ALGEBRAIC SIGNATURES FOR SCALABLE WEB DATA INTEGRATION FOR ELECTRONIC COMMERCE TRANSACTIONS

Chima Adiele
Department of Computer Science
University of Manitoba
adiele@cs.umanitoba.ca

Sylvanus A. Ehikioya
Nigerian Communications Commission
Abuja, Nigeria
ehikioya@ncc.gov.ng

ABSTRACT

E-commerce transactions involve the access of data from enterprise-wide information. Providing enterprise-wide information is becoming increasingly difficult because of problems inherent in identifying semantic equivalent data. Web data integration consolidates and standardizes Web data, thus making Web data readily available to meet e-commerce needs. Integrating data from different sources involves combining concepts and knowledge of such data into a global view, so users can access such data without relying on the complex organizational structures of the underlying data sources. Unfortunately, most existing efforts involve significant manual input and adopt *ad hoc* approaches to data integration, ignoring the theoretical foundations of, and the necessary formalisms to specify, the integration process. In this paper, we leverage algebraic signatures to specify components of the integration system. We present an algebraic data model that reduces the topological structure of Web data sources to regular expressions. We define algebraic operators and functions to manipulate objects in the algebraic model. Central to the integration process is a filter mechanism that recognizes a "regulated schema" so that all the participating schemas are guaranteed to be in the same format. We also show that our approach guarantees a scalable and correct integration system.

Keywords: Web data integration, E-Commerce, Algebraic Signatures, Semistructured Data Model

1. Introduction

The Internet offers a virtual marketplace where buyers and sellers exchange goods and services. E-commerce involves business activities that deal with the exchange of values for goods and services on the Internet where all transactions are driven electronically. E-commerce, by nature, is a "self-service model" that allows the customer to locate a merchant's Web site, get product information, initiate product order and make payments. Locating products or businesses on the Web is an important integral aspect of e-commerce transactions. Therefore, participants have to contend with information overload on the Web, as it is the responsibility of the users to locate and interpret Web data.

Regardless of the increasing volumes of e-commerce sites, e-commerce transactions come with some challenges. Maamar (2003) identifies the following challenges that affect e-commerce operations: 1) The user has to identify the relevant Web sites to access catalogs; 2) It is necessary for the user to understand how the Web sites operate; 3) The user has to specify his / her needs according to the characteristics or terminology of the Web sites; and 4) there is also security concerns in dealing with sensitive information, such as user's account information. The issue of secure e-commerce infrastructure has been extensively addressed in the literature [Bryant and Colledge, 2002; Evincible, 2003; Tygar, 1996; VeriSign, 2003]. However, data infrastructure (identifying, interpreting and standard format) for e-commerce data on the Web is largely ignored. Web users, and in particular, e-commerce actors desire an environment where data is transparent (but appears visible) to all the participants. Therefore, a model that makes e-commerce data on the Web visible to all the actors is desirable. Web data integration provides a framework to consolidate and standardize Web data, thus making Web data readily available. We present a scalable integration system that is capable of integrating Web data sources. Our design philosophy hinges on three pivotal components - suitable input, appropriate integration methodology and transparent integration output. We believe that applying appropriate integration methodology based on sound theoretical foundation to a set of well-formed input schemas will generate the desired integration output that is transparent.

The model we propose leverages algebraic signatures with sound integration principles and algorithm to guarantee scalability and correctness of the integration system. We define algebraic operators and functions to manipulate objects in the algebraic model. Central to the integration process is a filter mechanism that recognizes a "regulated schema" so that all the participating schemas are guaranteed to be in the same format. We show that the process of recognizing a well-formed schema is decidable. Our model allows the autonomy of data sources, as every data source is represented in a *Del-G* model. *Del-G* is an acyclic digraph that encapsulates in it structural information of every object it represents. In our model, the integration methodology reduces to simple matching of terms based on semantic names since every participating data source is translated to a regulated schema, and every term in the local schemas is drawn from a common ontology. We design an efficient integration algorithm that automates the integration process, thus freeing the user from the burden of understanding the intricacies of the underlying data sources. We formally specify the integration components to reduce system's complexity and provide a clear understanding of the overall integration system. The main task involved in adding new sources is to ensure that the local schemas are well formed. Therefore, the model we propose scales over multiple sources. Finally, the output of the integration methodology is a transparent integration result called global integrated schema (GIS). The GIS is a correct and transparent representation of the local schemas. This paper is significant in the following ways:

a) Formally specifying an integration model for e-commerce transactions provides a clear understanding of the model, reveal ambiguities, incompleteness, and contradictions in the informal definition, and thus permit the correctness verification of the integration components.
b) Algebraic signatures provide a formal foundation for establishing the correctness of the integration model. In particular, the use of algebraic operators and functions to manipulate objects in the algebraic model, abstracts away irrelevant portions of the informal definition, and emphasizes systems' functionalities that precisely specify the behaviour of the model.
c) Our integration algorithm automates the integration process. Automating the integration process removes the burden of identifying and interpreting e-commerce data from the user.
d) Our model scales over multiple data sources because reducing the complexity of the integration process enhances model's scalability.

An algebraic signature is an implicit property-based formal framework that implicitly expresses operations by their algebraic equivalences. Our interest is on *what* system's operations are required to accomplish, rather than *how* the task is to be carried out. We use set notations as in [Scheurer, 1994] to describe structural components, and predicate logic to describe pre- and post-conditions for any requirement. Both the pre-conditions and post-conditions are given 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. A general form of a quantified statement can be written in one of the following forms:

a) <quantifier><declaration(s)>●<predicate>
b) <quantifier><declaration(s)>|<constraint>●<predicate>

Two frequently used kinds of quantifiers are universal ($\forall$) and existential ($\exists$). The constraint is a condition that the declaration must satisfy. The symbols "|" and "●" are part of the syntax, which 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 $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\Rightarrow$ (conditional), and $\Leftrightarrow$ (bi-conditional). Truth tables for the logical connectives are available in most discrete mathematics textbooks. Other symbols used in this paper are explained in context. A glossary of symbols is provided in Appendix A.

The remaining part of this paper is organized as follows. Section 2 examines related background materials and integration challenges. Section 3 examines the need for a common ontology. A common ontology elucidates the content and meaning of Web data to support the information needs of users of Web information. In Section 4, we use algebraic signatures to represent a flexible semistructured data model, called *Del-G*. This representation enhances the reduction of *Del-G* to regular expressions. Section 5 defines the integration process, thus bringing out the basic components the integration process. We describe the input process, an important integral part of the integration process in Section 6. We note that having a suitable input schema is a *sine qua non* to a successful Web data integration. Therefore, we show that a well-formed input schema is recognizable and the process of recognizing the schema is decidable. Section 7 discusses the integration mechanism and Section 8 examines the output process. We also discuss our integration algorithm, and provide necessary formalisms for the integration process. Finally, we conclude in Section 9, and provide insight into future work.

