An Ontology-based Mapping Repository for Dynamic and Customized Data Integration

(CISE Technical Report)

Xiao Li, Randy Chow
University of Florida
Department of Computer & Information Science & Engineering
Gainesville, FL 32611, USA
Email: {xl1, chow}@cise.ufl.edu

Abstract

There exist an increasing number of (semi-)structured information sources over the Web. Data integration techniques are often applied to increase the access efficiency of heterogeneous information sources. Data integration systems are built by combining a set of data sources for meeting specific user requirements. Due to various user needs, a growing trend of increasing variety of data integration system is noticeable even for the same application. In the academic community, much effort has been devoted to building a single data integration system. However, the existing approaches are impractical when the scale of data integration systems becomes large. This article proposes a key component of a total solution for holistically constructing a large number of customized data integration systems with the assistance of a considerable number of cooperative humans (a.k.a, a community). This component is an ontology-based mapping repository, called M-Ontology. Since each data integration system requires mappings for resolving semantic interoperability among information sources, M-Ontology aims at the efficient storage, management and discovery of mappings. M-Ontology is shared by all the data integration systems in the same application domain for mapping sharing and reuse. The internals of the mapping model are semantics-based for easy understanding and management by (even non-technical) community members. Following this model, a "human-friendly" mapping insertion algorithm is proposed for construction of a well-organized mapping ontology through incremental insertion of domain-specific mapping instances. Three types of human intervention strategies (i.e., validation, avoidance and prevention) are designed for improving the performance of construction algorithms. In addition, this article presents a mapping discovery algorithm to find the many-to-many complex mappings by leveraging the mapping ontology. The discovered mappings are recycled to further enrich the mapping ontology using the insertion algorithm. Finally, we conduct two sets of experiments on real-world data integration over the Web, and the results show that the approach is feasible and effective.

1 Introduction

WWW has been experiencing a tremendous growth in (semi-)structured information, in particular, the databases behind deep web [30, 10, 7]. *Data integration* (**DI**) techniques are often applied over these (semi-)structured sources to increase the efficiency of information access. A *DI system* achieves higher efficiency by providing users a uniform query interface, called *mediated schema*, to a set of integrated data sources. User queries through such a mediated schema are translated to

respective local queries and then the combined local query results are returned. The query translation is based on the *mappings* between the mediated schema and the query interfaces of underlying data sources.

In general, DI systems are designed for specific user requirements to integrate a set of data sources. The differences in user interests and preferences require different source selection and mediated schemas. For example, *kayak.com* and *zoomtra.com* are two DI systems for searching airfares. Although kayak.com is more popular in general, zoomtra.com provides cheaper air tickets to India most of the time. Thus, users who plan to travel to India by air prefer zoomtra.com to kayak.com. The difference between kayak.com and zoomtra.com is mainly due to their different selection of data sources. Thus, it is desirable that users have the capability of building their own DI systems with freedom to select their preferred data sources (i.e., *customizability*). In the academic community, much effort has been devoted to building a single DI system (e.g., COMA++[25] and MetaQuerier [11]), but these approaches are impractical when the scale of the DI systems becomes large due to increasing data sources and varying user needs.

Furthermore, user-specified source selection is not static but dynamic, and the query interfaces of data sources also are evolving. For example, users might want to insert a new data source into an existing DI system. The changes on source selection might affect the functionalities and the mediated schema of a DI system, and can even destroy the whole system. This means the existing mediated schema and mappings need to be updated to adapt to these changes. Therefore, DI systems should be allowed to dynamically evolve to adapt to the changing user needs and source interfaces (i.e., *dynamic re-configurability*). Most often, this important requirement is not fully considered in the existing DI research, especially in the context of multiple customized DI systems.

Consequently, construction and maintenance of a large number of customized DI systems become a very critical and practical problem. In this setting, it is impractical for a small set of experts to build such a large number of systems. Instead, this workload of a small set of expert builders should be distributed to a considerable number of cooperative members in an application community, which can include both users and domain experts. Mass collaboration (a.k.a. *community-based*) techniques [49, 32, 37, 33] are often used and shown promising in addressing the scalability and human involvement issues. Although mass collaboration was introduced to DI system construction by McCann et al. [32, 33], they do not consider the potential problems caused by the large scale of DI systems to be built. Among the problems, the most critical ones are *storage*, *management* and *discovery* of mappings, which are the cornerstones of a dynamic and customized DI system. These three problems are the focus of this paper.

For solving such problems, this paper proposes an ontology-based mapping repository, called *M-Ontology*. The previous work simply regard mapping repositories as persistent storage of mappings and their related information. However, in the context of dynamic and customized data integration, we believe that mapping repositories with richer semantics should play a more important role because of the necessity of building and maintaining a potential large scale of DI systems. The implementation of such a mapping repository has three major design considerations:

1) *Sharing of mappings*: The large scale of customized DI systems often indicates that repetitive mappings need to be stored and managed. Separate storage might cause a high degree of data redundancy and potential update anomalies. Thus, the mapping repository should not be designed for a single DI system, but be shared by a group of DI systems in the same application domain.

2) *Understanding of mappings*: The mapping repository is intended for use not only by ma-

chines but also by humans. Its construction benefits from joint contribution of a community, including domain experts and non-technical individuals. Based on the information these members receive, they can create new mappings, browse/update the existing mappings, and validate/modify the mappings created by machines. Thus, mappings should be organized in a way that is easy to understand for human users, particularly for non-expert users.

3) *Reuse of mappings*: Discovering the mappings between mediated schemas and data sources is a critical issue in the DI system construction and maintenance. We believe that reusing the existing mappings is one of the best (or maybe the only) ways to find the new mappings, especially those with complex expressions. As Krueger pointed out in [27]: "Abstraction is the essential feature in any reuse technique." Reusing mappings implies modeling mappings at some level of abstraction. In other words, mapping modeling decides the extent of mapping reuses.

Following the above three design requirements, M-Ontology views a mapping as an instance of a relation connecting two concepts. The concepts can be automatically extracted from the schemas of data sources that are connected by mappings. A relation in M-Ontology is a higher-level abstraction of mappings based on their semantics. Such a semantics-based mapping modeling is purposely designed for better understanding by humans. M-Ontology functions like a domain-specific knowledge base for mapping sharing and reuse to facilitate ease of machine automation. The specific contributions we make are as follows:

• Motivated by various and dynamic user needs, we present a research problem of dynamic customization of DI. A solution framework is provided for holistically constructing a large number of customized DI systems with mass collaboration. The core of this solution is an ontology-based mapping repository, which is the focus of this paper.

• We introduce a semantics-based modeling for mappings (rather than schemas) to make mapping management easier for community members who are responsible for identifying new mappings, correcting inaccurate mappings, and eliminating redundant mappings.

• We develop a "human-friendly" mapping insertion algorithm for incremental construction of M-Ontology. For improving the performance of the construction algorithms, we also develop three strategies for human interactions with the ontology, i.e., validation, avoidance and prevention.

• We propose a semi-automatic approach to mapping discovery for constructing and reconfiguring DI systems. Many-to-many mappings with expressions can be found by leveraging the mapping ontologies abstracted from the existing mappings.

The rest of this paper is organized as follows. Section 2 briefly discusses the application contexts of M-Ontology. Section 3 presents the prototypes and functionalities of the M-Ontology. Section 4 presents the implementation and the experimental analysis of M-Ontology. Section 5 discusses the related work on DI, mapping modeling and discovery. Finally, we conclude with a summary of future work in Section 6.

2 Problem Definition

Before digging into the prototypes and functionalities of the M-Ontology, we first present a research problem of dynamic and customized DI, and then provide a solution framework for this problem.

Data integration is a fundamental and classical problem [22] that aims at integrating data for multiple autonomous and heterogeneous data sources by generating a uniform view of these data. Users are freed from locating and querying each individual source. Each DI system consists of

a mediator and its underlying data sources (shown in Figure 1(a)). The role of a *mediator* is to access data sources and present the relevant data to users. Each data source has its own schema (called a *local schema*), such as a nested XML schema or a relational schema. These schemas are used to structure the stored data. There are two main challenges with respect to the interoperability between the mediator and data sources: query rewriting and result rewriting. *Query rewriting* (a.k.a. query reformulation or transformation) is to rewrite user queries in terms of mediated schemas into the queries expressed in local schemas. *Result rewriting* (a.k.a. answer rewriting) is to rewrite the retrieved data from source formats (i.e., local schemas) to the mediated ones.

In the Web era, the enormous volume of data available on the Web attracts a phenomenal number of users, and the continued rapid growth of users again leads to an even more explosive growth of contents on the Web. DI systems are often built for improving the efficiency of information access. Due to the scale of different user needs, a growing trend of increasing variety of DI systems is noticeable even for the same applications. However, the number of pre-constructed DI systems is still too few to meet various user needs. In addition, their updates are relatively slow, compared with the dynamism of user needs and source interfaces. Consequently, *dynamic customizability* is a highly desirable feature for DI system construction and maintenance. To meet such challenges, DI systems must be constructed and managed with two unique aspects:

- *Customizability*: the construction of a new DI system enables users to specify their needs and preferences.
- *Dynamic re-configurability*: the existing DI systems can evolve to adapt to the changing user needs and source interfaces.

