- [18] Lenzerini, M. Data Integration: A Theoretical Perspective. *ACM PODS*, 2002.
- [19] McBrien, P. and Poulouvassilis, A. Data integration by bi-directional schema transformation rules. *In Proceedings of ICDE'03 IEEE*, 2003.
- [20] Boyd, M., Kittivoravitkul, S., and Lazanitis, C. et al. AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. *In Proceedings of CAiSE04*, Springer Verlag LNCS, V. 3084, 82-97, 2004.
- [21] Rizopoulos, N. Automatic discovery of semantic relationships between schema elements. *In Proceedings of the 6th ICEIS*, 2004.
- [22] Karasavvas, K., Antonioletti, M., and Atkinson, M. et al. Introduction to OGSA-DAI Services. *LNCS*, V. 3458, 1-12, May 2005.
- [23] Antonioletti, M., Atkinson, M., Baxter, R. et al. The design and implementation of Grid database services in OGSA-DAI. *Concurrency and Computation: Practice and Experience*, 17:2-4, 357-376, 2005.
- [24] Atkinson, M., Karasavvas, K., Antonioletti, M., Baxter, R. et al. A new Architecture for OGSA-DAI. *In proceedings of All Hands Meeting*, 2005.
- [25] Hunter, J. and Crawford, W. Java Servlet Programming. *O'Reilly*, 2001, 2nd edition.
- [26] Alpdemir, M. N., Mukherjee, A., Paton, N.W. et al. Service-based distributed querying on the grid. *In the Proceedings of the First International Conference on Service Oriented Computing*, Springer, 467-482, 2003.
- [27] Bernstein, P., Melnik, S., Petropoulos, M. et al. Industrial-strength schema matching. *SIGMOD Records*, 33:4, 38-43, 2004.
- [28] Euzenat, J. and Valtchev, P. Similarity-based ontology alignment in OWL-lite. *In Proceedings of ECAI*, 2004.
- [29] Noy, N. and Musen, M. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. *In Proceedings of IJCAI*, 2001.
- [30] Aumüller, D., Do, H., Massmann, S. et al. Schema and Ontology Matching with COMA++. *ACM SIGMOD*, 2005.
- [31] Mitra, P., Wiederhold, G., and Jannink, J. Semi-automatic Integration of Knowledge Sources. *In Proceedings of Fusion'99*, 1999.
- [32] Ehrig, M. and Staab, S. QOM: Quick ontology mapping. *In Proceedings of ISWC*, 2004.
- [33] Ehrig, M. and Sure, Y. Ontology mapping - an integrated approach. *In Proceedings of ESWS*, 76-91, 2004.
- [34] Ichise, R., Takeda, H., and Honiden, S. Rule Induction for Concept Hierarchy Alignment. *IJCAI*, 2001.
- [35] Shvaiko, P. and Euzenat, J. A Survey of Schema-based Matching Approaches. Technical Report, DIT-04-087, *University of Trento*, 2004.
- [36] Kalfoglou, Y. and Schorlemmer, M. Ontology mapping: the state of the art. *Knowledge Engineering Review*, 18:1, 1-31, 2003.
- [37] Wohrer, A. and Brezany, P. Mediators in the Architecture of Grid Information Systems. Masters Thesis, Institute for Software Science, *University of Vienna*, 2004.
- [38] Miller, R., Hass, L., and Hernandez, M. Schema Mapping as Query Discovery. *VLDB*, 2000.
- [39] Hernandez, M., Miller, R., Haas, L. et al. Clio: A Semi-Automatic Tool For Schema Mapping. *ACM SIGMOD*, 2001.
- [40] Lerner, B. A model for compound type changes encountered in schema evolution. *ACM TODS*, 25:1, 83-127, 2000.