2. Background

Data integration is a problem that has attracted considerable attention from the research / industry communities. Several academic prototypes and industrial integration systems have been developed to integrate data with tangential success. We discuss a selection of these integration systems. Garlic [Haas *et al*, 1997; Haas *et al*, 1999] is a wrapper / mediator-based system, designed to address large information systems. Garlic derives its strength from powerful wrappers and robust query processing mechanism capable of handling complex operations. In Garlic, the component export schemas are tightly integrated within an object-oriented data model and the system optimizes and executes queries over diverse data sources in an object-extended version SQL. Garlic ignores heterogeneity in schemas, but handles differences in query capabilities using a flexible query optimizer. The Stanford-IBM Manager of Multiple Information Sources (TSIMMIS) [Chawathe *et al*, 1994] focuses on the development of tools to facilitate integration of heterogeneous information sources. The system propagates all schemas of the data sources' wrappers to a global view. TSIMMIS is a query-centric system that selects a set of queries and provides a procedure to answer each query from the set of queries using available data sources. Queries on the global view are mapped into views or queries on the data sources using query languages or logical rules. TSIMMIS does not believe on automation.

Industrial systems [Bukhres *et al*, 1993; Sheth and Karabatis, 1993] adopt a more *ad hoc* approach to integration. Sheth and Karabatis (1993) describe a technique that is based on specifying dependencies among databases so that the system can guarantee consistency among databases in a multi-database system. This approach is amenable to industrial needs because custom coding, although costly and laborious, is relatively simple. It is also relevant to the transaction management problem as many global transactions are carried out by consistency enforcement. Global consistency of data sources is also the driving force behind InterBase [Bukhres *et al*, 1993]. InterBase uses a flexible transaction manager that ensures global consistency of data sources to provide a framework for integrating different data sources. InterBase provides interfaces to the individual data sources for access data. InterBase is an example of a practical system for industrial applications, despite the *ad hoc* integration approach that depends on substantial manual input.

There are also domain specific efforts to standardize documents and messages for e-commerce applications based on eXtensible Markup Language (XML). An Electronic Business using the eXtensible Markup Language (ebXML) [W3C, 2002] is a domain-independent standardization effort that aims at developing a domain and application-independent standard for exchanging business data. One problem of ebXML is that it provides only a high-level specification of documents and messages (i.e. only tags and terminology is defined) but does not define the structure of business documents. Such structure has to be defined by the business partners or by a particular industry. BizTalk [Microsoft, 2000] allows data to be exchanged between systems using standardized XML schemas. The Metadata Interchange Specification (MDIS) [Metadata, 1997], developed by a consortium of database and software experts, is another emerging standard for specifying and exchanging metadata. The Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT)[1] is a United Nations Economic Commission for Europe (UNECE) initiative to specify syntax rules for the preparation of messages to be exchanged between partners. EDIFACT has been used among big partners who set the standards for several years for B2B e-commerce. A major problem with all these approaches is that they define schemas for data exchange. In particular, Fensel (2001) notes that the EDIFACT standard is rather procedural and difficult, thus making the programming of business transactions expensive, error prone, and laborious to maintain. Industry efforts, including BizTalk, Standard Interchange Language (SIL) [Uniform, 1999], and Electronic Data Interchange (EDI) [Swatman and Swatman, 1991; Chau, 2001], which use standardized structures and formats as a vehicle for information exchange is not appealing because it lacks the needed flexibility to accommodate legacy systems and the corporate databases of most enterprises are mainly in legacy systems.

2.1 Challenges of Integrating Heterogeneous Data Sources

The task of integrating data from heterogeneous data sources is difficult because of the differences in the underlying data sources. Differences arise between different data sources because of conflicting representations of two or more schemas. These differences arise because of the designers' perception, different information needs, or using different tools to express such perceptions. Indeed, two designers modeling the same problem space, or even two overlapping spaces under consideration, may not necessarily describe the real world state (RWS) of objects[2] in the same way. Different authors [Batini *et al*, 1986; Lee *et al*, 1999; Sheth and Larson, 1990] have classified schema heterogeneity in different ways at varying degrees, which sometimes coincide, overlap, or differ. We classify schema heterogeneity into two broad groups – semantic and structural heterogeneity.

[1] EDIFACT (Available at): http://www.unece.org/trade/untdid/welcome.htm

[2] Larson *et al*. (1989) defined real world states of an object *A*, denoted RWS(*A*), as the set of instances of object *A* at a given time.

Semantic Heterogeneity - Semantic heterogeneity gives rise to naming conflicts. Naming conflicts may be a problem of synonym where different concepts have the same name, or homonym where the same concept is given different names. Naming conflicts may occur in a situation where data in different systems are given different interpretations due to the designers' different perceptions of the same set of real world objects. For instance, a schema may have an "Employee" class, while another schema may have a more restricted "SD-Employee" class (a group of employees in the sales department). Naming conflicts are resolved using a common ontology. A common ontology eliminates multiple views of objects as every object in the ontology has a unique name, and objects with the same name represent the same RWS.

Structural Heterogeneity - Structural heterogeneity refers to a situation where designers use different data models (including structured, semistructured and unstructured data models), formats, and constructs to represent the same real world objects. Subtypes of structural heterogeneity include behavioural heterogeneity, data model heterogeneity, interschema properties, key conflicts, and data type heterogeneity. A detailed discussion of these heterogeneities can be found in [Adiele, 2004].

Albeit, an efficient integration system is expected to resolve heterogeneities without user intervention, as the way and manner an integration system resolves conflicts greatly enhances its applicability. A transparent integration system insulates the user from the different data structures and organizational characteristics of the individual data sources.

3. The Need for a Common Ontology

The Web offers an amalgamation of data from different domains to its multiple users, but fails to provide the requisite environment for all actors to exchange information due to inconsistent behaviours amongst the participants [Beneventano *et al*, 2004]. To resolve the problem of multiple understanding of real world state of objects among numerous users of Web information, and enhance data sharability, the semantic content of data need to be made explicit and represented. Ontology provides a framework for participants to speak a common language and hence understand themselves. The need for increased automation of the integration process demands greater explicit representation of the semantic content of local schemas. The adoption of a common standard is not new, as society in general operates on common standards. For example, it be difficult to control traffic, and risky to drive on our roads without appropriate road traffic regulations and standards. A common ontology facilitates the capture of the intended meaning of local schemas. The adoption of a common ontology by all participating data sources enhances interoperability without semantic ambiguities. The data sources also retain autonomy and display high degree of flexibility because they are not tied to a particular data model. 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,

$$\mathrm{CO} \mathrel{\widehat{=}} \{\tau_i\colon \mathbb{P}_1\mathrm{LABEL} \bullet \forall i, j\colon \mathbb{N} \mid i \neq j \bullet \tau_i \neq \tau_j\} \tag{1}$$

A semantic object represents a data element together with its 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. Semantic objects are organized hierarchically so that adjacent complex objects are related by either IS-A or HAS-A relationships. We represent IS-A relationship with a "," and HAS-A relationship with a ";". Formally,

$$\begin{aligned} &\mathrm{SemObj} ::= \langle \mathrm{CT}\rangle;(\mathrm{AT})^{*} \\ &\quad \mathrm{CT} ::= \langle \mathrm{CT}\rangle \mid \langle \mathrm{CT}\rangle;(\mathrm{CT})^{*} \mid (\mathrm{CT})^{*}\,(\mathrm{AT})^{+} \end{aligned} \tag{2}$$

Complex type (CT) represents complex objects, and atomic type (AT) represents atomic objects. We denote the occurrence of an object zero or more times with " * " and the occurrence one or more times with " $^{+}$". A comma delimits each occurrence of an object and shows IS-A relationship. It is mandatory for all participants in the distributed system to accept the common ontology. The common ontology we adopt does not compromise data model autonomy of participating data sources, as individual data sources only map elements from the data sources to the common ontology.

