

A Formal Model of Dynamic Identification of Correspondence Assertions for E-Commerce Data Integration*

Sylvanus A. Ehikioya and Chima Adiele

Department of Computer Science
University of Manitoba
Winnipeg MB, Canada R3T 2N2
{adiele, ehikioya}@cs.umanitoba.ca

Abstract

Data integration involves merging two or more schemas into a global schema based on the identification of common concepts, and also the set of different concepts that are mutually related by some semantic properties. Identifying semantic relationships that exist between a set of objects in one schema and a different set of objects in another schema is crucial to the integration process.

Earlier integration strategies depend on significant manual input from the users and the ingenuity of the database administrators (DBAs) for the identification and declaration of correspondence assertions. We argue that it would be exceedingly difficult for users / DBAs to identify, declare, and specify the assertions that exist between objects, especially a mapping between two different attributes, for all attributes in the local databases. In this paper, we design an integration algorithm that leverages a simple semistructured data model and a common ontology to dynamically identify correspondence assertions. We exploit set theory and logic to formally specify the

assertions. We show how different users across diverse platforms can query the global integrated schema.

Keywords: Dynamic Correspondence Assertions, Electronic Commerce, Data Integration, Semistructured Data, Formal Specification.

1. Introduction

The Internet supports a customer-driven business environment, and organizations have taken advantage of this system to leverage their operations and customer base, thus maximizing profit. Making the customer a focal point of Internet-based commerce is key to customer relationship management. Trading on the Internet where all transactions are carried out electronically is called electronic commerce (e-commerce). In e-commerce, the Web provides a platform for buyers and sellers to electronically exchange goods and services by sharing information represented on the Web. The Web is an amalgam of data from a broad spectrum of domains that emanate from

* The funding for this research is provided by the University of Manitoba Graduate Fellowship and NSERC Grant Number #RGPIN217215-00.

disparate data sources. Web data exist in different formats; ranging from structured data and unstructured free text documents, to large volume of semistructured data, mainly represented as XML documents.

Many e-commerce systems' researchers [3, 8, 15, 17] focus on infrastructure to innovate the traditional marketplace, and provide an environment that guarantees customer satisfaction. For example, developments in the area of technological infrastructure are yielding positive results, as there is relative low cost of access to data through high speed Internet connectivity. In addition, communication links between computers are also well developed to guarantee high throughput and low latency. In terms of process infrastructure, the traditional marketplace is linked to the digital marketplace through electronic payment mechanisms, such as credit cards, debit cards and e-money, in transactions that are secured against intentional and unintentional abuses. Established protocols and legal frameworks offer a secure environment to regulate transactions and protect participants from abuses. However, data infrastructure development have continued to lag behind, and fails to provide easy access and transparency to e-commerce related data on the Web as users are currently saddled with the responsibility of locating and interpreting the enormous amount of Web data. In this context, transparency (structural) means an integration process devoid of the costly user / DBA intervention.

Current e-commerce data needs transcend the amalgamation of data. E-commerce participants desire a system that supports easy access and transparency to e-commerce related data on the Web. Data integration provides a global view of data across organizational boundaries, while retaining the autonomy of the data sources.

Unfortunately, traditional integration systems [7, 9, 14] are unable to adequately integrate e-commerce data because of the heterogeneity of Web data sources and non-interopability of e-commerce data. In this paper, we introduce a syntactically simple and semantically rich semistructured data model that is flexible to represent e-commerce data set. We propose an efficient integration algorithm that automates the integration processes including, identification of assertions. In addition, we identify correspondence assertions that use fine-grained comparison of objects at different object levels. Our approach captures the real meaning of elements, as every element is drawn from a common ontology, and hence eliminates the existence of multiple views and interpretations of real world states (RWS) of objects [LNE89]. Finally, we provide a query example to illustrate the integration mechanism.

This paper contributes by showing that correspondence assertions can be identified dynamically, if the semantic content of data is explicit and conceptualizations expressed in a natural way. We automatically identify similarities based on the matching of names and descriptions. To achieve this objective, we use a common ontology and a flexible semistructured data model. The common ontology ensures that objects with common names in different data sources represent the same RWS, although they may have different structural representations. Thus, we use the common ontology to resolve naming conflicts without the costly intervention of the database administrator (DBA). A simple and flexible semistructured data model allows structural independent representation of schemas, which enhances easy and fine-grained comparison of objects across data sources. Simple data models have inherent advantages over complex models in data integration because the operations to transform and merge data [4] are

correspondingly simpler in simple models. We use set theory and logic to provide a measure of formalism because formally specifying an integration system for e-commerce transactions provides a clear understanding of the system and reduces the complexity of the integration process. We show how users across diverse platforms can query the integrated schema.

In our specifications, we use common set notations (\cap , \cup , \supseteq , \subseteq , \notin , \in , \mathbb{N} , \mathbb{P}) as in [11] to describe structural components, and predicate logic to describe pre- and post-conditions for any requirements. We give these conditions (pre- and post-conditions) as predicates. A simple predicate, which usually has one or more arguments, is of the form $P(x)$, where x is an argument that is used in the predicate P . The specification of a requirement follows the general format of a quantified statement. In general, a quantified statement can be written in one of two forms:

- a).<quantifier><declaration(s)>•<predicate>
- b).<quantifier><declaration(s)>|<constraint>
•<predicate>

The universal quantifier (\forall) and existential (\exists) quantifiers are in common use. Every declaration must satisfy a given constraint. The symbols “|” and “•” which are part of the syntax mean "satisfying" and "such that", respectively. To create compound predicates, statements can be nested and combined together using one or more logical connectives, such as: *and* (\wedge), *or* (\vee), *not* (\neg), *conditional* (\Rightarrow), and *biconditional* (\Leftrightarrow). Truth tables for the logical connectives are available in most discrete mathematics textbooks. The formal specification of a requirement in this paper follows the general format of a quantified statement

The rest of this paper is organized as follows. Section 2 describes *Del-G*, a simple

and flexible semistructured data model, and discusses the need for a common ontology. Section 3 discusses the schema integration process while in Section 4 we describe correspondence assertions and provide formalism for the assertions. Section 5 discusses correspondence rules, and integration constraints. In particular, we define the inheritance property. In Section 6, we discuss the integration process, and provide an efficient integration algorithm that dynamically identifies correspondence assertions. In Section 7, we discuss related work and conclude the paper with an insight into future work in Section 8.