As a consequence of dynamic customization, a large variety of DI systems will be built and need to be maintained dynamically (illustrated in Figure 1(b)). The set of DI systems can be formalized by a triple $\langle Set_G, Set_L, Set_M \rangle$, where

- Set_G is a set of global schemas (i.e., mediated schema), each of which corresponds to a DI system;

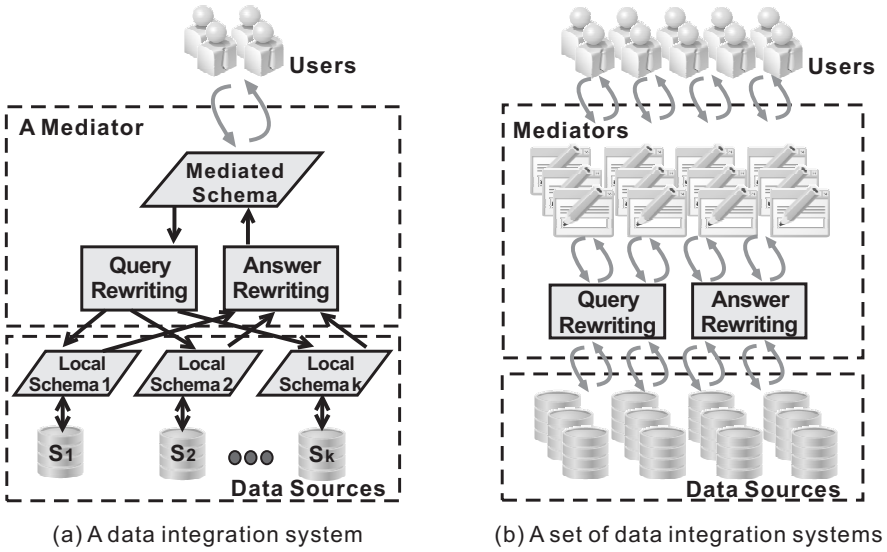


Figure 1: Data integration systems.

- Set_L is a set of local schemas, each of which corresponds to a data source;
- Set_M is a set of mappings, each of which corresponds to a logical formalism capturing transformation rules between a global schema and a local schema.

From the users viewpoints, the system interface and the selection of source databases are the most relevant customization. The number and the composition of source databases utilized by a DI system have a great impact on the perceived interface, and the contained functionalities and contents. For this reason, in this paper, we focus on data source selection for system customizability. DI systems can be built based on a user-specified set of source databases, and can be reconfigured with dynamical insertion and deletion of data sources, e.g., modifying global schemas due to some imported new sources.

Reducing the workload of human builders is the most crucial challenge to the implementation of such a feature. Facing this challenge, three strategies can be employed. a) Construction automation: automating the construction and maintenance. b) Mass collaboration: distributing the workloads to a considerable number of community members. c) Mass customization: grouping similar user needs to decrease the number of customized DI systems that need to be built. Mass customization is a marketing strategy for solving the drawbacks of mass production, and thus it is beyond the scope of this paper (full details are provided in [38]). Our proposed solution is based on the first two strategies, i.e., construction automation and mass collaboration:

1) Automating the system construction is one of the primary research issues in the DI community, but the number of DI systems to be built is always assumed to be very few (or just one). That is, they are not concerned with the issues related to how to build a large group of DI systems. In our solution, a large number of DI systems are to be built *holistically*, rather than individually. Its main benefit is to decrease human workloads by avoiding repetitive tasks and reusing shared components. Therefore, a mechanism is required for organizing and managing the unstructured shared components for better reuse.

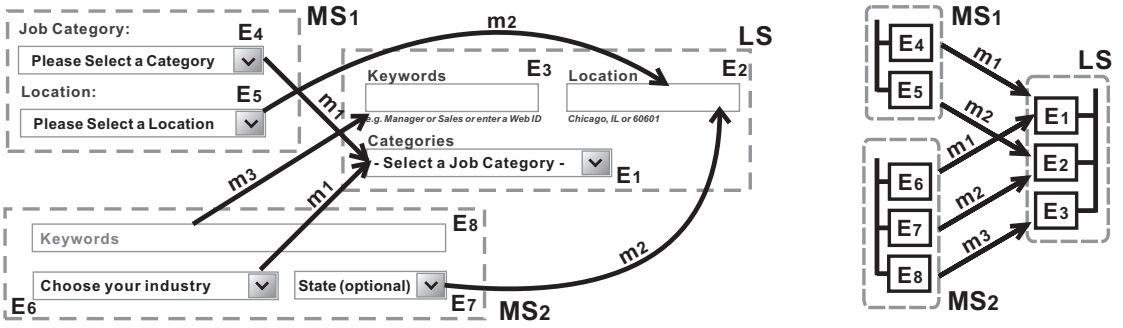
2) Mass collaboration has been introduced for DI systems [32], since the construction of them cannot be fully automated in the foreseeable future. The current research focuses on how to learn from community members to improve the construction accuracy. However, no solution has been provided about how these members can efficiently communicate and share their information. Generally, a centralized knowledge base functions well as a communication platform. The stored information can be inserted by one person and corrected by another. Consequently, it is necessary to propose an approach for building and maintaining such a knowledge base in a human-friendly manner.

Following the above discussion, this paper concentrates on a central issue in this holistic construction approach, i.e., the storage, management and discovery of mappings. Mappings play a critical role in any DI system. Both query rewriting and result rewriting rely on the mappings from mediated schemas to local schemas. In addition, the mappings among local schemas also facilitate the creation of mediated schemas [41]. Hence, mappings are one of the essential components for sharing and reuse in the dynamic and customized DI systems. Although much effort has been made in mapping discovery, finding complex mappings with expressions is still an open problem. Due to its AI-completeness, the human interaction is inevitable in mapping discovery. Community-driven approach is introduced [33] to distribute the heavy workload of discovery and maintenance to community members, but the mapping storage and management have not been paid enough attention.

The remaining part of this paper presents an ontology-based mapping repository (i.e., M-Ontology). It functions like a central mapping knowledge base shared by multiple DI systems, each of which has its own local mapping table. The table only stores the mappings required by this DI system. When new mappings are required, these DI systems can obtain them by employing the mapping discovery service provided by M-Ontology. In addition, M-Ontology can also be gradually enriched and updated by getting feedbacks from these dynamic and customized DI systems, e.g., the to-be-matched schemas and the verified mappings.

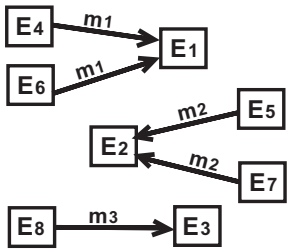
3 M-Ontology

The foundation of M-Ontology is the modeling of mappings. We develop a novel semantics-based approach to modeling mappings. This model can be employed to organize unstructured domain-specific mappings into a well-defined ontology. Such a modeling makes mappings easier to manage by humans, especially for those non-technical users. Many-to-many complex mappings can also be discovered more efficiently by leveraging the knowledge abstracted from the existing mappings in M-Ontology. After a motivating scenario, this section presents M-Ontology from three aspects: Section 3.2) how to model a mapping ontology; Section 3.3) how to construct a mapping ontology; Section 3.4) how to utilize an existing mapping ontology to discover new mappings.

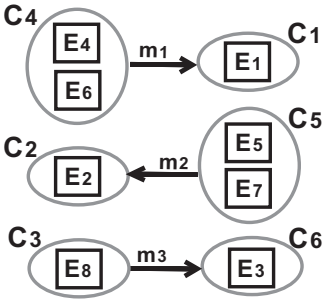


From	To	Rule
MS1.E4	LS.E1	m1
MS1.E5	LS.E2	m2
MS2.E6	LS.E1	m1
MS2.E7	LS.E2	m2
MS2.E8	LS.E3	m3

(c)



(d)



(e)

Figure 2: Three job search forms with the mappings.

3.1 Motivating Scenario

To motivate our proposed model, we use a simple scenario illustrated in Figure 2(a). Consider two simple DI systems MS_1 and MS_2 built for job seekers with different preferences. MS_1 and MS_2 provide users different query forms (i.e., mediated schemas) by integrating their own data sources. There exists an overlap between these two sets of sources, such as *LS*. The contents of *LS* can be accessed by a query form (called *local schema*). In a query form, the components (e.g., textbox and menus) and the related descriptive texts and potential instances are regarded as *schema elements*.

In order to integrate *LS* into MS_1 and MS_2, the system designers need to specify the mappings between schemas (called *schema-level mappings*) $MS_1 \rightarrow LS$ and $MS_2 \rightarrow LS$ for translating user queries. Since the schema-level mappings hide the mapping details, we decompose them into two sets of mappings between schema elements (called *element-level mappings*) so that reusing element-level mappings becomes possible. The directed edges in Figure 2(a) indicate the element-level mappings from MS_1 and MS_2 to *LS*.

Without considering the structural information, the schemas of *LS*, MS_1 and MS_2 can be transformed to a flat schema as shown in Figure 2(b). Each schema is represented as a dotted-line rectangle, in which a solid-line rectangle corresponds to a schema element. A mapping edge and its connected elements constitute an element-level mapping between schemas.

There are two common approaches to model element-level mappings. The first simply uses a table to model mappings as shown in Figure 2(c). Its columns are employed to represent the properties of mappings, such as source elements, target elements and mapping expressions. The second uses a mapping graph as shown in Figure 2(d) to connect all the mappings together. That is, each node corresponds to a source or target element and each edge represents a mapping. This graph-view structure is more explicit and straightforward for discovering new mappings through composition of mappings [31, 20, 6].

However, the above two approaches become almost impractical for managing a large volume of mappings, particularly when the mappings are shared and operated by a community. Thus, we introduce a semantics-based mapping modeling to makes mappings easier to manage. In the MS_1 and MS_2 DI systems, we observed that some mappings are highly similar. For example, two mappings $\{E_4, E_1, m_1\}$ and $\{E_6, E_1, m_1\}$ share the same target elements E_1 and mapping expressions m_1. From these similar mappings, some concepts can be identified by grouping the schema elements based on their semantics; the related relations can be transformed from the corresponding element-level mappings. These concepts and relations can be used to form a mapping ontology. For instance, shown in Figure 2(e), the original two pairs of mappings $\{E_4, E_1, m_1\}$ and $\{E_6, E_1, m_1\}$ can be represented by a single relation $\{C_4, C_1, m_1\}$ where the concepts C_4 and C_1 are formed respectively by $\{E_4, E_6\}$ and $\{E_1\}$. In this setting, mapping management becomes more intelligible to human users. Manipulations on individual mappings can be replaced by more straightforward and convenient operations on concepts and relations. Mapping redundancy and incorrectness are easier to detect by humans or even machines. Repetitive operations on semantics-equivalent mapping instances might also be avoided to eliminate the potential update anomalies. Furthermore, semantic modeling also benefits the discovery of new mappings by both humans and machines. Since it is a semantics-based abstraction of mappings, the reuse of previous mappings become more straightforward and effective.