4. Algebraic Data model

The dynamic and complex nature of Web data requires a flexible data model for its representation. We define a simple algebraic data model, we call *Del-G*, which is a directed edge-labelled graph that encapsulates in it structural information of every object it represents, where each edge is labelled with at most one context label (name). *Del-G* represents data as a set of nodes and edges, similar to [Buneman *et al.* 1997; Papakonstantinou *et al*, 1995; Seo *et al*, 1997]. Simple data models have inherent advantage over complex models in data integration because the operations to transform and merge data are correspondingly simpler in simple models [Batini *et al,* 1986]. Our model has the following features:

1. A unique singleton node, called the root node;
2. Labels are on the edges;
3. The model contains at most one edge between two nodes; and
4. Every element is drawn from a common ontology.
5. Represents unknown elements.

We leverage *Del-G*'s simplicity and the expressive power provided by its features to achieve data integration.

4.1 Formal Foundation

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 databases. In particular, we use the following informal definitions to describe tree concepts, where the elements *u* and *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 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 $N : \mathbb{P}_1$ NODE; E: $\mathbb{P}$ EDGE; and L: $\mathbb{P}$ 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.

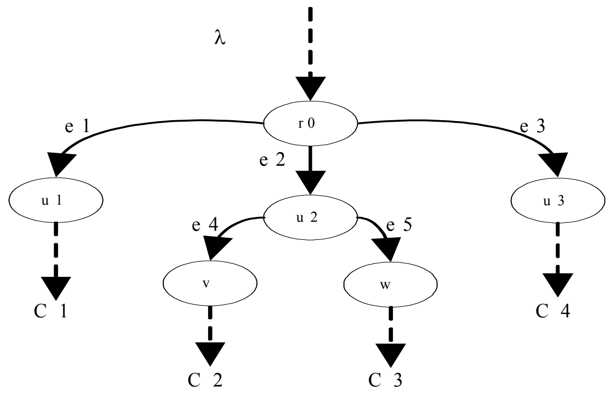


Figure 1: A Representation of *Del-G*

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. We formally define the functions *Source* and *Target* below.

Source, *Target*: EDGE $\rightarrow$ NODE (that is, *Source* : EDGE $\rightarrow$ NODE; *Target*: EDGE $\rightarrow$ NODE)

$\forall e_i$: EDGE $\bullet$ ($\exists u, v$: NODE $\mid u \neq v \wedge (u, v) = e_i \bullet$

$Source(e_i) = u \wedge Target(e_i) = v$)

For example, in Figure 1, $Source(e_4) = u_2$ and $Target(e_4) = v$. We define the function *Parent* to enhance the hierarchical reference of objects. A node u is the parent of another node v, written $Parent(v) = u$, in *Del-G* if there exists an edge e_i such that u is the source node and v is the target node. The parent of a node is traversed before the child node in a pre-order traversal. We give the definition of the function *Parent*.

Parent: NODE $\rightarrow$ NODE

$\forall u, v$: NODE $\mid u \neq v \bullet$

$Parent(v) = u \Rightarrow \exists_1 e_i$: EDGE $\bullet$

$Source(e_i) = u \wedge Target(e_i) = v$

For example, in Figure 1, node u_2 is the $Parent(v)$. A node v is the *Child* of another node u, written $Child(u) = v$ in *Del-G* if the $Parent(v) = u$. We give the definition of the function *Child*.

Child: NODE $\rightarrow$ $\mathbb{P}$NODE

$\forall u$: NODE $\bullet$

$Child(u) = \{v$: NODE $\mid Parent(v) = u\}$

We represent the set of non-terminal nodes as *Complex-Node* and the set of terminal nodes as *Atomic-Node*. A node $u \in$ *Complex-Node* has a child, while a node $v \in$ *Atomic-Node* has no child.

$Siblings(w, v)$ is a boolean function that returns true if w and v have the same direct parent, otherwise it returns false. Formally,

Siblings : NODE $\rightarrow$ $\mathbb{P}$NODE

$\forall u, v$: NODE $\mid u \neq v \bullet$

$Siblings(u, v)$ = TRUE $\Leftrightarrow$

$\exists w$: NODE $\mid (w \neq u \wedge w \neq v) \bullet u, v \in Child(w)$

4.2 The *Del-G* Model

In *Del-G*, every edge is associated with a label. Recall, an edge is an ordered pair of nodes, which represent objects, and every object has a label given by the function:

NodeLabels: NODE $\rightarrow$ LABEL

To synchronize the label of an object with the formalism of *Del-G*, we associate the label of an edge to the *Target*. We give the signature and definition of a labelling function, *EdgeLabels* that maps every edge to a label.

EdgeLabel: EDGE $\rightarrow$ LABEL

$\forall e$: *Edge* $\mid \exists l$: LABEL $\wedge$

$EdgeLabel(e) = l \Leftrightarrow NodeLabel(Target(e)) = l$

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 from 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. $\varphi = \bigcup (Root, Complex\text{-}Node, Atomic\text{-}Node) \subseteq N \mid$

$\bigcap(Root, Complex\text{-}Node, Atomic\text{-}Node) = \varnothing$;

2. $r_0 \in Root$ is a special node in *Del-G*, called the root node;
3. λ: LABEL is a special label called the entry point label; and
4. $Val: N \to N \times CV$.

A *Schema* also satisfies the following conditions:

a) terminal nodes have no outgoing edges;
b) the root node has no incoming edge, but every schema has an entry point label that is associated with the root node; and
c) 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.

$$\begin{aligned}
&\forall S\text{: SCHEMA} \bullet S = (\varphi, r_0, \lambda, Val) \Leftrightarrow \\
&\quad \varphi = \bigcup (Root, Complex\text{-}Node, Atomic\text{-}Node) \mid \\
&\quad \bigcap (Root, Complex\text{-}Node, Atomic\text{-}Node) = \varnothing \wedge \exists r_0\text{: } Root \bullet NodeLabel(r_0) = \lambda \;\wedge \\
&\qquad \forall e\text{: EDGE} \bullet (\exists u, v\text{: NODE} \mid u \neq v \wedge u, v \in \varphi \bullet Source(e) = u \wedge Target(e) = v) \wedge \\
&\qquad (\exists l\text{: LABEL} \bullet EdgeLabel(e) = l) \wedge \\
&\qquad\quad Val(Target(e)) \in CV \Leftrightarrow Target(e) \in Atomic\text{-}Node)))
\end{aligned} \tag{3}$$

Example 1. We show in Figure 2 an example of a *Del-G* schema with information about authors of books.