2. Representation of E-Commerce Data Set

Most of the data used in e-commerce activities reside in legacy systems, largely represented with rigid schema-based relational databases. However, some e-commerce data are also found on the Web, where the data structure is irregular. Representing e-commerce data with a rigid schema-based lacks the flexibility to cope with the dynamic nature of e-commerce data. The flexible nature of a semistructured data model makes it a good platform to represent data of different formats. In addition, semistructured data supports easy transformation and exchange across organizational boundaries. To describe the semistructured data model for this research, we first define a common ontology since every term in the model is drawn from the ontology.

2.1 Common Ontology

We adopt a common ontology to describe the terms used by different autonomous data sources. The ontology helps us to define the meaning of the terms used in the data sources. Thus, ontology

provides a framework for participants to speak a common language and hence understand themselves. Data elements from the individual data sources are only mapped to the common ontology. The common ontology we adopt does not compromise data model autonomy of participating data sources, as only the mapping of elements from the individual data sources to the common ontology is done. A formal definition of common ontology follows.

Definition 1: A *common ontology* CO is a finite set of terms $\{\tau_i\}$ organized in a hierarchical structure. Each term, τ_i , is a conceptual label representing a semantic object.

Formally,

$$CO \cong \{\tau_i: \mathbb{P} \text{ LABEL} \bullet \forall i, j: \mathbb{N} \mid i \neq j \bullet \tau_i \neq \tau_j\} \quad (1)$$

A semantic object represents a data element together with the underlying contextual information, referred to as semantic context. The semantic context is a combination of one or more objects drawn from the common ontology, which could be a complex type, an atomic type, or a combination of both. The semantic object, together with its underlying structure, describes the real world state of the object it represents.

2.2 The *Del-G* Model

A directed edge-labelled graph (*Del-G*) for an object is an acyclic digraph that encapsulates in it structural information of every object it represents, where each edge is labelled with at most one context label (name). The general terminology applicable to trees / graphs applies in their usual way (in this paper) to refer to relationships between objects in a schema or database. In particular, we use the following informal

definitions to describe tree concepts, where the elements u, v represent nodes and e an edge.

- $Child(u)$ returns the children (sub-elements) of u .
- $Parent(u)$ returns the parent of the element u .
- $Ancestors(u)$ returns the set of ancestors of u .
- $Descendants(u)$ returns the set of descendants of u .
- $Siblings(u, v)$ is a boolean function that returns true if u and v have the same direct parent, otherwise it returns false.
- $NodeLabel(u)$ returns the label of a node u .
- $EdgeLabel(e)$ returns the label of an edge e .
- $Source(e)$ returns the source node of an edge e .
- $Target(e)$ returns the target node of an edge e .

An edge in *Del-G* is a link between an ordered pair of nodes. Thus, $e_i = (u, v)$ represents an edge e_i where the $Source(e_i) = u$ and $Target(e_i) = v$. The basic types [NODE, EDGE, LABEL] represent a node, edge, and label, respectively.

Let $N : \mathbb{P}_1 \text{ NODE}$; $E : \mathbb{P} \text{ EDGE}$; and $L : \mathbb{P} \text{ LABEL}$

Figure 1 shows a generic representation of *Del-G*, which we now describe. Nodes $r_0, u_1, u_2, u_3, v, w \in N$ represent objects, and the pairs of nodes $(r_0, u_1), (r_0, u_2), (r_0, u_3), (u_2, v)$, and $(u_2, w) \in E$ are edges with the corresponding labels e_1, e_2, e_3, e_4 , and $e_5 \in L$. Edges model the hierarchical relationship between ordered pairs of objects, where in each pair, one is called source object and the other target object.

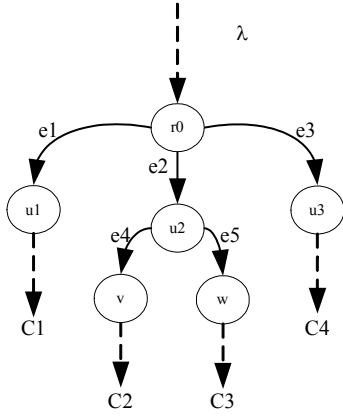


Figure 1: A Representation of *Del-G*

A label is a string of characters that describes the object it represents. There is a unique singleton set called $Root \subseteq N$, whose only member is the unique node, called *RootNode* (r_0). A node $r_0 \in Root$ is a root node if it has no incoming edge such that $Parent(r_0) = \emptyset$. The root node is traversed before any other node n in a pre-order traversal. Edges with solid lines represent edges that show hierarchical relationships between objects, while edges with broken lines are pseudo edges that associate either the *entry-point label* (λ) to r_0 or terminal nodes to constant values, CV ($C_i \in CV$, where $1 \leq i \leq n$). It would be necessary to differentiate between source object and target object of an ordered pair of nodes in an edge. This reasoning is motivated in part by the fact that an edge embodies a hierarchical order between objects.

Definition 2: A *Del-G* is a 4-tuple (N, E, L, CO) where:

1. N is a finite nonempty set of nodes.
2. E is a set of edges; and $\forall e: E \bullet \exists u, v: NODE \mid u \neq v \bullet Source(e) = u \wedge Target(e) = v$.
3. L is a set of labels; and $\forall e: EDGE \bullet \exists_1 l: L \mid l \in CO \bullet EdgeLabel(e) = l$.
4. CO is the common ontology.

Point 1 specifies that N is a finite nonempty set of nodes that represent objects.

By the inheritance property of objects, a node inherits the structural information of the object it represents. Point 2 specifies E as a set edges, and an edge as an ordered pair of nodes (u, v) , where u is the source and v is the target. Point 3 specifies L as a set of labels such that every label is drawn form CO , and every edge has at most one label. Point 4 specifies that CO is a common ontology.