Based on the above discussion, it is evident that semantic modeling is a better approach with

several advantages. The rest of this section presents our proposed mapping management system, M-Ontology, in details: 1) an abstraction-oriented internal structure; 2) incremental construction; 3) reuse-oriented mapping discovery.

3.2 Modeling

In each global or local schema, elements are the most fundamental building blocks. These schema elements are structured in a certain order and required to obey some constraints, such as domain constraints and referential constraints. Since this paper only focuses on the mapping modeling and management, for simplicity, all web schemas mentioned below are created from single-step HTML query forms. These schemas are flat (i.e. like single tables) without structural constraints. Our solution, however, can be easily extended to support more complex schemas with richer logical structures.

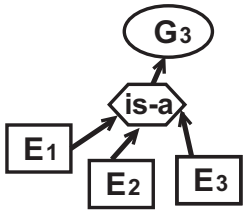
W3C HTML specification [1] defines a single-step form as "a section of a document containing normal content, markup, special elements called *controls* (checkboxes, radio buttons, menus, etc.), and labels on those controls". User requests are normally made by modifying the HTML controls, e.g., clicking radio buttons, entering text, etc. A control and its associated attributes and instances are regarded as "a whole", also referred to as a *schema element*.

In M-Ontology, element-level mappings are considered as the first-class entities. Mappings between two schemas can be decomposed into some separate element-level mappings; element-level mappings between two schemas can also be combined to generate a schema-level mapping [21]. In Section 3.1, we use a simple example to explain the basic idea of the proposed element-level mapping modeling. The example shown in Figure 2(e) comprises only several one-to-one simple mappings, but real-world mappings can be much more complex. Therefore, it is necessary for M-Ontology to support a semantically richer representation of element-level mappings. Generally, an element-level mapping can be defined as

$Mapping$ ($List_{E1}$, $List_{E2}$, Exp), where

- $List_{E1}$ and $List_{E2}$ are two ordered lists of schema elements, whose semantics are relevant to each other. The element number of a list can be one or greater than one, and thus mapping cardinality might be 1:1, 1:n or n:m ($n > 1$ and $m > 1$).
- Exp denotes a high-level declarative expression that specifies the transformation rules from $List_{E1}$ to $List_{E2}$. Expressions can be list-oriented functions (e.g. equivalence, concatenation, mathematic expressions) or other more complex statements (e.g. if-else, while-loop). In addition, the format of Exp should be both human-understandable (i.e., able to be easily modified by normal users), and machine-processable (i.e., can be automatically transformed to executable rules).

With this mapping representation, we can present a graph model for the proposed mapping ontology. This ontology is modeled as a directed acyclic graph where a *node* represents a *concept* with a set of associated *instances*; an *edge* corresponds to a *relationship* between two concepts. Both nodes and edges have some properties that describe their semantics and constraints. Based on the composition of concepts, all the nodes are classified into three different types: *Elementary*, *Generalization* and *Aggregation*. Edges also have three categories for different purposes: *Part-of*, *Is-a* and *Transformation*. Figure 3 and Figure 4 depict an ontology fragment about departure date in the domain of air ticket booking. The graph model will be explained in details with this example as follows.



E-Node	Descriptive Attribute
E_1	<name: outbound_departMonth>, <id: monthCalendar>, <class:month>,<text:Depart>
E_2	<name: AIR_frommonth>, <id: dm>, <class: formfield>
E_3	<name: dateLeavingMonth>, <title: Select depart month>, <id: AIR_frommonth>, <text:Depart>

Figure 3: A fragment of a mapping ontology (E-Nodes and G-Nodes).

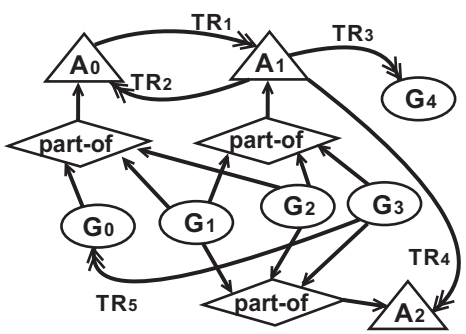
3.2.1 Concepts and Instances

Elementary Nodes: An Elementary node (called an *E-Node*, shown as a rectangle in Figure 3) represents the concept of a schema element using a tuple $\langle$*URI*, *D-Attributes*, *Instances*, *I-Constraints*$\rangle$, where *URI* identifies the location of the schema, *D-Attributes* refer to the descriptive attributes associated with this element (e.g., name, id, value, class, domain), and *Instances* and *I-Constraints* are the known data instances and the instance constraints, such as the instance type and domain.

In M-Ontology, E-Nodes are the most fundamental concept units for constructing high-level concepts, i.e., the concepts represented by Generalization nodes and Aggregation nodes. The introduction of E-Nodes is to eliminate *language heterogeneity*. Language heterogeneity refers to the difference in the languages used to represent schemas, such as XHTML or HTML. Each schema element can be transformed to an E-Node without semantic loss. The semantics of an E-Node is reflected by the D-attributes of the corresponding schema element. Schema designers always attempt to use a schema element for conveying a certain concept. Intuitively, such a concept can be instantiated by the data instances under the corresponding schema element. For example, the E-Node E_1 in Figure 3 corresponds to a schema element whose D-Attributes are $\langle$name: outbound_departMonth$\rangle$, $\langle$id: monthCalendar$\rangle$, $\langle$class: month$\rangle$ and $\langle$text: Depart$\rangle$, and instances are represented in a number from "1" to "12". It can be correctly inferred from its D-Attributes and instances that this element indicates a concept about departure months.

Generalization Nodes: A Generalization node (called a *G-Node*, shown as a round in Figure 3 and 4) represents a concept by a tuple $\langle Set_{E-Node}$, *D-Attributes*, *D-Labels*, *Instances*, *I-Constraints*$\rangle$, where Set_{E-Node} refers to an unordered set of E-Nodes as the internal constitution of this G-Node. *D-Attributes* and *D-Labels* are used to describe the semantics of the G-Node, and *Instances* denote a potential instance set Set_{gn}^{INS} under the constraints of *I-Constraints*.

The semantics of a G-Node is equivalent to that of its inclusive E-Nodes. The D-Attributes and D-Labels of a G-Node *gn* are two sets of word tuples Set_{gn}^{DA} and Set_{gn}^{DL}. Each word tuple (w_i, wf_i) consists of a distinct word w_i and the corresponding appearance frequencies wf_i in *gn*. Set_{gn}^{DA} is generated from the normalized D-Attributes of the included E-Nodes whose insertion is already verified by humans. The D-Labels can be automatically derived from frequent appearance of D-Attributes or manually input by humans. The detailed algorithms for finding the D-Attributes and D-Labels of a G-Node are discussed in Section 3.3. In addition, the Instances and I-Constraints of a G-Node are directly obtained from the inclusive E-Nodes. For example, G_3 has the same instance set and I-Constraints as E_1, E_2 and E_3.



G-Node	Instance Format
G_0	month:English
G_1	year:num
G_2	day:num
G_3	month:num
G_4	month-day(month:num)

A-Node	Instance Format
A_0	month-day-year(month:English)
A_1	month-day-year(month:num)
A_2	year-month-day(month:num)

Figure 4: A fragment of a mapping ontology (G-Nodes and A-Nodes).

The purpose of G-Nodes is for *syntactic heterogeneity*. The E-Nodes in Set_{E-Node} share the same semantics and instance formats but their syntactic representations are different. That is, the instances of these E-Nodes share the same format with the identical semantics if they represent the same object, but the D-Attributes of these E-Nodes could be totally different. To take an example, in Figure 3, the concept of G-Node G_3 is generalized from three verified E-Nodes. Although these E-Nodes have equivalent semantics and instances format, their D-Attributes are different in the attributes' numbers, types and values. It is not necessary to impose any syntactic transformation when these E-Nodes exchange their instances.

Aggregation Nodes: An Aggregation node (called an *A-Node*, shown as a triangle in Figure 4) represents a concept by a triple $\langle List_{G-Nodes}, D\text{-}Labels, Instances\rangle$, where $List_{G-Nodes}$ denotes an ordered list of G-Nodes, *D-Labels* and *Instances* respectively refer to its descriptive attributes and possible instances.

For the purpose of representing many-to-many mappings, A-Nodes are generated by aggregating an ordered list of G-Nodes. That is, an A-Node saves the order of its inclusive concepts. Its Instances and D-Labels can be fetched from its inclusive nodes, and then combined in such an order. For example, two A-Nodes A_1 and A_2 in Figure 4 represent a concept "departure dates" by aggregating the same three G-Nodes G_1, G_2 and G_3. These G-Nodes respectively represent a calendar year, day and month by numbers and their formats are shown in Figure 4. Although their semantics are equivalent, their instance formats are different due to different element orders. A_1 uses middle endian forms "month-day-year" (i.e., starting with the month). A_2 uses big endian forms "year-month-day" (i.e., starting with the year). Note that aggregation cannot be directly applied on E-Nodes in M-Ontology. In other words, A-Nodes only connect with G-Nodes. This requirement enforces E-Nodes to be encapsulated into G-Nodes. It indicates more concepts can be generated by clustering schema elements.