Author [Name, Birthday, Book [Title, ISBN]]

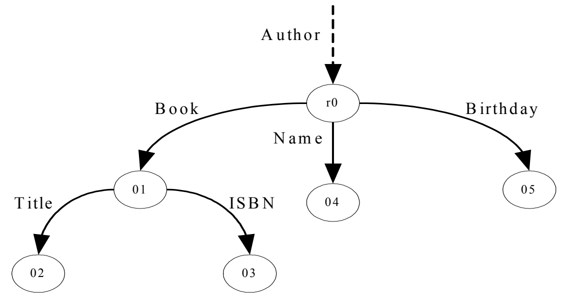


Figure 2: Graphical Representation of Semistructured Data in *Del-G*

Author is the entry-point label associated with the object being described, and its value is a sequence of sub-objects enclosed in the square bracket. These sub-objects may include atomic objects as in [Name] or complex objects as in [Book [Title, ISBN]], and the values of atomic objects are drawn from the set of constant values *CV*.

A semistructured data contains the sequences of labels along the paths that can be found in the edge-graph. In the above representation, the entry point label "Author" is associated with the root node r_0; (01) $\in$ *Complex-Node*, while (02, 03, 04, 05) $\in$ *Atomic-Node*. Additional functions and terminologies are defined in context.

In the *Del-G* model, as in OEM, 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 *Del-G* over the other models [Buneman *et al*, 1997; Papakonstantinou *et al*, 1995; Seo *et al*, 1997] 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.

5. The Integration Process

The integration process addresses data integration challenges by leveraging the common ontology and semistructured data model to map elements from the local schemas to the global integrated schema. Given two local schemas as input, where elements from the schemas are drawn from the common ontology and represented with *Del-G* model, the integration process generates a global integrated schema as output. An integral part of the process is an integration algorithm that automatically identifies assertions, determines attribute relationships, and generates correspondence rules without the costly user intervention. Therefore, the major task in this phase is to dynamically resolve heterogeneities that exist between local schemas with a view to producing a global integrated schema that guarantees data transparency across data sources. A formal definition of the integration process follows.

Definition 4. The *integration process* is a 4-tupple (S_{LS}, CO, Υ_{GIS}, GIS) where:

1. $S_1, S_2 \in S_{LS}$ are local schemas to be integrated such that every element in S_1 and S_2 is drawn from the common ontology CO. S_1 and S_2 are the input to the systems, and it is required that the input be presented in the right format.
2. CO is a common ontology that provides standardization for the terms used;
3. Υ_{GIS} is an integration generation function (Υ_{GIS}) that generates the GIS from the local schemas using correspondence assertions and correspondence rules. Υ_{GIS} is an integral part of the integration methodology that processes a given input to generate the desired output.
4. GIS is the global integrated schema over all the local schemas. The GIS is the output of the integration process.

The above definition presents the local schemas $S_1, S_2 \in S_{LS}$ as input to the integration process. Elements from the local schemas are mapped to the common ontology to ensure that elements with the same semantic names actually represent the same real world state. In addition, all the input schemas is represented with the *Del-G* model so that they can communicate uniformly. The system we model allows autonomy of user input, and therefore requires that all local schemas must be guaranteed to be in the same format. The main task of the input process is to present all the participating local schemas in the same format. One of the ways to ensure that all the participating local schemas are in the same format is to provide a theoretical framework that supports such claims. Providing this theoretical framework is vital to guarantee a "suitable input". A suitable integration methodology allows a measure of autonomy. Therefore, the task of the integration methodology is to design an algorithm that automatically integrates the local schemas into *GIS*. Finally, the required output, the *GIS* should correctly represent all the local schemas. In the sections that follow we discuss components of the integration process in detail and provide the necessary formalisms.

6. Input Process

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 objects. The aim of the first phase is to ensure that all participating data sources are represented with a *Del-G* schema. Determining the "well-formedness" of a given local schema is crucial to the integration mechanism. In particular, Sheth and Larson (1990) observe that "unless the schemas are represented in the same model, analyzing and comparing their schema objects is extremely difficult." Central to the integration process is a filter mechanism that recognizes a "regulated schema" so that all the participating schemas are guaranteed to be in the same format. We, therefore, provide a theoretical foundation to show that the filter mechanism is capable of identifying regulated schemas in a finite number of steps.

First, we give the following definitions to simplify our exposition. We define a regular expression in the context of a *Del-G* schema to enable us establish the fact that an FSA accepts a *well-formed Schema*.

Definition 5. A *regular expression R* over the alphabet $\Sigma \mid \Sigma \subseteq CO$ is defined as follows:

i) ε, is a regular expression;
ii) $a \mid a \in \Sigma$ is a regular expression;
iii) if R_1 and R_2 are regular expressions, then $(R_1 \cup R_2)$ is a regular expression;
iv) if R_1 and R_2 are regular expressions, then $(R_1.R_2)$ is a regular expression;
v) if R_1 is a regular expression, then (R_1^*) is a regular expression;
vi) no expression is a regular expression unless it is obtained from (i) – (v).

Let $L(R)$ be the language associated with the regular expression, R, over the alphabet, Σ. Then $L(R) \subseteq \Sigma^* \models$ $L(R) \subseteq CO$. We recursively define $L(R)$ as follows:

a) $L(\varepsilon) = \{\varepsilon\}$;
b) if $a \in \Sigma$, then $L(a) = \{a\}$;
c) $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$;

d) $L(R_1.R_2) = L(R_1).L(R_2)$;
e) $L(R_1^*) = L(R_1)^*$;
where L^* is the reflexive and transitive closure of the language L.

Definition 6. A *Path-Label* (*pl*) in *Del-G* is a sequence of labels $(\lambda.\ l_1.\ \ldots\ .\ l_n) \subseteq CO$ of a path, each delimited by a period, such that λ is the entry point label, and the labels l_i correspond to edges e_i $(1 \le i \le n)$, and $Target(e_n) \in$ *Atomic-Node*. The pseudo edge, e_0, is associated with the entry-point label λ.

Definition 7. A *Data-Path* (*dp*) in *Del-G* is an alternating sequence of labels on edges, and nodes delimited by periods of the form $(\lambda r_0.l_1o_1.\ \ldots.\ l_no_n)$, such that a path of n edges $(e_1.e_2.\ \ldots.\ e_n)$ could be traversed in a predefined order through $(n+1)$ nodes $(r_0.o_1.\ \ldots\ o_n)$. In addition, every *dp* in *Del-G* satisfies the following conditions.
1. λ is the entry point label.
2. l_1 is the label of edge e_1 such that $Source(e_1) = r_0$ and $Target(e_1) = o_1$.
3. l_n is the label of edge e_n such that $Source(e_n) = o_{n-1}$ and $Target(e_n) = o_n \wedge o_n \in$ *Atomic-Node*.