The schema of an object is a set of nodes that describes the structure of the object in *Del-G*. The *Del-G* model is defined with its unique syntax and its associated semantics. The syntax is derived from the structure of *Del-G* and the objects. The semantics of the model shows the inherent relationships (including parent, child, ancestor, descendant, etc.) that exist between objects. Let CV denote the set of constant values that represent the values of objects. Val maps terminal nodes in *Atomic-Node* to CV . A formal definition of the schema of an object follows.

Definition 3: The *Schema* of an object $S = (\varphi, r_0, \lambda, Val)$ is a *Del-G* where:

1. $\varphi = \bigcup (Root, Complex-Node, Atomic-Node) \subseteq N \mid \bigcap (Root, Complex-Node, Atomic-Node) = \emptyset$;
2. $r_0 \in Root$ is a special node in *Del-G*, called the root node;
3. $\lambda: LABEL$ is a special label called the entry point label; and
4. $Val: N \rightarrow N \times CV$.

A *Schema* also satisfies the following conditions: 1) terminal nodes have no outgoing edges; 2) the root node has no incoming edge, but every schema has an entry point label that is associated with the root node; and 3) each node n is reachable from the root node r_0 and its label, λ . Let

SCHEMA be the basic type of a schema. A formal specification of a schema follows.

$$\begin{aligned} \forall S: \text{SCHEMA} \bullet S = (\varphi, r_0, \lambda, Val) \Leftrightarrow \\ \varphi = \bigcup (Root, Complex-Node, Atomic-Node) | \\ \bigcap (Root, Complex-Node, Atomic-Node) = \emptyset \wedge \\ \exists r_0: Root \bullet NodeLabel(r_0) = \lambda \wedge \\ \forall e: EDGE \bullet (\exists u, v: NODE \mid u \neq v \wedge u, v \in \varphi \bullet \\ Source(e) = u \wedge Target(e) = v) \wedge \\ (\exists l: LABEL \bullet EdgeLabel(e) = l) \wedge \\ Val(Target(e)) \in CV \Leftrightarrow \\ Target(e) \in Atomic-Node))) \end{aligned} \quad (2)$$

In the *Del-G* model, as in object exchange model (OEM) [10], labels are first class citizens, and are used in place of a schema. However, unlike OEM, we attach labels on the edges instead of the nodes to exploit path knowledge. Path knowledge is important to successful access of semistructured data. This model, like OEM, is self-describing, and has no a priori schema. One major improvement of *Del-G* over the other models [2, 10, 12] is that the user is not saddled with the responsibility of identifying and interpreting the objects since every object must come from a common ontology. To avoid repeatedly traversing sub-graphs due to multiple edges between two nodes, we assume that *Del-G* contains at most one edge between two nodes. There is a natural recursive ancestral relationship that creates the notion of path knowledge, and hence enhances the quality of queries posed. The *Del-G* model mimics the simplicity of OEM, which is flexible for semistructured data, yet powerful enough to represent other models and enhance integration. Therefore, the model is able to provide information exchange among organizations, since it could represent every other model.

3. Schema Integration Process

Figure 2 shows the “input-process-output” components of a schema integration

process. In this paper, we adopt a two-phase integration strategy. In Phase 1, our primary concern is to present the input schemas in a platform independent format and eliminate multiple views of object.

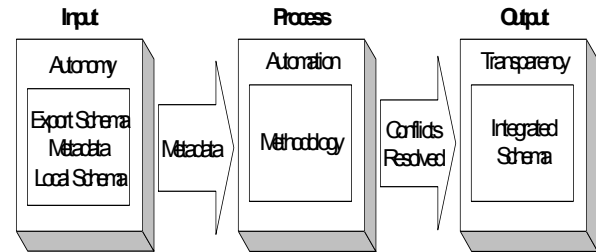


Figure 2. The Integration Process

To this end, the integration mechanism allows data sources access to a Web-based *Del-G* template so the local schemas are mapped into the platform independent *Del-G* format to ease comparison of objects. *Del-G* is able to represent local schemas since every schema can be represented as a graph. Sheth and Larson [15] observe that, “unless the schemas are represented in the same model, analyzing and comparing their schema objects is extremely difficult.” Terms in the local schemas are also mapped to the common ontology to eliminate multiple views of objects among data sources. A *filter mechanism* validates the input to ensure that every local schema conforms to the *Del-G* standard. The filter mechanism either accepts or rejects a given local schema. Phase 1 is largely carried out offline.

In Phase 2, accepted local schemas are used as input to an integration algorithm that dynamically identifies correspondence assertions, and generates integration rules after analyzing the assertions. The goal of our approach is to generate an output that is transparent, at least from the users’ point of view. Transparency in this context refers to structural transparency. The user should see the global view as a common and reliable interface local to a particular user. Next, we

discuss correspondence assertions and their analyses.

4. Correspondence Assertions

We define correspondence assertion as a declarative statement, generated by the integration system, that shows the relationship between objects in different data sources at various levels of perception. This definition differs from [6] that only measures relationship at the same level of perception (e.g., entity to entity, links to links, or attribute to attribute). Here, we relax the definition to allow for the comparison of object types at different levels of perception (e.g., objects and sub-objects, objects and attributes, and so on) provided the elements have the same names.

4.1 Assertions Specification

Recall, objects with the same name represent the same real world state, because objects are drawn from a common ontology. Thus, erroneous choices regarding names that hitherto existed in earlier systems no longer exist. However, structural conflicts still exist since different designers may model a particular object differently due to variations in the designers' perceptions and modeling constructs. Therefore, we require, as in [6], that corresponding objects have the same names. Let L_1 be the name of object O_1 from schema S_1 and let L_2 be the name of object O_2 from schema S_2 . O_1 has children $O_{11}, O_{12}, \dots, O_{1n}$ (sub-objects or attributes) with corresponding names $L_1^{L1_1}, L_1^{L1_2}, \dots, L_1^{L1_n}$. O_2 has children $O_{21}, O_{22}, \dots, O_{2m}$ (sub-objects or attributes) with corresponding names $L_2^{L2_1}, L_2^{L2_2}, \dots, L_2^{L2_n}$. A relationship exists between O_1 and O_2 if there exists names $L_1^p \in S_1$ and $L_2^r \in S_2$ such that $L_1^p = L_2^r$. For example, L_1^p represents the i th element whose parent is $p \in S_1$ (where p : LABEL and i : \mathbb{N}_1). L_1 represents an element

without a parent in S_1 . In *Del-G*, there is only one distinguished element without a parent, the entry-point label.