3.2.2 Relationships

Is-a Edges and Part-of Edges: Is-a edges and Part-of edges (shown as single-arrow edges separately in Figure 3 and 4) represent hierarchical relations in mapping ontologies. They respectively correspond to two distinct forms of concept abstraction: *generalization* and *aggregation*. Both attempt to conceptualize E-Nodes. The main benefit of generalization is to share semantic information among schema elements: 1) In a G-Node, all the E-Nodes have the same semantics and instance formats, their descriptive labels (such as name, id, class) and instances can be exchanged

for understanding their semantics more precisely. 2) In a G-Node, all the inclusive E-Nodes share the mappings. That is, as long as an E-Node is assigned to a G-Node, all the pre-configured mappings associated with this G-Node are also applied on this E-Node. In addition, the introduction of aggregation is necessary for modeling n:m complex mappings which include multiple schema elements in a certain order.

Transformation Edges: A Transformation edge (called *T-Edge*, shown as a directed edge with double arrows in Figure 4) connecting two nodes represents a transformation relation. Its mapping expression is stored as an attribute of the related T-Edge. T-Edges cannot connect E-Nodes.

Combining with the G-Nodes and A-Nodes, T-Edges aims to address three different types of heterogeneity between two semantic-overlap concepts. They are *instance heterogeneity*, *structural heterogeneity* and *semantic heterogeneity*: 1) Instance heterogeneity refers to the variety in instance formats for the same entity. For example, both G_0 and G_3 represent the departure months, but they respectively use different formats, i.e., number and English. Thus, a T-Edge TR_5 might be created for transforming instances from G_3 to G_0, if necessary. 2) Structural heterogeneity often occurs when the aggregation orders of concepts are different. For example, A_1 and A_2 represent a date in a different order, i.e., " month-day-year" and "year-month-day". A T-Edge TR_4 is used to link A_1 and A_2. 3) Semantic heterogeneity represents the differences in the concept coverage and granularity. For instance, TR_3 is used to connect two concepts A_1 and G_4 with different coverage. Different from A_1, G_4 only represents the departure date without "year".

In addition to the above explicit relations, there exist some implicit transformation relations stored in G-Nodes. In each G-Node, E-Nodes have an equivalence relation between each other. As mentioned above, the instances of these E-Nodes are shared among them without any change. This kind of relation is common in real applications.

3.2.3 Metadata

M-Ontology stores not only mappings but also their related context information, i.e., the metadata of mappings. The metadata is mainly used to facilitate community cooperation and enhance system effectiveness and robustness. The metadata includes who, what, where, when and how aspects associated with the whole lifecycle of mappings, including their creation, verification, modification and deletion. To store the metadata, the proposed mapping ontologies assign the following metadata on each node and edge:

Creation: Nodes/edges can be created manually or automatically. If humans are the creators, M-Ontology will record the user names and the creation time in the related nodes/edges. If they are discovered by machines, the algorithm names, the credibility values (a.k.a. similarity values) and the creation time will be stored. In addition, the creating process is also recorded. When a mapping mp is inserted, all the relative nodes and edges are annotated by mp.

Verification: Verification checks for accuracy and completeness of nodes/edges. In the current implementation, each node/edge is accompanied by a set of attribute triples: $\langle VS, Name, Time \rangle$, where VS refers to the validation status of this node/edge, $Name$ and $Time$ respectively represent the user names and time points. VS has three types of values: UNKNOWN, INCORRECT, and CORRECT. Human users are able to view and verify all the components of mapping ontologies manually. Although the current M-Ontology cannot automatically evaluate the correctness of mappings, it is not difficult to support such functionality if machines have the capability of detecting the usage of mappings.

Modification and Deletion: Mappings are not static. Their nodes/edges need be updated or deleted. Since they might also be maliciously/accidentally revised or removed, some version mechanisms [26, 12] should be built into M-Ontology. Although the current implementation of M-Ontology does not include such functionalities, it does record the related information, such as, who and when the nodes/edges are modified or deleted.
Annotation: Not all the mappings can be automatically discovered if there exist some hidden contexts. For example, the scale-factor of instances can be set to any integer value, such as 1, 1000 and 1000000. The money amounts can be reported in different currencies, such as Dollar, Euro and Yuan. Normally, such contexts are almost impossible to reason by machines from the individual E-Nodes. Thus, M-Ontology enables users to annotate the nodes and edges with additional information for understanding by others.

Digging a little deeper into the structure of the mapping ontology, we can see two important considerations:

1) Traditional domain ontology construction aims at semantic richness [45] that is important for performing complex reasoning. However, richer semantics requires more human involvement in ontology construction. This contradicts with one of our original goals, that is, the best-effort reduction of human efforts during the construction and maintenance of DI systems. Therefore, the contents that M-Ontology contains and maintains are restricted to a *sufficiently rich* level. It only includes the mapping-related information, since it is used only for mapping storage and reutilization. The following section 3.3 discusses how to construct such an ontology.

2) Most traditional mapping repositories [35, 37, 49] store many-to-many mapping instances directly without organizing them based on their semantics. In contrast, M-Ontology attempts to identify and store the hidden concepts from instances in a straightforward approach, and at the same time, address five types of heterogeneity: language heterogeneity, syntactic heterogeneity, instance heterogeneity, structural heterogeneity and semantic heterogeneity. It aims at better mapping management and reutilization not only by machines but also by normal users. Section 3.4 explains how to reuse mappings in such an ontology.

3.3 Construction

M-Ontology is an ontology-based mapping repository. Mappings are the main information sources for constructing such a domain-based mapping ontology. The construction of the ontology is through incremental insertions of element-level mappings. The incremental approach is very critical for building a long running and shared mapping repository. Since both users and data sources are autonomous units, source selection and local schemas are dynamic. New mappings continue to be inserted as new data sources are inserted or old ones evolve. M-Ontology requires periodic update. It is more effective to update the existing ontology incrementally rather than re-learn the ontology from scratch, especially when the ontology has been manually corrected.

Like schema matching in data integration, ontology construction is also a labor-intensive, time-consuming and error-prone problem [39, 45]. In order to reduce human efforts involved in the construction, M-Ontology provides a semi-automatic algorithm for inserting mappings. Each insertion involves three main tasks: a) *concept identification*, including E-Nodes, G-Nodes and A-Nodes; b) *relationship identification*, including Is-a edges, Part-of edges and T-Edges; c) *metadata storage*, including all the metadata of concepts and relationships. Generally, it is straightforward to record the metadata at the same time as the edges and nodes are manipulated during the whole life cycle

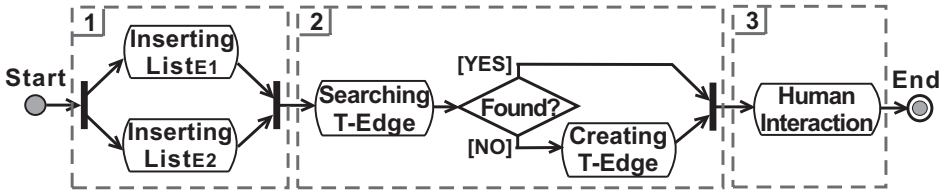


Figure 5: The flowchart of ontology construction.

of ontology. Hence, the following contents in this subsection mainly focus on how to deal with the first two tasks when inserting a mapping (*mp*). As defined in Section 3.2, an element-level mapping is represented as *Mapping* ($List_{E1}$, $List_{E2}$, *Exp*). M-Ontology splits the semi-automatic mapping insertion into three sequential phases as illustrated in Figure 5: 1) inserting the element lists, $List_{E1}$ and $List_{E2}$; 2) inserting the expression *Exp*; 3) validating the insertion by humans.

• **Element Insertion** (phase 1): The purpose of this phase is to obtain E-Nodes, G-Nodes, A-Nodes, Is-a edges and Part-of edges when inserting the element lists. The element lists, $List_{E1}$ and $List_{E2}$, are inserted separately. Algorithm 1 illustrates the whole process of inserting an element list $list_E$ into a mapping ontology *mo*. Besides updating the ontology *mo*, the algorithm returns an *abstract node cn* corresponding to $list_E$ for the next phase (Note that for the simplicity of description, an *abstract node* is introduced hereafter to represent a node that might be a G-Node/A-Node.)

Algorithm 1: Insert an element list into a mapping ontology.

Data: A mapping ontology *mo*, an element list $list_E$
Result: $mo \leftarrow mo \cup \{list_E\}$, a returned abstract node *cn* corresponding to $list_E$

```
1  an = new ANode();
2  foreach Schema Element e in list_E do
3      if e ∈ mo then
4          gn = e.GetENode().GetGNode();
5      else
6          en = new ENode(e);
7          gn = SearchGNode(mo, en);
8          if gn = ∅ then gn = new GNode(); end if
9          gn.InsertENode(en);
10     end if
11     if List_E.GetElementNum()>1 then an.InsertGNode(gn); end if
12 end foreach
13 anInMO = SearchANode(mo, an);
14 if anInMO ≠ ∅ then cn = anInMO;
15 else if an.GetNodeNum() ≠ 0 then cn = an;
16 else cn = gn;
17 end if
18 mo.InsertCNode(cn);
19 return cn;
```

In the algorithm, each schema element *e* in $list_E$ is checked if it already exists in the ontology *mo* (line 2-3). If true, its G-Node is (a part of) the target abstract node (line 4). If false, it is encapsulated into a new E-Node *en* whose initial validation status is UNKNOWN (line 6). Then, line 7 seeks the most suitable G-Node *gn* in *mo* for inserting *en*. If not found, *en* is inserted into a newly created G-Node; otherwise, *en* is directly inserted into the found *gn* (line 8-9). If $list_E$ only contains a single element, *gn* is the abstract node *cn* that will be returned (line 8,15,17); or else each of the found G-Nodes *gn* is inserted into a newly created A-Node *an* in the same order with the original one of schema elements in $list_E$ (line 1,11). Then, line 13 is to go over *mo* for finding out an A-Node *anInMO* with the same contents like *an*. If such an A-Node exists, *anInMO* is the node which need to be returned; or else *an* will be returned (line 14-15,19). Before returning *cn*, the insertion effects on *mo* are executed in line 18 with the relative metadata recorded.