The basic types [PATH-LABEL, DATA-PATH] represent a path label and a data path, respectively. The following constraints hold for path label and data path.

$$\forall pl_i, pl_j\text{: PATH-LABEL} \bullet i \ne j \Rightarrow pl_i \ne pl_j \qquad (4)$$
$$\forall dp_i, dp_j\text{: DATA-PATH} \bullet i \ne j \Rightarrow dp_i \ne dp_j$$

Example 2. To illustrate our definitions we consider the schema of a *Del-G* object *O* represented in Figure 2. For example, (Author.r_0.Book.01.Title.02) is a data path of the schema of Figure 2, where the edges labeled (Author, Book, Title) could be traversed through the nodes (r_0, 01, 02), and “Author” is the entry point label. “Book” is the label of edge e_1 such that $Source(e_1) = r_0$ and $Target(e_1) = 01$. “Title” is the label of edge e_n such that $Source(e_n) = 01$ and $Target(e_n) = 03 \wedge 03 \in$ *Atomic-Node*. The data paths and corresponding path labels is given in Table 1.

Table 1: Data Paths and Path Labels of Figure 2

Data paths	Corresponding Path Labels
dp_1 = Author.r_0.Book.01.Title.02	pl_1 = Author.Book.Title
dp_2 = Author.r_0.Book.01.ISBN.03	pl_2 = Author.Book.ISBN
dp_3 = Author.r_0.Name.04	pl_3 = Author.Name
dp_4 = Author.r_0.Birthday.05	pl_4 = Author.Birthday

The *Del-G* of Figure 2 can be completely represented by the union of *Data-Paths*, $\bigcup_{i=1}^{n} dp_i$. A *k-dp Del-G* is a *Del-G* where the cardinality of the set *Atomic-Node* = *k*. A *k-dp Del-G* can be constructed by concatenating a set of *k* data paths $\{dp_1, dp_2, \ldots, dp_k\}$ that are not prefix of each other. Notice that for every unique *Data-Path* dp_i there exists a corresponding unique *Path-Label* pl_i. Therefore, we can represent a *Del-G* structure with *Path-Labels*, which are regular expressions. Next, we define a well-formed schema in the context of regular expressions.

Definition 8. A *well-formed Schema*, S^{Wf}, for the schema of an object *O* is a 4-tuple (X_n, X_l, PL, DP) where
1. X_n is a finite set of nodes;
2. X_l is a set of labels;
3. PL is a set of path labels pl_i;
4. DP is a set of data paths dp_i.

In addition, S^{Wf} satisfies the following conditions.
a) for each regular expression $(pl_i \in PL)$ there exists some edge labels in X_l that instantiate the variables of S^{Wf};
b) for each data path $(dp_i \in DP)$, there exist some sequence(s) of nodes and edges in S^{Wf} such that each sequence is in the language $L(dp_i)$;
c) every node (excluding the root node) has at most one parent. $Parent(r_0) = \varnothing$;
d) no two siblings in S^{Wf} are the same.

A formal definition of conditions c) and d) follows.

$$\begin{aligned}&\forall v\text{: NODE} \bullet (\exists S\text{: SCHEMA} \wedge \exists u, w\text{: NODE} \mid u, w \in S \bullet\\&\quad (Parent(v) = u \wedge Parent(v) = w \Rightarrow u = w) \wedge \qquad (5)\\&\qquad \forall a, b\text{: NODE} \mid a \ne b \wedge Siblings(a, b) \bullet (\exists l_i, l_j : \text{LABEL} \bullet\\&\qquad\quad NodeLabel(a) = l_i \wedge NodeLabel(b) = l_j \Rightarrow l_i \ne l_j)\end{aligned}$$

A path label pl_i connects labeled edges to nodes in the form: $a \xrightarrow{pli} b$ where a, $b \in X_n$ and pl_i is a regular expression over X_l.

Definition 9. A *regulated schema* for an object O is a well-formed schema such that $(\forall l : \text{LABEL} \mid l \in CO \bullet (\exists e : \text{EDGE} \bullet EdgeLabel(e) = l))$.

Definition 10. A *filter mechanism* is an accepting device, which either accepts a local schema if it satisfies the accepting conditions, or rejects otherwise. The accepting conditions are as follows:

1. Every local schema must have an entry point label that is associated with the root node.
2. Every element in the local schema (excluding the root) has at most one parent.
3. Every name (label) in the local schema is also in the common ontology (*CO*). If a term A_1 from the local schema is not in *CO*, but $Parent(A_1)$ is in *CO*, then A_1 will be added to the *CO*.
4. The entry point label of every object must be in *CO*.

If these conditions are satisfied, the filter mechanism accepts the local schema, and the accepted local schema then becomes a regulated schema. The "filter mechanism" parses the regulated schema with its associated metadata and mappings. Metadata information includes translation rules from local schema to regulated schema, mappings from local names to ontology names, and systems constraints. *Accept* is a boolean function that accepts a local schema as a regulated schema. The definition of the function *Accept* follows.

$$
\begin{aligned}
&Accept\text{: SCHEMA}\\
&\forall S\text{: SCHEMA} \bullet \exists CO\text{: } \mathbb{P}\text{LABEL} \bullet\\
&\quad Accept(S) \Leftrightarrow (\exists r_0\text{: } Root \mid r_0 \in S \wedge\\
&\qquad \exists \lambda\text{: LABEL} \mid \lambda = NodeLabel(r_0) \wedge \lambda \in CO \bullet\\
&\qquad\quad \forall v\text{: NODE} \mid v \in S \bullet (\exists_1 u\text{: NODE} \mid (u \neq v \wedge u \in S) \bullet Parent(v) = u) \wedge\\
&\qquad\qquad \forall l_i : \text{LABEL} \mid l_i \in S \bullet (\exists l_j : \text{LABEL} \mid l_j \in CO \bullet l_i = l_j) \vee\\
&\qquad\qquad\quad \forall l_i \in S \mid (l_i \neq \lambda \wedge l_i \notin CO) \bullet Parent(l_i) \in CO \Rightarrow\\
&\qquad\qquad\qquad Insert(l_i) \bullet l_i \in CO)
\end{aligned}
\tag{6}
$$

We give the signature of a function *Insert* that inserts a term (name) in *CO*; *Insert*: LABEL $\to$ LABEL.

Recall a filter mechanism that accepts a regulated schema is an accepting device. This implies that it is possible to construct a finite state automaton that accepts a regulated schema since it is known [Sipser, 1997] that a finite state automaton can act as an accepting device under certain conditions. First we define a finite state automaton for S^{Wf}.

Definition 11. A *finite state automaton* (FSA) for a well-formed schema S^{Wf} is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. $Q \mid Q$: NODE is a finite set of states;
2. $\Sigma \mid \Sigma$: LABEL $\wedge\, \Sigma \subseteq CO$ is a finite set of alphabets;
3. $\delta: Q \times \Sigma \to Q$ is a transition function;
4. $q_0 \in Q \mid q_0 \in Root$ is a start state
5. $F \subseteq Q \mid F \subseteq Atomic\text{-}Node$ is the set of final states (accepting states).

Juxtaposing Definition 11 with Definition 10, we make the following deductions that facilitate the construction of a FSA from a well-formed schema.

a) Point 1 in Definition 8 defines the start state.
b) Point 2 specifies that the finite state automaton is deterministic.
c) Point 3 defines a finite set of alphabets, Σ.
d) Point 4 states that $\lambda \in \Sigma$.