Identity Assertion

O_1 is said to be identical to O_2 if and only if $L_1 = L_2$ and $L_1^p = L_2^q$, ($1 \leq i \leq n$ and $1 \leq j \leq m$ such that $n = m$). Informally, O_1 and O_2 are precisely the same because the designers used the same constructs and perceptions, which means that $(L_1 = L_2 \wedge (Child(L_1) = Child(L_2)))$.

Equivalence Assertion

O_1 is said to be equivalent to O_2 if $L_1 = L_2$ and $L_1^p = L_2^q$, ($1 \leq i \leq n$ and $1 \leq j \leq m$ such that $n \neq m$). Informally, O_1 and O_2 are not precisely the same because equivalent and inexact modeling constructs and perceptions have been used, which means that $(L_1 = L_2 \wedge ((Child(L_1) \supseteq Child(L_2)) \vee (Child(L_1) \subseteq Child(L_2))))$. Object L_1 is modeled in such a way that it emphasized all the values of object L_2 , or object L_2 is modeled so that it emphasized all the values of L_1 .

Intersection Assertion

O_1 and O_2 have an intersection assertion if $L_1 = L_2$ and $Child(L_1) \cap Child(L_2) \neq \emptyset$. Informally, O_1 and O_2 are perceived the same way but the modeling constructs differ slightly, which means that $(L_2 = L_2 \wedge (Child(L_1) \cap Child(L_2) \neq \emptyset))$. Object L_1 is modeled in a way that it emphasized some values of object L_2 , and object L_2 is modeled so that it also emphasized some values of L_1 .

Compatibility Assertion

O_1 is said to be compatible to O_2 if $L_1 = L_2$, and $Child(L_1) \cap Child(L_2) = \emptyset$. Informally, O_1 and O_2 are neither identical nor equivalent, but the modeling constructs are not contradictory. With compatible assertion, given by $L_1 = L_2 \wedge (L_1^p \cap L_2^q = \emptyset)$ (where, $1 \leq i \leq n$ and $1 \leq j \leq m$), L_1 and L_2 represent the same object since they have the

same name, but the attributes of $L_1^{p_i}$ in S_1 are different from the attributes of $L_2^{q_j}$ in S_2 .

Inclusion Assertion

O_1 and O_2 have an inclusion assertion, if $L_1 \neq L_2$ and $(L_1 \in \text{Descendant}(L_2) \vee L_2 \in \text{Descendant}(L_1))$. Informally, inclusion assertion states that O_1 and O_2 have different perceptions but equivalent modeling constructs. L_1 is either a sub-object or attribute of L_2 , or L_2 is a sub-object or attribute of L_1 . Suppose L_1 is a sub-object of L_2 ($L_1 \in \text{Descendant}(L_2)$), and all the attributes of L_1 are also attributes of the corresponding sub-object of L_2 , then we have a proper inclusion.

Included-Intersection Assertion

O_1 and O_2 have an included-intersection assertion if $L_1 \neq L_2 \wedge (\exists b: \text{Descendant}(L_2) \mid L_1 = b \wedge \text{Child}(L_1) \cap \text{Child}(b) \neq \emptyset) \vee (a: \text{Descendant}(L_1) \mid L_2 = a \wedge \text{Child}(L_2) \cap \text{Child}(a) \neq \emptyset)$. Informally, included-intersection assertion states that O_1 and O_2 are perceived differently, and also the modeling constructs differ. L_1 is either a sub-object or attribute of L_2 , or L_2 is a sub-object or attribute of L_1 . Suppose L_1 is a sub-object of L_2 , and intersection of the attributes of L_1 and the attributes of the corresponding sub-object of L_2 is not empty, then we have an included-intersection assertion.

Included-Compatible Assertion

O_1 and O_2 have an included-compatible assertion if $L_1 \neq L_2 \wedge (\exists b: \text{Descendant}(L_2) \mid L_1 = b \wedge \text{Child}(L_1) \cap \text{Child}(b) = \emptyset) \vee (a: \text{Descendant}(L_1) \mid L_2 = a \wedge \text{Child}(L_2) \cap \text{Child}(a) = \emptyset)$. Informally, included-compatible assertion states that O_1 and O_2 are neither perceived the same way nor have equivalent modeling constructs, but are not entirely unrelated. L_1 is either a sub-object or attribute of L_2 , or L_2 is a sub-object or attribute of L_1 . Suppose L_1 is a sub-object of L_2 and intersection of the attributes of L_1 and the attributes of the corresponding sub-object

of L_2 is empty, then we have an included-compatible assertion.

Mutual-Inclusion Assertion

O_1 and O_2 have a mutual-inclusion assertion if $L_1 \neq L_2$ and $(L_1 \in \text{Descendant}(L_2) \wedge L_2 \in \text{Descendant}(L_1))$. Informally, mutual-inclusion assertion states that O_1 and O_2 are not perceived the same way but have equivalent modeling constructs. L_1 is a descendant of L_2 , and L_2 is also a descendant of L_1 .

Exclusion Assertion

O_1 and O_2 have an exclusion assertion if $L_1 \neq L_2$ and $(L_1 \notin \text{Descendant}(L_2) \wedge L_2 \notin \text{Descendant}(L_1))$. Informally, exclusion assertion states that O_1 and O_2 neither agree in constructs, nor in the designers' perception. L_1 and L_2 are not the same, and they are not attributes of each other. However, L_1 and L_2 could have a common ancestor.

Notice that exclusion assertion can only occur between entry-point labels in S_1 and S_2 . In the next session, we formalize the process of mapping elements of two schemas and develop suitable algorithms to produce the mapping results.