The core of Algorithm 1 is two search problems (line 13,7): SearchANode(*mo*, *an*) and SearchGNode(*mo*, *en*):

a) **SearchANode** can be implemented by directly evaluating the equivalence between *an* and the A-Nodes in *mo*. Two A-Nodes are equivalent only when they include the same G-Nodes in the same order.

b) **SearchGNode** is implemented by comparing the semantic similarities of all the G-Nodes in *mo* with the E-Node *en*. The target G-Node *gn* is the one with the highest similarity value sim_{gn-en} that must exceed a given threshold. If no G-Node is eligible, SearchGNode returns a null value to the caller. The similarity sim_{gn-en} between *gn* and *en* is determined by two steps:
Step 1: *Preprocessing*. The D-Attributes and Instances of *en* are collected and then normalized by NLP (natural language processing) techniques: tokenization, stop-word removal and stemming. The resulting texts are considered as two bag of words, that is, two unordered sets of words, Set_{en}^{DA} and Set_{en}^{INS}.
Step 2: *Calculating*. Resulting from the previous verified insertion of E-Nodes, the G-Node *gn* has an instance set Set_{gn}^{INS} and two word sets Set_{gn}^{DL} and Set_{gn}^{DA}, respectively, corresponding to its D-Labels and D-Attributes. If *gn* and *en* are constraint-compatible, their semantic similarity is calculated by,

$$sim_{gn-en} = w_1 \times sim_1(Set_{gn}^{DL}, Set_{en}^{DA}) + w_2 \times sim_2(Set_{gn}^{DA}, Set_{en}^{DA}) + w_3 \times sim_3(Set_{gn}^{INS}, Set_{en}^{INS})$$

where w_1, w_2 and w_3 are three weights in [0,1] such that $\sum_{i=1}^{3} w_i = 1$. The similarity functions sim_1, sim_2 and sim_3 are implemented by a hybrid matcher that combines several linguistics-based matchers, such as WordNet-synonyms distances, 3-gram distances and Jaccard distances. Many algorithms for schema/ontology matching have been proposed (see the surveys[42, 18]). Most of them can be easily employed in this phase. As this is not the target of this paper, the details are not explained in this paper.

• **Expression Insertion** (phase 2): The target of this phase is to identify and create T-Edges. The inputs of phase 2 include an expression *Exp*, and two abstract nodes cn_1 and cn_2 that respectively correspond to $List_{E1}$ and $List_{E2}$. Phase 2 checks if there exists a T-Edge with the same expression as *Exp* from cn_1 to cn_2. If it does not exist, a T-Edge containing *Exp* is created with a validation status of UNKNOWN; otherwise, this phase is finished.

• **Human Interaction** (phase 3): If M-Ontology is only used to store mappings, the whole process is fully automatic. However, M-Ontology also supports reuse-oriented mapping discovery. Since mapping discovery is known as an AI-complete problem[22, 34], human involvement cannot be

avoided in the construction of M-Ontology, during which many new mappings are discovered. Humans are responsible for the validation and correction of nodes and edges. For improving the performance of M-Ontology, three intervention strategies are designed, as follows,

1. **Validation**. For alleviating possible redundancy and error propagation, only the nodes and edges that have been validated by humans can affect the future mapping insertion. VerifyENode and VerifyTEdge are two of the main verification functions.

   a) *VerifyENode* (Verification of E-Nodes): After an E-Node *en* is verified, its two normalized bags of words, Set_{en}^{DA} and Set_{en}^{INS}, start to affect the corresponding G-Node *gn*. The D-Attributes Set_{gn}^{DA} and Instances Set_{gn}^{INS} of *gn* need to be updated to combine the words occurring in the E-Node. The machine-found D-Labels Set_{gn}^{DL} might also evolve by selecting the D-Attributes with the top-k frequencies (e.g., k=5). In case of updates on the G-Node, the relative A-Node is also updated if necessary.

   b) *VerifyTEdge* (Verification of T-Edges): The verification of a new T-Edge affects not only its originally associated mapping, but also all the indirectly connected E-Nodes inserted in the past and future. Such an activity indicates $n \times m$ mappings are validated and stored, where n and m are respectively the value of possible E-Node combination of the connected abstract nodes. More mappings can be found from VerifyTEdge, if mapping composition [31, 20, 6] is supported.

2. **Avoidance**. Validation does not always guarantee the correctness and non-redundancy of the mapping ontology, especially in the context of mass collaboration. Thus, some automatic detectors are proposed based on the structural properties of M-Ontology. Alarms are sent to human users for assistance if one of the following cases happen: (i) two or more verified T-Edges are found from one abstract node to another; (ii) a new abstract node is created for mapping insertion; (iii) the found cn_1 and cn_2 are not identical if the original mapping expression is "equivalence".

3. **Prevention**. Even if the current contents of M-Ontology are correct and non-redundant, human users are also encouraged to involve in the enrichment of mapping ontologies for improving the performance of construction. Manually modifying the attributes of abstract nodes (such as D-Labels and Instances) can improve the precision and recall of construction algorithm, since our algorithm is based on these attributes to insert E-Nodes and expressions. In addition, the comments on nodes/edges are welcomed from the creators/modifiers. The information stored in metadata (such as annotation) is critical for understanding of others.

Following the above mapping insertion algorithm, M-Ontology is constructed and enriched as more and more mappings are inserted. The main functionalities of M-Ontology are mapping storage, management and discovery. First, the basic mapping retrieval can be achieved in a fully automatic way. Even if the algorithm does not find the correct abstract nodes for insertion, the inserted mappings can also be accurately retrieved based on their URI and related metadata. Second, the management of a large volume of mappings becomes easier for human beings: (i) the mapping ontology can be represented as a graph, and thus, the operations in mapping management can be transformed to several simple operations on graphs; (ii) the semantic modeling makes sense of unorganized mapping information so as to enhance the understanding by humans; (iii) the representative object-based clustering algorithm is more straightforward to human users since they can

easily improve the performance by manually changing the representatives, e.g., adding or removing several D-Lables and D-Attributes. Third, based on the semantic modeling and construction mechanism, a reuse-oriented algorithm for mapping discovery can be efficiently implemented as discussed in the next section.

3.4 Mapping Discovery

Mapping discovery is a critical operation in the construction and maintenance of dynamic and customized DI systems. It is used to discover the mappings between mediated schemas and local schemas in two different scenarios: i) building a new DI system with a mediated schema and a set of local schemas; ii) reconfiguring an existing DI system with a modified/existing mediated schema and an updated set of local schemas.

Our approach is to match two schemas through an intermediary mapping ontology. Its core is the reuse of previous mappings. Different from the previous work [42, 47, 17, 13], our strategy is not to directly reuse individual mappings, but to reuse their previous concept classification. The main idea is based on a feature of M-Ontology: when an E-Node is inserted into an existing G-Node, all the mappings associated with this G-Node will be automatically assigned to this E-Node. That is, all the E-Nodes in the same G-Nodes share their mapping information since they have the identical semantics in the same formats. This feature greatly enhances the reuse of existing mappings. The matching between $Schema_1$ and $Schema_2$ consists of three sequential phases (as shown in Fig 6.):

• **Searching Abstract Nodes** (phase 1): This phase is to find two sets of abstract nodes $ASet_1$ and $ASet_2$ that respectively correspond to $Schema_1$ and $Schema_2$. This phase is completed in two steps:
Step 1: *G-Node searching*. For each element in both schemas, the most suitable G-Node is obtained by the procedure described in lines 3-7 in Algorithm 1. In this step, the previous verified outcomes can be directly reused if the element has been inserted; otherwise, the suitability of G-Nodes is decided by the function *SearchGNode* in line 7. Then, all the suitable G-Nodes are inserted into the corresponding abstract-node sets.
Step 2: *A-Node searching*. An A-Node is selected into the abstract-node sets only when all its inclusive G-Nodes are already contained in the sets.

• **Discovering Mappings** (phase 2): This phase is to discover schema mappings from $Schema_1$ to $Schema_2$ and vice versa, represented as MP_1^2 and MP_2^1 respectively. It has two steps:

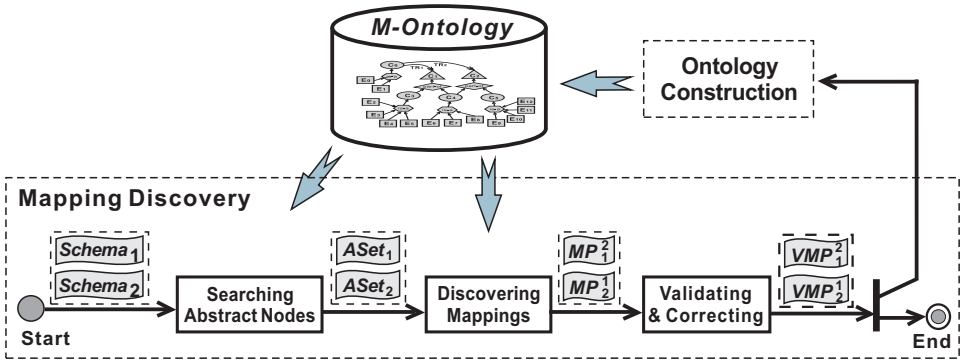


Figure 6: The flowchart of mapping discovery.