Theorem 1. There exists a finite state automaton that accepts a *well-formed Schema*, S^{Wf}.

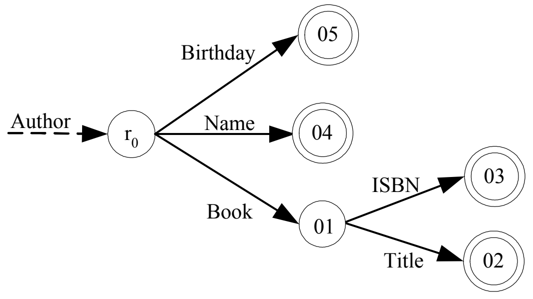
Proof. To simplify our proof, we assume without loss of generality that $\Sigma \subseteq CO = \{a, b\}$, and L is a regular language over Σ. Let R be a regular expression associated with L. But, a *well-formed Schema*, S^{Wf}, in *Del-G* is completely represented by a set of data paths *DP*, and for each data path dp_i, there is a corresponding *Path-Label*, pl_j, such that pl_j uniquely connects the root node, r_0, to a given node u, where $u \in Atomic\text{-}Node$. The proof follows the inductive definition of regular expressions.

Base Case. Suppose R is either ε, a, or b, then, L is either $\{\varepsilon\}$, $\{a\}$, or $\{b\}$. In this case, L is FSA-acceptable because it is finite. Observe that we can have a *Del-G* represented with only one node, r_0, and the entry-point label, λ (where $\lambda = \varepsilon$, a, or b), such that $dp_i = \lambda.r_0$ and $pl_j = \lambda$.

Inductive Case. Suppose R is of the form $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions, then both $L(R_1)$ and $L(R_2)$ are regular and are acceptable by an FSA. But, $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$, so $L(R_1 \cup R_2)$ is acceptable by an FSA.

Suppose R is of the form $(R_1.R_2)$, where $R_1$and R_2 are regular expressions, then both $L(R_1)$ and $L(R_2)$ are regular languages and are acceptable by an FSA. But, $L(R_1.R_2) = L(R_1).L(R_2)$, so $L(R_1).L(R_2)$ is acceptable by an FSA. Finally, suppose R is of the form (R_1^*), where R_1 is a regular expression, then $L(R_1)$ is regular by definition and are acceptable by an FSA. But, $L(R_1^*) = L^*R_1$, so $L(R_1^*)$ is acceptable by an FSA.

Example 3. To illustrate the proof of Theorem 1, we construct an FSA for the *Del-G* schema of Figure 2.



(a)

	Book	Title	ISBN	Name	Birthday
r_0	01			04	05
01		02	03		
02					
03					
04					
05					

(b)

Figure 3: Finite State Automaton (FSA) of a *Del-G* Schema

Figure 3(a) shows the state diagram of the *Del-G* schema of Figure 2. It has six states each representing a node in the *Del-G* schema. When the automaton receives the *Del-G* represented as a set data paths *DP*, it processes each data path dp_i and produces an output. The output is either "*accept*" or "*reject*". The automaton rejects the entire data paths if any data path $dp_i \in DP$ is rejected. Recall that there exists a pseudo edge, e_0, pointing from nowhere to the root node r_0 in a *Del-G* schema. This pseudo edge for Example 2 is associated with the *entry-point label*, "*Author*". Notice that a pseudo arrow pointing from nowhere to a state, r_0, in Figure 3(a) indicates the start state. We attach the *entry-point label*, "*Author*" to this pseudo arrow. Figure 3(b) represents the transition function (δ) for the FSA of Figure 3(a).

Let $M = (Q, \Sigma, \delta, r_0, F)$, where
1. $Q = \{r_0, 01, 02, 03, 04, 05\}$;
2. Σ = {Book, Title, ISBN, Name, Birthday};
3. δ is described as shown in Figure 3(b);
4. $r_0 \in$ Q is the start state; and
5. $F = \{02, 03, 04, 05\} \subseteq$ Q.

If A is the set of all strings that machine M accepts, we say that A is the language of machine M and write $L(M) = A$. We say that M accepts A. From Theorem 1, we deduce that the filter mechanism accepts a regulated schema. Another interesting task is to determine whether the filter mechanism accepts a regulated schema in a finite number of computational steps. That is, can the filter mechanism decide if an input schema is a well-formed schema? Theorem 2 answers this question. To answer this question, we define a Turing machine, M, in the context of a *Del-G* schema.

Definition 12. A *Turing machine* is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are finite sets, and
1. $Q \mid Q$: NODE is the set of states,
2. $\Sigma \mid \Sigma$: LABEL $\wedge \Sigma \subseteq$ *CO* is the input alphabet not containing the special blank symbol ⊔,
3. Γ is the tape alphabet, where ⊔ $\in \Gamma$ and $\Sigma \subset \Gamma$,
4. δ: $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$is the transition function,
5. $q_0 \mid q_0 \in$ *Root* $\wedge$ *Root* $\subseteq$ Q is a start state,
6. $q_{\text{accept}} \mid q_{\text{accept}} \in$ *Atomic-Node* $\wedge$ *Atomic-Node* $\subseteq$ Q is the accept state, and
7. $q_{\text{reject}} \mid q_{\text{reject}} \notin$ *Atomic-Node* is the reject state, where $q_{\text{accept}} \neq q_{\text{reject}}$.

A Turing machine gives a precise definition of an algorithm or a mechanical procedure. It is a simple machine that can decide if a computational problem is solvable in a finite number of steps.

Theorem 2. The process of recognizing a well-formed schema by the *filter mechanism* is decidable.

Proof Idea. Recall from Theorem 1 that an FSA accepts S^{Wf}. Also, from (Definition 10), it can be deduced that a filter mechanism can be represented as a deterministic finite state automaton. The problem reduces to: given an S^{Wf} to find a Turing machine, TM *M*, that accepts it in a finite amount of computational steps. We only need to present a TM *M* that decides S^{Wf}.

M = On input $\langle B, w\rangle$, where *B* is a FSA and $w \mid w$: PATH-LABEL is a string:

1. Simulate *B* on input *w*.
2. If the simulation ends in accept state $u \mid u \in$ *Atomic-Node*, *accept*; otherwise *reject* if it ends in a non-accepting state $u \mid u \notin$ *Atomic-Node*.

Proof. First we examine the input $\langle B, w\rangle$. It is a representation of a FSA *B* together with a string *w*. We can represent *B* as a list of five components (Q, Σ, δ, q_0, F). When *M* receives input, *M* first checks whether it properly represents a FSA *B* and a string *w*. If not *M* rejects the input string *w*. Then *M* carries out the simulation in a direct way, keeping track of *B*'s current state and B's current position in the input *w* by writing the information down on its tape. Initially, *B*'s current state is $q_0 \in$ *Root* and *B*'s current input is the entry point label λ. The transition function δ updates states Q: NODE and position *w*. When *M* finishes processing the last symbol of *w*, *M* accepts the input if *B* is in the accepting state; otherwise *M* rejects the input if *B* is in a non-accepting state.