5. Correspondence Rules

In this system, the integration algorithm automatically performs matching based on the name and structure of schema elements to generate correspondence assertions. Semantics is inbuilt in the structure because of the hierarchical nature of *Del-G*. An analysis of each assertion is carried out to produce formal rules that state how to derive the constructs, which are to be inserted into the global integrated schema (*GIS*). These formal rules, also called correspondence rules (\mathfrak{R}), are stated automatically using descriptive logic. We employ descriptive

logics to simplify the modeling of complex hierarchical structures using a set of logics. Formal rules present a well-founded logical framework in which concepts are intentionally defined using descriptions built from a set of constructors. Whereas correspondence assertions show the relationship between two local schemas, correspondence rules logically represent the derivations between the local schemas and the *GIS*. The specification of correspondence rules is an integral part of our research; however, we omit it in this paper without any loss of utility.

For each pair of elements of S_1 and S_2 that have common names there is an assertion that shows the relationship between those elements. Let ω be a set of pairs of elements (L_1^p, L_2^q) of S_1 and S_2 ($i, j: \mathbb{N}_1$) such that $L_1^p = L_2^q$, and elements of ω are related according to the assertions declared in the previous section. By definition,

$$\omega = \bigcup_{i,j}^n (L_1^p, L_2^q) \text{ such that} \\ \forall L_1^p \in S_1 \bullet (\exists L_2^q \in S_2 \bullet L_1^p = L_2^q) \quad (3)$$

A pair of elements (L_1^p, L_2^q) are equal if: 1) at least one of the elements is an entry-point label and have the same name (label) as the other element which may not necessarily be an entry-point label; 2) both elements are not entry-point labels and have the same names such that $p = q$. Formally,

$$\begin{aligned} & \text{Equal: LABEL} \times \text{LABEL} \\ & \forall L_1^p, L_2^q: \text{LABEL} \bullet \\ & (\exists S_1, S_2: \text{SCHEMA} \mid (L_1^p \in S_1 \wedge L_2^q \in S_2) \bullet \\ & (\text{Equal}(L_1^p, L_2^q) = \text{TRUE}) \Leftrightarrow \\ & (\exists \lambda_1, \lambda_2: \text{LABEL} \mid \lambda_1, \lambda_2 \in L_{EP} \bullet \\ & (\lambda_1 = L_1^p \wedge \lambda_2 = L_2^q \wedge L_1^p = L_2^q) \vee \\ & (\lambda_1 = L_1^p \wedge L_2^q \neq \lambda_2 \wedge L_1^p = L_2^q) \vee \\ & (\lambda_2 = L_2^q \wedge L_1^p \neq \lambda_1 \wedge L_1^p = L_2^q) \vee \\ & (L_1^p, L_2^q \notin L_{EP} \wedge p = q \wedge L_1^p = L_2^q)) \end{aligned} \quad (4)$$

Let ψ be an assertion such that:

$$\psi ::= \text{Identity} \mid \text{Equivalence} \mid \text{Intersection} \\ \mid \text{Compatibility} \mid \text{Inclusion} \mid \text{Mutual-} \\ \text{Inclusion} \mid \text{Exclusion}$$

$\psi(\omega)$ is a function that declares the relationship existing between the elements of ω . The assertion (ψ) shows how elements of ω are related. Let $\varpi = (L_1^p \cup L_2^q) \setminus \omega$ be a set of elements from the local schemas (S_1, S_2), such that $L_1^p \neq L_2^q$ (The operator ' \setminus ' is the set difference operator). Elements of ϖ maintain their local relationships with their ancestors or descendants, which are merged into the *GIS* by *inheritance*. \mathfrak{R} is a set of production rules for generating the *GIS* from ω and ϖ . *GIS* is generated such that elements of ω are merged according to their respective assertions, while elements of ϖ are merged into *GIS* such that the existing local relationships between those elements and their parents are maintained. To ensure that definitions of elements in the local schemas are still valid after integration, we introduce the notion of inheritance.

5.1 The Inheritance Property

An object in the integrated schema inherits the descendants of any of its children from the local schema if no further assertion exists between that child and a corresponding object in another schema that is involved in the integration process. For example, $L_1 \in S_1 \wedge L_2 \in S_2$ such that $L_1 = L_2 \Rightarrow \exists L: \text{LABEL}$ such $L \in \text{GIS}$. Suppose $L_1^p \in \text{Child}(L_1) \wedge L_1^p \notin \text{Child}(L_2)$ then $\text{Descendant}(L_1^p)$ in the local schema will be added into the (4) is $\text{Descendant}(L_k^q)$ since no other assertion exists between L_1^p and $\text{Child}(L_2)$. This integration process leverages the inheritance property to ensure that objects integrated into the *GIS* maintain existing relationships with objects in the local schemas, such that the definitions in the local schemas are still valid.

Let L_{SET} be a set of label. A formal definition of *Inheritance* follows:

$$\begin{aligned}
 & \text{Inheritance: } L_{SET} \rightarrow L_{SET} \\
 & \forall L_1: \text{LABEL} \bullet \exists S_1: \text{SCHEMA} \mid L_1 \in S_1 \bullet \\
 & \text{Inheritance}(L_1) \Leftrightarrow (\exists L_2: \text{LABEL} \wedge \\
 & \exists S_2: \text{SCHEMA} \mid L_2 \in S_2 \bullet L_2 = L_1 \wedge \\
 & (\exists a: \text{LABEL} \bullet a \in \text{Child}(L_1) \wedge \\
 & (\forall b: \text{LABEL} \mid b \in \text{Child}(L_2) \bullet b \neq a \Rightarrow \\
 & (\exists L: \text{LABEL} \wedge \exists GIS: \text{SCHEMA} \mid L \in GIS \bullet \\
 & L = L_1) \wedge a \in \text{Child}(L) \Rightarrow \exists c: \text{Child}(L) \bullet a = c \wedge \\
 & (\forall a_i: \text{Descendant}(a) \bullet \exists c_i: \text{Descendant}(c) \bullet a_i = c_i))
 \end{aligned} \tag{5}$$

5.2 Integration Constraints

To simplify correspondence rules generation process and improve the quality of meta-information provided to the integration system, we make the following clarifications. These clarifications influence the analysis of correspondence assertions and consequently, the generation of correspondence rules by the integration algorithm.