Step 1: *Searching the reachable abstract nodes*. An abstract node *cn* is considered *reachable* from $ASet_1$ only if it is able to find an edge to *cn* from any node in $ASet_1$. Thus, it is simple to obtain two sets of reachable nodes $RSet_1$ and $RSet_2$ respectively from $ASet_1$ and $ASet_2$ in a mapping ontology. Step 2: *Finding the overlap*. The overlaps $RSet_1 \cap ASet_2$ and $RSet_2 \cap ASet_1$ separately indicate the mappings MP_1^2 and MP_2^1 whose expressions are in the connected T-Edges. The overlap $ASet_1 \cap ASet_2$ denotes the semantics-equivalence mappings between two schemas. The demanded mappings can be directly derived from the above three overlaps.

• **Validating & Correcting Mappings** (phase 3): The final phase is to verify and correct the machine-discovered mappings by human beings. These verified mappings are recycled to incrementally construct M-Ontology by the proposed construction algorithm in Section 3.3.

Currently, the techniques of schema/ontology matching [42, 18] can hardly find the mappings with complex expressions that are very common in real-world data integration. The above approach attempts to overcome this bottleneck by reusing the existing expressions in mapping ontologies.

4 Implementation and Experiments

M-Ontology is built on top of an open-source system "Alignment Server" [19]. This system provides some basic services, like mapping storage and ontology matching. It employs a big table to store mappings, each of which is treated as a tuple in the table. We extend it to support our proposed mapping ontology.

To evaluate the feasibility and effectiveness of M-Ontology, we conduct experiments to simulate ontology construction and the subsequent mapping discovery. Mapping diversity and repetition depend on user selection of data sources, the composition of local schemas and the degree and frequency of their changes. Thus, it is hard to simulate the complete processes and performances of real applications. The following experiments focus on the effectiveness of two functions (i.e., *SearchGNode* and *VerifyENode*) that are the main factors determining the performance of the proposed algorithms in ontology construction and mapping discovery. More specifically, two questions are addressed in the experiments. i) *Feasibility*: in the real-world DI context, how high the percentage of schema elements have corresponding concepts stored in M-Ontology during the incremental construction process? ii) *Effectiveness*: how well do our algorithms perform correctly in searching out suitable G-Nodes for a schema (i.e., the function SearchGNode) after the insertion results of a set of schemas have been verified (i.e., the function VerifyENode)?

Performance Metrics. We use four metrics to evaluate the performance of the algorithms, *Hit-rate*, *Precision*, *Recall*, *Fmeasure*. *Hit-rate* is designed to measure the feasibility of M-Ontology. *Precision* and *Recall* are widely used in evaluating the effectiveness of information retrieval systems. *Precision* measures the degree of correctness of G-Node searching. *Recall* measures the degree of completeness of G-Node searching. By combining *Precision* and *Recall*, *Fmeasure*[25] is to examine the overall effectiveness of the proposed algorithm. In the setting of M-Ontology, these metrics can be defined for a specific value *m* as:

- $Hit\text{-}rate = MO_e / NUM_e$
- $Precision = Crt_e / Total_e$
- $Recall = Crt_e / MO_e$
- $Fmeasure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

where, NUM_e is the total element number; MO_e is the number of elements that can be correctly identified with the G-Nodes; Crt_e is the number of elements that are matched to the correct G-Nodes; $Total_e$ is the number of elements that are matched to a specific G-Node. Note all the above element numbers are from a schema to be inserted/matched.

Data Sets. To examine the performance of M-Ontology in real-world data integration problems, we collect 141 query form URLs from the UIUC web integration repository[2] after removing the inactive websites. All these query-form URLs are from three domains : Books(47), Movies(49) and Music Records(45), where the value in parentheses denotes the number of URLs in the domain. The information of query forms is manually extracted from the HTML source codes to generate the corresponding Web schemas. These schemas are represented in OWL to accommodate the mapping format of Alignment Server.

Domain	GN no.	Rare GN pct.	Rare Schema pct.
Movies	30	43.3%	16.3%
Books	35	49.0%	12.8%
Reords	23	56.5%	15.6%

Table 1: Statistics of the domains

Given these schemas in OWL, we identify the concepts (i.e. G-Nodes) by manually clustering the schema elements for each domain. As illustrated in Table 1, 30 G-Nodes are generated from the data set "Movies". Among them, 43.3% G-Nodes have only a single schema element, but these schema elements are from 16.3% schemas. That is, the "rare" concepts occur only in a few schemas. It indicates that most to-be-classified schema elements can find the corresponding concepts from the other schemas in the same domain. The other two domains have the same patterns.

Experiment Scenario. Assume that m Web schemas from the same domain are already inserted into M-Ontology based on our proposed algorithms. The insertions of these schemas have been already corrected and verified by human beings so that the attributes (e.g., Set_{gn}^{DA}) of the corresponding G-Nodes are updated to combine the verified schema elements. We attempt to find correct G-Nodes for a single schema X from M-Ontology. To estimate the performances, we design two sets of experiments: 1) X is not in M-Ontology (illustrated in Figure 7); 2) X might be in M-Ontology (illustrated in Figure 8).

Each experimental result shows eleven values of m ranging from 1 to 45. The performance measures for each m are calculated by the average of 200 samples, each of which is automatically generated from the schema sets in that domain. Each sample includes m different schemas existed in M-Ontology and the schema X. All these schemas are randomly selected.

Experiment 1: Without Schema Repetition. In this set of experiments, the schema X is randomly selected from the schemas that are not stored in M-Ontology. That is, X is different from any schema in M-Ontology for the purpose of removing the influence of possible repetitiveness. As shown in Figure 7, the same trends can be observed in all three domains.

In Figure 7(a), when m reaches 15, at least 90% elements in X can find out the correct G-Nodes so that these elements can directly reuse the associated mappings. The importance of existing schemas on Hit-rates are clearly indicated, as a sharp increase of Hit-rate can be seen when m is smaller than 15. The remaining curve seems to indicate that the additional schemas (> 15) almost

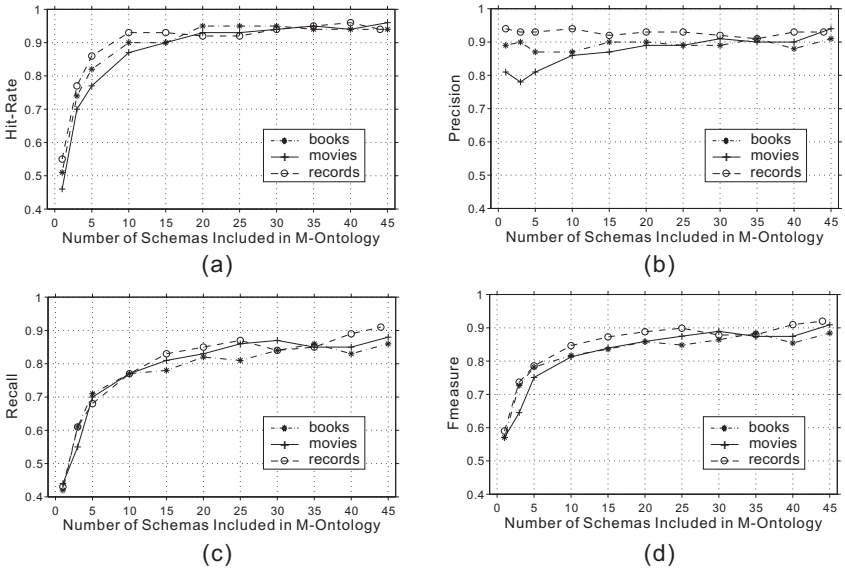


Figure 7: Experiment results of schema element classification without schema repetition.

do not affect the Hit-rates since 90% concepts are already collected from the first 15 schemas. However, this is not always true since more mapping edges might be added with more schemas inserted. The experimental results also match the conclusion by a related study [10] showing that the aggregated vocabularies used to describe schema elements are "clustering in localities and converging in size".

In Figure 7(b), the values of Precision stay around 90% after the initial learning process ($m >$ 10). Although more G-Nodes are available for classification with m increasing (Figure 7(a)), our proposed searching algorithm also can classify most schema elements into the correct concepts.

In Figure 7(c), a sharp increase of Recall can be seen before m reaches 10, followed by a steady and slow improvement with m increases. This phenomenon indicates that D-Labels of G-Nodes cannot be correctly identified when the inclusive schema elements are very few. When m increases, the contents of D-Labels become steady but Instances and D-Attributes can accumulate more useful information from the newly verified schema insertion. Thus, Recall increases slowly and steadily after m is larger than 10.

In Figure 7(d), the values of F-measure are higher than 85% when $m > 15$. As an overall performance measure, F-measure values indicate that the algorithms are effective in the classification of schema elements. Its real-world performance should improve further if we also employ the other two human intervention strategies, i.e., avoidance and prevention, mentioned in the end of Section 3.3.

Experiment 2: With Schema Repetition. When building a large number of DI systems, it is highly possible that X has already been inserted into M-Ontology since the ontology is shared by all the DI systems in the same domain. One typical example is given in the motivating scenario of Section 3.1. Thus, we also conducted the second set of experiments in which X is randomly selected from the whole schema set and it might be identical to one of the existing schemas in M-Ontology.