7. Integration Approach

In Phase 2, we design a schema integration algorithm (Section 7.2) that uses regulated schemas and meta-data as input to identify assertions and resolve heterogeneities. The schema integration algorithm automatically matches terms to generate correspondence assertions. Matching of terms is based on the name and structure of schema elements. Correspondence assertion is a declarative statement, generated by the integration system, which shows the relationship between objects in different data sources at various levels of perception. The integration algorithm also analyzes each assertion to produce formal rules. Formal rules state how to derive constructs to merge into the global integrated schema. These formal rules, also called correspondence rules ($\Re$), are generated automatically and stated using set logic. The schema integration algorithm adopts the ladder approach [Batini *et al*, 1986] to integration so that only two data sources are integrated at a time and the algorithm incrementally integrates a new component schema with an existing immediate result. The specification of correspondence assertions and correspondence rules is an integral part of our research; however, we omit them in this paper without any loss of utility. Interested readers should see [Adiele and Ehikioya, 2004] for details.

7.1 Integration Constraints

To simplify correspondence rules generation process and improve the quality of meta-information provided to the integration system, there is a need to explain certain constraints. These constraints influence the analysis of correspondence assertions and consequently, the generation of correspondence rules by the integration algorithm.

To enable us define these constraints, first, we define the function *Equal*, which is a Boolean function that shows if two context labels are the same. Let L_{EP} be a set of entry-point labels. A pair of elements $(L_1{}^p{}_i, L_2{}^q{}_j)$ are equal, denoted $Equal(L_1{}^p{}_i, L_2{}^q{}_j)$, if: 1) at least one of the elements is an entry-point label and has the same context label as the other element which may not necessarily be an entry-point label; or 2) neither element is an entry-point label and both elements have the same context labels such that $p = q$. Formally,

$$\begin{aligned}
&Equal\text{: LABEL} \times \text{LABEL}\\
&\forall L_1{}^p{}_i, L_2{}^q{}_j\text{: LABEL} \bullet\\
&\quad (\exists\, CS_1, CS_2\text{: SCHEMA} \mid (L_1{}^p{}_i \in CS_1 \wedge L_2{}^q{}_j \in CS_2) \bullet\\
&\quad\; (Equal(L_1{}^p{}_i, L_2{}^q{}_j) = \text{TRUE}) \Leftrightarrow\\
&\quad\;\; (\exists \lambda_1, \lambda_2\text{: LABEL} \mid \lambda_1, \lambda_2 \in L_{EP} \bullet (\lambda_1 = L_1{}^p{}_i \wedge \lambda_2 = L_2{}^q{}_j \wedge L_1{}^p{}_i = L_2{}^q{}_j) \vee\\
&\quad\;\;\; (\lambda_1 = L_1{}^p{}_i \wedge L_2{}^q{}_j \neq \lambda_2 \wedge L_1{}^p{}_i = L_2{}^q{}_j) \vee (\lambda_2 = L_2{}^q{}_j \wedge L_1{}^p{}_i \neq \lambda_1 \wedge L_1{}^p{}_i = L_2{}^q{}_j) \vee\\
&\quad\;\;\;\; L_1{}^p{}_i, L_2{}^q{}_j \notin L_{EP} \wedge p = q \wedge L_1{}^p{}_i = L_2{}^q{}_j))
\end{aligned} \tag{7}$$

A *Terminal-Element* is an edge label where the target of the edge is in the set of atomic nodes. Formally,

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

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

Element Constraints.

1. Integrating a non-terminal element with a non-terminal element - Corresponding elements of the same modeling concepts will be integrated into a similar element. For example, if L_1 is an object in CS_1 and L_2 is an object in

CS_2, and there is a correspondence assertion between both elements, then L_1 and L_2 are integrated into $L \in GIS$ (global integrated schema) as objects. Similarly, if L_1 and L_2 are attributes, they are integrated as attributes. Formally,

$$\begin{array}{l}
\textit{NonTerminal-Element-Constraint}: \text{LABEL} \times \text{LABEL} \rightarrow \text{LABEL} \\
\quad \forall a, b: \text{LABEL} \mid a, b \in \textit{NonTerminal-Element} \bullet \\
\qquad \textit{NonTerminal-Element-Constraint}\ (a, b) = \\
\qquad\quad (c \mid c: \text{LABEL} \wedge c \in \textit{NonTerminal-Element}) \Leftrightarrow \\
\qquad\qquad (a = b \wedge a, b \in \textit{NonTerminal-Element}) \bullet a = c \vDash b = c
\end{array} \quad (8)$$

2. Integrating a non-terminal element with a terminal element - To integrate a non-terminal element with a terminal element that have the same context label, we add both the terminal element and non-terminal element to the integrated schema as two different elements. The source name of the terminal element is appended to its context label so that it references its context schema. Formally,

$$\begin{array}{l}
\textit{Element-Constraint}: \text{LABEL} \times \text{LABEL} \rightarrow \text{LABEL} \\
\forall a, b: \text{LABEL} \mid (a \in \textit{Terminal-Element} \wedge b \in \textit{NonTerminal-Element}) \bullet \\
\quad \textit{Element-Constraint}\ (a, b) = (a', c : \text{LABEL} \mid \\
\qquad (a' \in \textit{Terminal-Element} \wedge c \in \textit{NonTerminal-Element}) \Leftrightarrow \\
\qquad\quad (a = a' \wedge b = c) \bullet \\
\qquad\qquad (\exists\ CS_1, CS_2: \text{SCHEMA} \wedge \exists L_1, L_2: \text{LABEL} \mid (L_1 \in CS_1 \wedge L_2 \in CS_2 \wedge \\
\qquad\qquad\quad \textit{Parent}(a) = L_1 \wedge \textit{Parent}(b) = L_2) \bullet L_1 = L_2 \Rightarrow \\
\qquad\qquad\qquad (\exists\ GIS: \text{SCHEMA} \wedge \exists L: \text{LABEL} \mid (L \in \textit{NonTerminal-Element} \wedge L \in GIS) \bullet \\
\qquad\qquad\qquad\quad L = L_1 \vDash L = L_2 \wedge \textit{Parent}(a') = L \wedge \textit{Parent}(c) = L))
\end{array} \quad (9)$$

Note that $a' \in GIS$ is the same as $CS_1.a \in GIS$ which shows that $a' \in GIS$ is a terminal element that maps a corresponding terminal element in CS_1.

Inheritance Property of Integrated Elements. An object in the integrated schema inherits the descendants of any of its children from the context schema if no further assertion exists between that child and a corresponding object in another context schema that is involved in the integration process. For example, $L_1 \in CS_1 \wedge L_2 \in CS_2$ such that $L_1 = L_2 \Rightarrow \exists L$: LABEL such that $L \in GIS$. Suppose $L_1{}^p{}_i \in \textit{Child}(L_1) \wedge \neg(\exists\ L_2{}^q: \text{LABEL} \mid L_2{}^q \in \textit{Child}(L_2) \bullet L_1{}^p{}_i = L_2{}^q{}_j))$ then $\textit{Descendant}(L_1{}^p{}_i)$ in the context schema will be added into the *GIS* as $\textit{Descendant}(L^q{}_k)$ since no other assertion exists between $L_1{}^p{}_i$ and $\textit{Child}(L_2)$. This integration process leverages the inheritance property to ensure that objects integrated into the *GIS* maintain existing relationships with objects in the context schemas, such that the definitions in the local schemas are still valid.