Element Constraints

1. Corresponding elements of the same modeling concepts are integrated into a similar element. For example, if L_1 is an object in S_1 and L_2 also an object in S_2 , and there is a correspondence assertion between both elements, then L_1 and L_2 are integrated into $L \in GIS$ as objects. Similarly, if L_1 and L_2 are attributes, they are integrated as attributes.
2. Corresponding elements modeled differently are integrated into a more accommodating type. For example, if objects and attributes correspond, they are modeled as objects in integrated system.

Data Type Constraints

1. When types are compatible, the less restrictive of the two data types being integrated is held as the type for the

IS. For example, a decimal is chosen when the types to be integrated are decimal and integer.

2. Data types that are not closely related are incompatible, e.g. string and integer. The union of the two local data types will form the data type for the integration systems.

Cardinality Constraints

Another interesting thing to consider is how to resolve constraint conflicts among data sources. We adopt a generous strategy that allows a wider scope of the constraints of the two schemas involved in an integration process by taking the least restrictive of the cardinality constraints. In this way, the cardinality constraints of the two schemas are subsumed in IS. For example, if two objects L_1 and L_2 from schemas S_1 and S_2 respectively have the following corresponding pairs of cardinality constraints ($[0:1]$, $[1:1]$), ($[1:1]$, $[1:n]$), and ($[1:n]$, $[0:n]$), then the effective cardinality constraint for the IS is ($[0:1]$), ($[1:n]$), and ($[0:n]$) respectively.

Terminal Element Constraints

We state, without prejudice to elements constraints, that the assertion between terminal elements is dependent on the assertions between their parents. If there is an assertion between terminal elements that conflicts with the assertion between the parents of such terminal elements, to that extent, the terminal element assertion is invalid, and does not lead to a “well-formed” integrated schema. Suffice it to say that no correspondence assertion exists between terminal elements unless there is an assertion between their respective parents. For example, a correspondence assertion exists between terminal element $a \in \text{Child}(L_1)$ from S_1 and terminal element $b \in \text{Child}(L_2)$ from S_2 if and only if $(a, b) \in \omega$ and $(\text{Parent}(a), \text{Parent}(b)) \in \omega$. A *Terminal-*

Element is an edge label where the target of the edge is in the set of atomic nodes.

Formally,

$$\begin{aligned} \text{Terminal-Element} &\cong l \mid l: \text{LABEL} \bullet \\ (\exists e: \text{EDGE} \bullet \text{EdgeLabel}(e) = l &\Leftrightarrow \\ \text{Target}(e) \in \text{Atomic-Node}) & \end{aligned}$$

The definition of terminal element assertion (*Terminal-Assertion*) follows.

Terminal-Assertion: Terminal-Element \times *Terminal-Element*

$\forall a, b: \text{Terminal-Element} \bullet$

$$\begin{aligned} \text{Terminal-Assertion}(a, b) &\Leftrightarrow (\exists S_1, S_2: \text{Schema} \wedge \\ \exists L_1, L_2: \text{Label} \mid L_1 \in S_1 \wedge L_2 \in S_2 \wedge & \quad (6) \\ \text{Parent}(a) = L_1 \wedge \text{Parent}(b) = L_2) \bullet \\ L_1 = L_2 \wedge \text{Parent}(a) = \text{Parent}(b) & \end{aligned}$$

If the defined correspondence assertions exist, the terminal elements are then integrated into the *GIS* according to the assertion. Stated differently, the above formalism stipulates that no correspondence assertion exists between any two corresponding terminal elements if the parents of the terminal elements are unrelated. For example, if the parents of a and b are not related by any correspondence assertion, then a and b cannot be integrated into a corresponding terminal element in the *GIS*. Thus, a and b would rather be inherited by their respective parents in the *GIS*, using the inheritance property.

6. The Integration Mechanism

The integration mechanism deals with the process module of Figure 2. The process module provides an integration algorithm that dynamically identifies assertions, determines attribute relationships, and generates correspondence rules without costly user intervention. Therefore, the major task in this module is to dynamically resolve heterogeneities that exist with a view to producing a global integrated schema that

guarantees data transparency across data sources.

6.1 The Integration Algorithm

In this system, the integration algorithm automatically matches terms to generate correspondence assertions. Matching of terms is based on the name and structure of schema elements. Let d be the degree of each element, and let p and q represent the parents of elements. $L_1^{p_i}$ and $L_2^{q_j}$ (where, $0 \leq i, j \leq d$) represent elements of the local schemas S_1 and S_2 respectively. For example, $L_1^{p_i}$ represents the i th element whose parent is P in S_1 , and $L_2^{q_j}$ represents the j th element whose parent is q in S_2 . Let $L_{EP} \subseteq L$ be the set of entry-point labels such that the entry-point labels λ_1 and λ_2 are represented as L_1 and L_2 since $i, j = 0$ and $p, q = \emptyset$. In this framework, matching of elements is based on names and structure of schema elements. The integration algorithm determines if a pair of elements $(L_1^{p_i}, L_2^{q_j})$ from S_1 and S_2 , respectively is equal (4). Let E_{LS} be the sum of all the pairs of edges in (S_1, S_2) , then,

$$E_{LS} \leq \sum_{i=1}^d \sum_{j=1}^d (L_1^{p_i}, L_2^{q_j}) \quad (7)$$

The problem at hand is to integrate two regulated schemas S_1 and S_2 into *GIS*. The integration process is of the form:

$$\text{GIS} = \bigcup_{i=1}^n \Upsilon_i[\psi_i(S_1, S_2)] \quad (8)$$

where ψ_i is the correspondence assertion between pairs of elements from S_1 and S_2 , and Υ_i is the corresponding integration rule over each correspondence assertion that generates elements from S_1 and S_2 . The integration algorithm uses a breadth-first search strategy to manage an integration queue structure IQ . The elements of IQ are pairs of labels from S_1 and S_2 , where the initial head of IQ is a pair of the entry-point

labels (λ_1, λ_2) from S_1 and S_2 , respectively. Pairs of elements of (S_1, S_2) are enqueued into IQ in a parent-child fashion. The algorithm compares each pair of elements in (S_1, S_2) at most once to dynamically identify similarities based on the matching of names and semantic descriptions. The algorithm is dominated by three operations; 1) *enqueue*, which enqueues pairs of elements from S_1 and S_2 into IQ ; 2) *compare* compares each pair of elements from S_1 and S_2 to determine likely assertions; and 3) *dequeue* removes pairs of elements that have been generated in GIS from IQ .