Intuitively, the results of Experiment 2 should be better than that of Experiment 1. The results

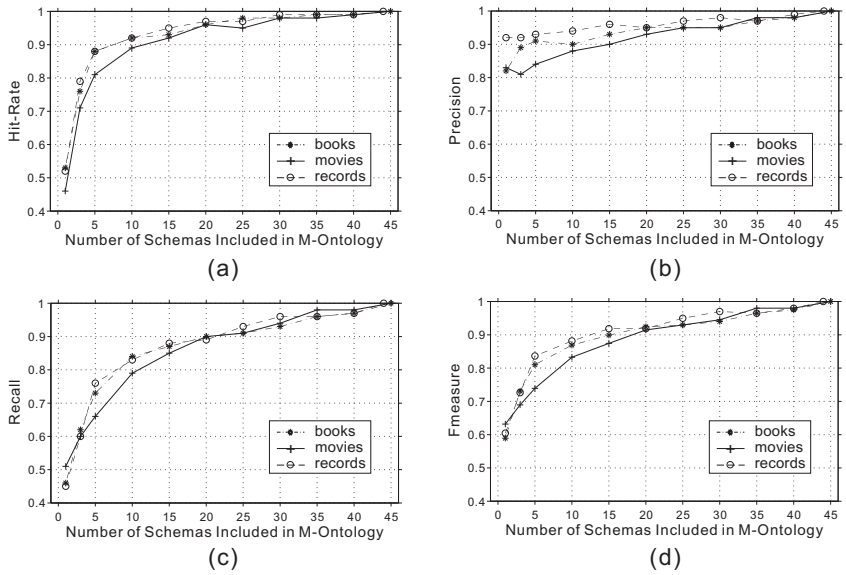


Figure 8: Experiment results of schema element classification with schema repetition.

prove this expectation clearly. Figure 8(a) reveals that the Hit-rate has reached 90% when m is around 10 and almost achieves the highest value (i.e., 100%) when m is around 45. Figure 8(d) shows that F-measure values are above 90% after m is greater than 20, and also reach the highest value. In real implementation, the algorithms should perform better than the experiment results in Figure 8, since most users will choose the data sources with more functionalities (i.e., complex query forms) and better data quality.

From the above two sets of experiments, we see that schema elements in the same domain can be clustered to generate a relatively "small" mapping ontology. That indicates it is feasible to construct a mapping ontology as a knowledge base shared by DI systems in the same domain. The performance results in terms of Fmeasure demonstrate the effectiveness of the algorithms even without considering the avoidance and prevention human intervention strategies. Furthermore, as the number of data sources on the Web has been steadily increasing [30, 10, 7], better performance of M-Ontology can be expected when more mappings are included.

5 Related Work

5.1 Requirement Specification in Data Integration Systems

Data integration systems are regarded as mediators to effectively match information needs and resources. The scale and diversity of user needs have been recognized in the context of DI over the Web [30, 11]. Generally, users can specify their preference and needs explicitly or implicitly. MetaQuerier [11] and WISE-Integrator [24] DI systems allow users to explicitly select a domain of interest. For each domain, a single mediated schema is created by combining the local schemas of *all* inclusive data sources. Such a mediated schema can only provide the functionalities that are supported by all the underlying sources; otherwise, the results from data sources can not conform to the original queries inputted from the users. More data sources it includes, less functionalities it can support. Moreover, different users may have different selection criteria for data sources,

e.g., data qualities, site credibility and brand loyalty. Thus, this approach hardly satisfy the diverse user needs and preferences. PayGo-based DI architecture [30] attempts to obtain the information needs from the keywords input by users. However, the extracted user needs are usually not exact and it is difficult for machines to transform user queries from flat keywords to structured queries. MySearchView [28, 24] enables users to specify their preferred data sources for generating the personalized DI systems, but it provides users a very limited source type for selection, i.e., only single-keyword-box data sources. In reality, query forms are much more complex, i.e., they include more controls. Due to the limit on source types, MySearchView does not address the challenges of mapping discovery between complex query forms, which is the core in DI system construction.

This paper follows the basic idea of MySearchView on personalized source selection, but employs a different construction approach, i.e., community-driven construction, for the purpose of eliminating its limitation on data sources. Since such customization leads to a large variety of DI systems to be constructed, the construction burdens should be distributed to a considerable number of cooperative members in a community, which can include users, technicians and domain experts. In this setting, this paper proposes a "human-friendly" mapping repository for efficient mapping storage, management and discovery. Unlike the traditional DI systems, these DI systems in the same domain share the same mapping repository. Its main consideration is that the construction and maintenance of the DI systems might benefit from the previous outcomes of repetitive tasks. This assumption is also observed in our experiment results. The more mapping information a mapping repository owns, the more benefits their construction and maintenance can obtain.

5.2 Mapping Modeling for Data Integration

Mapping modeling is the foundation of mapping repositories. The existing methods can be classified based on the extent to which mappings are abstracted: *instance-level* (IL) and *abstract-level* (AL) modeling. IL modeling is a method that regards mapping instances as separate entities without any abstraction, while AL modeling considers mapping instances as a whole body under some hidden templates/rules, which can be discovered from manual or automatic analysis and reasoning on the mapping instances.

IL modeling is widely utilized in many DI systems. Most mapping repositories [35, 37, 49] simply employ a large table to store schema-level or element-level mappings. Its columns represent different properties of mappings, such as schema/element names, mapping expressions, and similarity values. Others use graph models in which nodes denote schemas [5, 25] or elements [4, 36] and edges represent mappings. In a mapping graph, a new mapping edge between two nodes could be discovered by combining all the edges in a mapping path between these two nodes [31, 20, 6].

Compared with IL modeling, AL modeling facilitates mapping management and re-utilization by providing the patterns/heuristics hidden in mapping instances. Patterns are useful to assist humans/machines in reviewing previous mappings and finding new mappings. A study in [29] employs machine-learning techniques for finding the patterns. It leverages a large number of the existing mapping instances to train classifiers. Essentially, these classifiers can be viewed as a media to store the implicit patterns for finding new mappings. However, patterns in real life are too complex to learn by machines especially when the volume of training data is not large enough. Another problem is that these patterns are not straightforward to users. It is difficult for human users to view, verify and modify them. Consequently, this method is often applied to discover

approximate mappings [30], rather than exact mappings with expressions. Other studies [17, 9, 47] rely on human efforts to find templates, which can be viewed as explicit mapping patterns. These templates are separately hardcoded into the systems. They simply assume that the templates are static and separate. It is highly desirable that these templates can be semantically organized and dynamically managed.

Our proposed semantic modeling is a type of AL modeling. Different from the above learning-based and template-based methods, our proposed semantics-based method is to form a domain-specific ontology by conceptualizing mapping instances. The ontology is a container that stores a large number of templates. Users can dynamically manage these templates that are organized based on their semantics.

Compared with the learning-based methods, our proposed semantic modeling is more straightforward to humans, especially for the non-technical users. Since user interaction is inevitable in mapping management and discovery, easy-to-interact is of much significance particularly for large-scale mappings. In this context, the ontology-based model enhances the understanding of humans. In addition, these mapping ontologies are visualizable, and thus, the operations in mapping management can be transformed to several simple operations on graphs.

Compared with the template-based methods, our method is abstract and dynamic. Each template represents a group of mappings with some specific patterns. In our model, individual templates are merged into domain ontologies based on their meanings. The semantics-based organization facilitates the discovery of new templates by maximally reusing the previous ones. Furthermore, these domain ontologies can be easily re-configured through periodic updates, such as insertion and modification.

5.3 Schema Element Clustering for Data Integration

Clustering schema elements is not a new approach in the field of data integration. Element clustering is a process of organizing similar elements into the same groups and dissimilar elements into the different groups. Its main applications in DI include schema merging, result merging and schema matching. For supporting schema merging, mediated schemas are generated by abstracting a representative schema element from each element cluster adopted in ARTEMIS [8] and Dragut et al. [16]. In result merging, element clustering facilitates detection of the duplicated results returned from different data sources IWIZ [40].

Most relevant to our work is the clustering algorithms in schema matching, including Corpus-based [29], IceQ [47], and Zhao & Ram [50]. The hierarchical agglomerative methods are employed by all the above references in constructing multiple concept clusters. However, all these algorithms operate in a batch mode. That is, they assume that all the schema elements are already collected before the start of clustering. In fact, it is not feasible in the dynamic context. For dynamic reconfiguration, schema elements to be clustered are incrementally occurred. Furthermore, Zhao & Ram [50] also attempts to use K-means and SOM neural network as element clustering methods. Like the above methods, they also did not consider the user interaction issues during the clustering construction.

Our clustering algorithm is to identify concepts and relations for constructing a mapping ontology. It is a type of representative object-based methods. The degree of similarity of an element to a cluster is equal to its similarity value to the representative of this cluster. The representatives (i.e., D-Labels) of a cluster are abstracted from the existing nodes based on the term frequency statistics

of their D-Attributes, which are treated as a bag of words.

Our algorithm mainly differs from the above work in several respects. 1) *Inputs*: the inputs to our system are mappings rather than schemas, since our system is to store, manage and discover the mappings with their expressions. 2) *Outputs*: their clustering algorithms only output the concept clusters without any relation among these clusters, but our construction method is to build domain ontologies including concepts and relations. 3) *Algorithm*: our algorithm is incremental and "human friendly". First, the representative object-based clustering is more straightforward to non-technical users. The cluster representatives offer users simple and straightforward descriptions of clusters that are easier for human understanding. The performance of our algorithm can be more easily improved by manually changing the representatives, e.g., adding or removing several D-Lables and D-Attributes. Second, for adapting to the evolution of DI systems, it classifies the to-be-matched elements as soon as they arrive. This incremental mechanism is very important for a long running and shared mapping repository.

5.4 Semi-automatic Discovery of Complex Mappings

Although schema matching has been widely researched for thirty years, most of the current solutions (as in surveys [18, 42]) only consider one-to-one mappings. In reality, many-to-many mappings are pervasive in real-world applications. Current solutions to complex mappings can be classified to three categories as follows:

Learning-based approaches learn mapping templates from the existing mapping instances. iMAP [13], also called COMAP [15], is proposed to solve several types of 1-to-n complex mappings by comparing the content or statistical properties of data instances from two schemas. However, the number of its hardcoded rules limits the types of complex mappings that can be found. HSM [46] and DCM [23] employ correlation-mining techniques to find the grouping relations (i.e., co-occurrence) among elements, but their solutions do not take into account how these elements correspond to each other (i.e., mapping expressions).