Initially, the pair of entry-point labels (L_1, L_2) are dequeued and compared. If L_1 and L_2 are equal, then pairs of elements containing children of L_1 and L_2 $(L_1^p_i, L_2^q_j)$ are enqueued into IQ to determine the specific assertion (ψ_i) . Once a particular assertion is established, merging occurs according to the corresponding integration rules (Y_i) . The *Merge* operation produces the correspondence rules (Y_i) and merges the elements of the local schemas into the GIS . Every other pair in IQ containing any merged elements is also dequeued. The child elements of each merged pair $(L_1^p_i, L_2^q_j)$ are enqueued into IQ and the process is repeated. A child element of any element from S_1 or S_2 that is merged into GIS is inherited if there is no assertion between that element and other elements in another local schema that have the same parents.

If L_1 and L_2 are entry-point labels, and are not equal, then the algorithm searches through S_2 to locate L_1 and S_1 to locate L_2 . The search returns “false” if not found, otherwise, the function returns “true”, and the location where it was found. If found is true, the algorithm enqueues children of pairs of elements $(L_1^p_i, L_2^q_j)$ that are equal into IQ , and compare them to identify the particular assertion. The process is repeated until all the elements in the local schemas are

compared or inherited into the GIS . If found is false, then elements of the two schemas are merged according to the corresponding assertion. The algorithm leverages the *inheritance property* to isolate inherited elements from the search space.

Recall that the sum of the pairs of elements (since every edge represents an element) in the two schemas is $\Theta(E_S)$, and the cost of enqueueing and dequeuing those elements is $O(E_S)$. In this algorithm, every pair of elements is compared at most once, and the cost of comparing all the pairs of elements in any given $Del-G$ is $O(E_S)$. Therefore, the algorithm takes $O(E_S + E_S) = O(2E_S) = O(E_S)$, where E_S is the size of the sum of edges of (S_1, S_2) . We omit an elaborate integration example that would have demonstrated how our algorithm works because of space constraints.

6.2 Correctness of the Integration Process

The integration mechanism produces a set of pre-computed views, GIS , that represents the local schemas, and could be queried by different users across diverse platforms. The query process relies on such views to derive correct answers. In $Del-G$, a regular expression represents a query that exploits path knowledge to retrieve all pairs of nodes connected by a path. A query Q expressed in terms of the GIS is decomposed into sub-queries q_i expressed in terms of the local schemas $\bigcup_{i=1}^n S_i$.

6.3 Querying the GIS

For a user to access a semistructured data model, such as $Del-G$, the user has to query the system to request the information needed. Queries are posed based on the underlying syntax of a given language. The ability to query semistructured data is supported by languages that exploit arc

variables bounded to labels on the arcs rather than nodes in the graph [2]. For the purpose of this paper, all queries are written in *Lorel* [1]. The *Lorel* query language takes advantage of successive representation of edge-labels to reach arbitrary depths in the edge-labeled graph. To formally define a query, we define a path label.

Definition 4: A *Path-Label* (pl) for an object O in $Del-G$ is a sequence of labels $l_j \in L \mid j: \mathbb{N}_1$, where each l_j is delimited by a period, such that starting from O , a sequence of n edges is traversed and the labels l_j correspond to edges e_i ($0 \leq i \leq n$). The pseudo edge, e_0 , is associated with the entry-point label, λ .

Definition 5: A *query* Q on a schema S is a regular language over $\Sigma \mid \Sigma : LABEL \wedge \Sigma \subseteq CO$, such that $Q(S) = (u, l_i, v_1)$. The triple (u, l_i, v_1) is a path from u to v_1 labelled with l_i . The element (u, l_i, v_1) is a solution (satisfies) to $Q(S)$ if and only if l_i is an instance of a *Path-Label*, $pl \in S$.

A query Q over GIS is decomposed into sub-queries $(q_i \mid i: \mathbb{N}_1)$ over $\bigcup_{i=1}^n S_i$. Suppose, there is *exact* decomposition of Q into q_i such that every query in q_i is contained in $Q(GIS)$ and $Q(GIS)$ is also contained in $\bigcup_{i,j=1}^n q_i(S_j)$, then we write

$$\bigcup_{i,j=1}^n q_i(S_j) \sqsubseteq Q(GIS) \Rightarrow Q(GIS) \sqsubseteq \bigcup_{i,j=1}^n q_i(S_j) \bullet$$

$$Q(GIS) \equiv \bigcup_{i,j=1}^n q_i(S_j) \quad (9)$$

where the symbol " \sqsubseteq " shows that q_i is contained in Q .

Example 1: Suppose a user wants to retrieve the titles of all the books written by "J. D. Ullman". The user poses a query over the GIS , $(Q(GIS))$, which is decomposed into

sub-queries q_1 and q_2 over the data sources $(\bigcup_{i,j=1}^n q_i(S_j))$.

Basically, the query, $(Q(GIS))$, is processed as follows:

- Send a source query q_1 to Student-Books to retrieve the titles of all the books authored by "J. D. Ullman", which is written as:

```
SELECT Title X
FROM S1.Books.Title X
WHERE Author.Name = "J. D. Ullman"
```
- Send a source query q_2 to Convenient-Books to retrieve the titles of all the books authored by "J. D. Ullman" written as:

```
SELECT Title X
FROM S1.Books.Title X
WHERE Author = "J. D. Ullman"
```
- The FROM clause binds the variable X to each node specified in the path [$Books.Title$]. Notice the differences in the constraints in the WHERE clauses of q_1 and q_2 because of the differences in the structures of S_1 and S_2 . The query results in the element (u, l_i, v) , such that $u \in Root$ and $v \in Terminal-Node$. The computed union of the outputs of q_1 and q_2 provide the answer to the user query.

The integration mechanism produces a set of pre-computed views, GIS , that aggregates the local schemas, and could be queried by different users across diverse platforms. The query process relies on such views to derive correct answers. In $Del-G$, a regular expression represents a query that exploits path knowledge to retrieve all pairs of nodes connected by a path. A query Q expressed in terms of the GIS is decomposed into sub-queries q_i expressed in terms of the local schemas $\bigcup_{i=1}^n S_i$. The solution for Q is the computed union of all the partial solutions of q_i .

7. Related Work

In this section, we examine previous research efforts that are tangentially related to our work. Spaccapietra and Parent [13] identify correspondence assertions for declaring semantic relationships between objects to eliminate naming and semantic conflicts among data sources. Chen [5] introduces derivation assertion to accommodate ‘more heterogeneity’. Larson *et al.* [6] present definitions of attribute equivalence, which are used to identify and specify the relationship between a pair of attributes. Yu *et al.* [16] develop a scheme to semi-automatically determine attribute relationships.

However, we note that these approaches to identifying correspondence assertions [5, 6, 13, 16] depend on significant manual input from the users and the ingenuity of the DBAs. We argue that it would be exceedingly difficult for users / DBAs to identify, declare, and specify the assertions that exist between objects, especially a mapping between two different attributes, for all attributes in the local data sources. Furthermore, specifying mappings between all possible pairs of attributes is time consuming, tedious, and error prone.

8. Conclusion

The Web is an amalgam of a broad spectrum of data that emanate from disparate information sources. However, the Web fails to integrate and provide easy access and transparency to integrated e-commerce related data on the Web, which is necessary for successful e-commerce solutions. Data integration provides a global view of data across organizational boundaries, while retaining the autonomy of the data sources. Regrettably, existing integration systems require substantial manual input from the users. We argue that task of manually

identifying assertions for all attributes in the local databases is indeed enormous. In this paper, we proposed an efficient integration algorithm that dynamically identifies assertions, without the costly intervention of the DBA. We leveraged a common ontology and a flexible semistructured data model to make explicit the content of data and express conceptualizations in a natural way. We exploit set theory and logic to provide a measure of formalism. Formally specifying an integration system for e-commerce transactions provides a clear understanding of the system and reduces the complexity of the integration process. We provide example to illustrate how to query the *GIS*. Future plans include the specification of the entire integration process, and to show formally how the algorithm resolves different heterogeneities.

References

- [1] Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J. L., “The Lorel Query Language for Semistructured Data”, *International Journal on Digital Libraries*, Vol.1, No. 1, 1997, pp. 68-88.
- [2] Buneman, P., Davidson, S., Hillebrand, H., and Suciu, D., “A Query Language and Optimization Techniques for Unstructured Data”, *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal Canada, 1996, pp. 505-517.
- [3] Ben-Ameur, W., and Kerivin, H., “New Economical Virtual Private Networks”, *Communications of the ACM*, Vol. 46, No. 6, June 2003, pp. 69-74.
- [4] Batini, C., Lenzerini, M. and Navathe, S., “A Comparative Analysis of Methodologies for Database Schema Integration”, *ACM Computing Surveys*, Vol. 18, No. 4, December 1986, pp. 323-364.

- [5] Chen Y., "Integrating Heterogeneous OO Schemas", *Journal of Information Science and Engineering*, Vol. 16, 2000, pp. 555-591.
- [6] Larson, J. A., Navathe, S. B., and Elmasri, R., "A Theory of Attribute Databases with Application to Schema Integration", *IEEE Transaction on Software Engineering*, Vol. 15, No. 4, 1989, pp. 449-463.
- [7] Motro, A., and Buneman, P., "Constructing Superviews", *Proceedings of 8th ACM International Conference on Management of Data*, Ann Arbor, Michigan, April 1981.
- [8] Menascé, D. A., Barbará, D., and Dodge, R., "Preserving Quality of Service Through Self-Tuning: A Performance Model Approach", *Proceedings of the 3rd ACM Conference on Electronic Commerce (EC'01)*, Tampa, Florida, October 14- 17, 2001, pp. 224-234.
- [9] Mannino, M. V., and Effelsberg, W., "Matching Techniques in Global Schema Design", *Proceedings of the IEEE COMPDEC Conference*, LA., California, 1984, pp. 418-425.
- [10] Papakonstantinou, Y., Garcia-Molina, H., and Widom, J., "Object Exchange Across Heterogeneous Information Sources", in Yu, P. S., and Chen, A. L. P., Editors, 1995, *11th International Conference on Data Engineering*, March 06-10, 1995, Taipei, Taiwan, pp. 251-260.
- [11] Scheurer, T., *Foundations of Computing: System Development with Set Theory and Logic*, Addison-Wesley Publishers, Reading, MA., 1994.
- [12] Seo, D., Lee, D., Lee, K., and Lee, J., "Discovery of Schema Information from a Forest of Selectively Labeled Ordered Trees", *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, May 1997, pp. 54-59.
- [13] Spaccapietra, S., and Parent, P., "View Integration: A Step Forward in Solving Structural Conflicts", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 2, 1994, pp. 258-274.
- [14] Templeton, M., Brill, D., Dao, S. K., Lund, E., Ward, P., Chen, A. L. P., and MacGregor, R., "Mermaid: A Front End to Distributed Heterogeneous Databases", *Proceedings of IEEE*, May 1987, pp. 695-708.
- [15] Tygar, J. D., "Atomicity in Electronic Commerce", *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96)*, New York, May, 1996, pp. 8-26.
- [16] Yu, C., Sun, W., Dao, S., and Keirse, D., "Determining Relationships among Attributes for Interoperability of Multi-Database Systems", Y. Kambayashi and M. Rusinkiewicz (Eds.): *First International Workshop on Research Issues on Data Engineering: Interoperability in Multidatabase Systems*, April 7-9, 1991, Kyoto, Japan, pp. 251-257.
- [17] Zhong, M., "A Faster Single-Term Divisible Electronic Cash: ZCash", *Journal of Electronic Commerce Research and Applications*, Vol. 1, 2002, Elsevier Science, B.V., pp. 331-33.