Template-based approaches search the most appropriate mapping templates from a template set. The main drawback is their limited capability of finding complex mappings. The templates are separate and static, as mentioned in Section 5.2. In addition, the template number is normally very limited. IceQ [47] integrates two human-specified rules into their systems for partially finding Part-of and Is-A type mappings (i.e., two common one-to-many mappings). QuickMig[17], as an extension of COMA[3], summarizes ten common mapping templates, some of which are complex mappings. Several possible combinations of templates are also presented. It also designs an instance-level matcher for finding splitting or concatenation relationships.

Ontology-based approaches employ external ontologies for mapping discovery. Two schemas to be matched are first matched to a single ontology separately, and then element-level mappings can be found by deducing from the intermediary ontology. However, its performance is largely affected by the coverage and modeling of ontologies. For example, the ontology defined in SCROL[43] does not consider the syntactic heterogeneity (defined in Section 3.2.1) caused by various semantics-equivalent representations. In addition, the existing ontology-based solutions [48, 43] simply assume that the shared ontologies are already built, but building such well-organized ontologies is as hard as finding complex mappings. Ontology construction is also a labor-intensive, time-consuming and error-prone problem [14, 39, 45].

Our proposed mapping discovery algorithm is an ontology-based approach. It provides an

integrated solution to three indispensable subproblems: ontology design, ontology construction and mapping discovery. The design of M-Ontology addresses five kinds of heterogeneity: language heterogeneity, syntactic heterogeneity, instance heterogeneity, structural heterogeneity and semantic heterogeneity. In addition, ontology construction employs learning-based approaches, i.e. incremental representative-based clustering. Different from the classical ontology construction [14, 39], the proposed ontology is generated from schemas and mappings, which are abundant in our proposed dynamic and customized DI framework. As more schemas and mappings are inserted into the ontology, more mappings can be correctly discovered from the ontology.

6 Conclusions and Future Work

In the Web era, numerous DI systems are needed in meeting various and dynamic user requirements. To meet such a challenge, two additional capabilities, i.e., customizability and dynamic re-configurability, are desirable for DI systems over the Web. For holistically constructing these DI systems through mass collaboration, this paper proposes an ontology-based mapping repository to efficiently store, manage and discover mappings. The sharing, understanding and reuse of mappings are the main design considerations. Following these requirements, we develop a complete solution for modeling, constructing and utilizing M-Ontology. Experiments are conducted on real-world data integration over the Web, and the results show that our proposed approach is promising. Although it is primarily designed for data integration over the Web, it should be useful for applications where numerous similar mappings exist, such as, the data mediation problem in the composition of semantic web service.

Currently, we have implemented the basic M-Ontology. Future work in this direction includes: applying the proposed mapping ontology to the generation of mediated schemas (a.k.a., schema integration) [8, 16, 44] in the context of dynamic and customized DI; extending the domain-specific M-Ontology to support cross-domain DI; combining version mechanisms [26, 12] into M-Ontology for mass collaboration.

References

[1] Html 4.01 specification: Forms. http://www.w3.org/TR/html4/interact/forms.html.

[2] The UIUC web integration repository. http://metaquerier.cs.uiuc.edu/repository.

[3] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. in: SIGMOD Conference, 2005, pp. 906–908.

[4] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. SIGMOD Record, 28(1) (1999) 54–59.

[5] P. A. Bernstein. Applying model management to classical meta data problems. in: CIDR, 2003.

[6] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing mapping composition. VLDB J., 17(2) (2008) 333–353.

[7] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. in: WebDB, 2008.

[8] S. Castano, V. D. Antonellis, and S. D. C. di Vimercati. Global viewing of heterogeneous data sources. IEEE Trans. Knowl. Data Eng., 13(2) (2001) 277–297.

[9] K. C.-C. Chang and H. Garcia-Molina. Conjunctive constraint mapping for data translation. in: ACM DL, 1998, pp. 49–58.

[10] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. SIGMOD Record, 33(3) (2004) 61–70.

[11] K. C.-C. Chang, B. He, and Z. Zhang. Toward large scale integration: Building a metaquerier over databases on the web. in: CIDR, 2005, pp. 44–55.

[12] R. Conradi and B. Westfechtel. Version models for software configuration management. ACM Comput. Surv., 30(2) (1998) 232–282.

[13] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: discovering complex semantic matches between database schemas. in: SIGMOD Conference, 2004, pp. 383–394. ACM.

[14] Y. Ding and S. Foo. Ontology research and development. part 1 - a review of ontology generation. Journal of Information Science, 28(2) (2002) 123–136.

[15] A. Doan. Learning to map between structured representations of data. PhD thesis, University of Washington, 2002. Co-Chair-Halevy, Alon Y. and Co-Chair-Domingos, Pedro M.

[16] E. C. Dragut, W. Wu, A. P. Sistla, C. T. Yu, and W. Meng. Merging source query interfaces onweb databases. in: ICDE, 2006, page 46.

[17] C. Drumm, M. Schmitt, H. H. Do, and E. Rahm. Quickmig: automatic schema matching for data migration projects. in: CIKM, 2007, pp. 107–116.

[18] J. Euzenat and P. Shvaiko. Ontology Matching. Springer-Verlag New York, Inc., 2007.

[19] J. Euzenat, P. Valtchev, C. L. Duc, and etc. Alignment api and alignment server. http://alignapi.gforge.inria.fr/.

[20] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. ACM Trans. Database Syst., 30(4) (2005) 994–1055.

[21] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. in: SIGMOD Conference, 2005, pp. 805–810.

[22] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. in: VLDB, 2006, pp. 9–16.

[23] B. He and K. C.-C. Chang. Automatic complex schema matching across web query interfaces: A correlation mining approach. ACM Trans. Database Syst., 31(1) (2006) 346–395.

[24] H. He, W. Meng, C. T. Yu, and Z. Wu. Automatic integration of web search interfaces with wise-integrator. VLDB J., 13(3) (2004) 256–273.

[25] D. H. Hong. Schema Matching and Mapping-based Data Integration. PhD thesis, University of Leipzig, http://lips.informatik.uni-leipzig.de/pub/2006-4, 2006.

[26] R. H. Katz. Towards a unified framework for version modeling in engineering databases. ACM Comput. Surv., 22(4) (1990) 375–408.

[27] C. W. Krueger. Software reuse. ACM Comput. Surv., 24(2) (1992) 131–183.

[28] Y. Lu, Z. Wu, H. Zhao, W. Meng, K.-L. Liu, V. Raghavan, and C. T. Yu. Mysearchview: a customized metasearch engine generator. in: SIGMOD Conference, 2007, pp. 1113–1115.

[29] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. in: ICDE, 2005, pp. 57–68.

[30] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can only afford to pay as you go. in: CIDR, 2007, pp. 342–350.

[31] J. Madhavan and A. Y. Halevy. Composing mappings among data sources. in: VLDB, 2003, pp. 572–583.

[32] R. McCann, A. Kramnik, W. Shen, V. Varadarajan, O. Sobulo, and A. Doan. Integrating data from disparate sources: A mass collaboration approach. in: ICDE, 2005, pp. 487–488.

[33] R. McCann, W. Shen, and A. Doan. Matching schemas in online communities: A web 2.0 approach. in: ICDE, 2008, pp. 110–119.

[34] S. Melnik. Generic Model Management: Concepts and Algorithms. PhD thesis, University of Leipzig, 2004.

[35] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A programming platform for generic model management. in: SIGMOD Conference, 2003, pp. 193–204.

[36] P. Mitra, G. Wiederhold, and M. L. Kersten. A graph-oriented model for articulation of ontology interdependencies. in: EDBT, 2000, pp. 86–100.

[37] N. F. Noy, N. Griffith, and M. A. Musen. Collecting community-based mappings in an ontology repository. in: ISWC, 2008, pp. 371–386.

[38] B. Pine and S. Davis. Mass customization : the new frontier in business competition. Harvard Business School Press, Boston, Mass., 1999.

[39] H. S. A. N. P. Pinto and J. P. Martins. Ontologies: How can they be built? Knowl. Inf. Syst., 6(4) (2004) 441–464.

[40] C. Pluempitiwiriyawej and J. Hammer. Element matching across data-oriented xml sources using a multi-strategy clustering model. Data Knowl. Eng., 48(3) (2004) 297–333.

[41] R. A. Pottinger and P. A. Bernstein. Merging models based on given correspondences. in: VLDB, 2003, pp. 862–873.

[42] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. VLDB J., 10(4) (2001) 334–350.

[43] S. Ram and J. Park. Semantic conflict resolution ontology (SCROL): An ontology for detecting and resolving data and schema-level semantic conflicts. IEEE Trans. Knowl. Data Eng., 16(2) (2004) 189–202.

[44] D. Rosaci, G. Terracina, and D. Ursino. An approach for deriving a global representation of data sources having different formats and structures. Knowl. Inf. Syst., 6(1) (2004) 42–82.

[45] M. Sabou, C. Wroe, C. A. Goble, and H. Stuckenschmidt. Learning domain ontologies for semantic web service descriptions. J. Web Sem., 3(4) (2005) 340–365.

[46] W. Su, J. Wang, and F. H. Lochovsky. Holistic schema matching for web query interfaces. in: EDBT, 2006, pp. 77–94.

[47] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. in: SIGMOD Conference, 2004, pp. 95–106.

[48] L. Xu and D. W. Embley. Discovering direct and indirect matches for schema elements. in: DASFAA, 2003, pp. 39–46.

[49] A. V. Zhdanova and P. Shvaiko. Community-driven ontology matching. in: ESWC, 2006, pp. 34–49.

[50] H. Zhou and S. Ram. Clustering schema elements for semantic integration of heterogeneous data sources. J. Database Manag., 15(4) (2004) 88–106.