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

$$\begin{array}{l}
\textit{Inheritance}: CL \rightarrow CL \\
\forall L_1: \text{LABEL} \bullet \exists\ CS_1: \text{SCHEMA} \mid L_1 \in CS_1 \bullet \\
\quad \textit{Inheritance}(L_1) \Leftrightarrow (\exists\ L_2: \text{LABEL}; \exists\ CS_2: \text{SCHEMA} \mid L_2 \in CS_2 \bullet L_2 = L_1 \wedge \\
\qquad (\exists a: \text{LABEL} \bullet a \in \textit{Child}(L_1) \wedge \\
\qquad\quad (\forall b: \text{LABEL} \mid b \in \textit{Child}(L_2) \bullet b \neq a \Rightarrow \\
\qquad\qquad (\exists L: \text{LABEL}; \exists GIS: \text{SCHEMA} \mid L \in GIS \bullet L = L_1) \wedge \\
\qquad\qquad\quad a \in \textit{Child}(L_1) \Rightarrow \exists c: \textit{Child}(L) \bullet a = c \wedge \\
\qquad\qquad\qquad (\forall a_i: \textit{Descendant}(a) \bullet \exists c_i: \textit{Descendant}(c) \bullet a_i = c_i))))
\end{array} \quad (10)$$

Example 4 illustrates elements constraints assumption and the inheritance property.

Example 4. An interesting case occurs when there is an assertion between an object L_1 and an attribute b from CS_1 and CS_2, respectively. To simplify this exposition, let us consider a very simple example, the address of the author(s) of a book modeled differently in two schemas, CS_1 and CS_2.

CS_1 = Author [Name, Address [Number, Street, Code]]

CS_2 = Author [Id, Name, Address]

Observe from Figure 4 that "Address" in CS_1 is modeled as a non-terminal element with three child elements, while "Address" in CS_2 is modeled as a terminal element. An assertion exists between the non-terminal element, "Address" and the terminal element "Address" since their parents are related and both have the same name. Relying on element constraints assumption permits the integration of non-terminal element and its descendants as Address in the *GIS*. For the terminal element, the source name is appended to its context label in the *GIS* (CS_2.Address) so that it references its context schema.

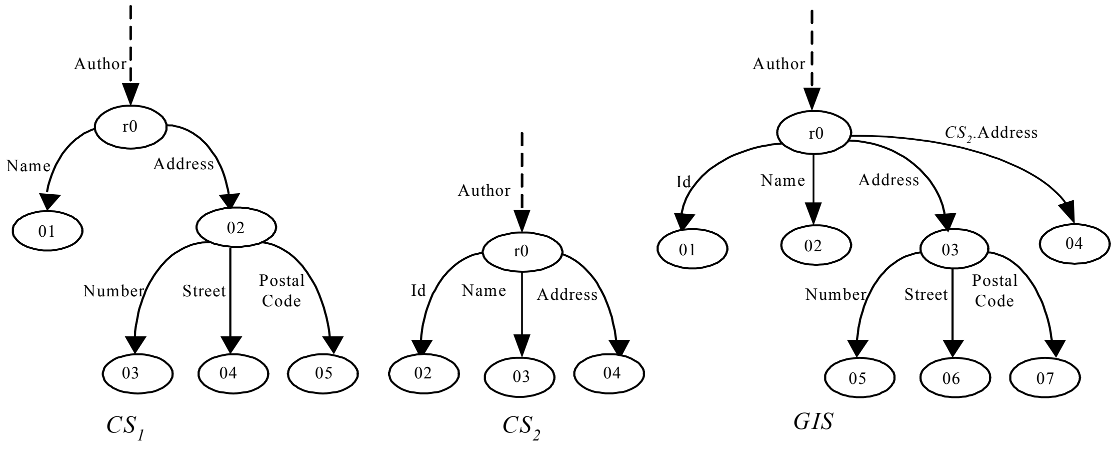


Figure 4: Attribute Integration Example

By the inheritance property of integrated elements, child elements of “Address” in CS_1 are *inherited* into the *GIS* since they do not occur as child elements of “Address” in CS_2. If any of the child elements of “Address” in CS_1 (e.g. CS_1.Address.Street) had any descendant, such descendant would have been inherited into the *GIS* because it is guaranteed not to occur in CS_2.

Data Type Constraints.
1. When types are compatible, the less restrictive of the two data types being integrated is chosen as the type for the *GIS*. For example, a decimal would be 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 *GIS* so that both may be stored.

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 schemas involved in an integration process by taking the least restrictive of the cardinality constraints. In this way, the cardinality constraints of the schemas are subsumed in *GIS*. 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 *GIS* is ([0:1]), ([1:n]), and ([0:n]) respectively.

Terminal Element Constraints. Without prejudice to elements constraints, the assertion between terminal elements is dependent on the assertions between their parents. If there is an assertion between terminal elements that conflicts with an 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 $a \in Child(L_1)$ from CS_1 and terminal element $b \in Child(L_2)$ from CS_2 if and only if $Equal(a, b)$ and $Equal(Parent(a), Parent(b))$.

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

$$\begin{aligned}
&\textit{Terminal-Assertion}: \textit{Terminal-Element} \times \textit{Terminal-Element} \\
&\forall a, b: \textit{Terminal-Element} \bullet \\
&\quad \textit{Terminal-Assertion}(a, b) \Leftrightarrow (\exists\, CS_1, CS_2: \text{SCHEMA} \wedge \\
&\quad \exists L_1, L_2: \text{LABEL} \mid L_1 \in CS_1 \wedge L_2 \in CS_2 \wedge \\
&\qquad Parent(a) = L_1 \wedge Parent(b) = L_2) \bullet \\
&\qquad\; L_1 = L_2 \wedge Parent(a) = Parent(b)
\end{aligned} \tag{11}$$

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* should instead be inherited by their respective parents in the *GIS*, using the inheritance property.

Key Constraints. Key constraints are resolved in the *GIS* by creating a new key that is the union of the two original key elements. Suppose $a \in CS_1$ is a terminal element and a key, and $b \in CS_2$ is also a terminal element and a key, and $Parent(a) = Parent(b)$. This implies that $a, b \in Child(\mathrm{L}) \in GIS$. But a and b cannot be keys in *GIS* because a key element must be unique. The approach is to create a new key (say α) such that $\alpha = (a \cup b)$. Let KEY be the set of terminal elements that are key elements in a given schema. A formal definition of a key constraint follows.

$$\begin{array}{l} \textit{Key-Constraint}: \textit{Terminal-Element} \times \textit{Terminal-Element} \\ \quad \forall a, b: \textit{Terminal-Element} \mid a \neq b \wedge a, b \in \textit{KEY} \bullet \\ \quad\quad \textit{Key-Constraint}(a, b) \Rightarrow (\exists\ CS_1, CS_2: \text{SCHEMA}; \exists \mathrm{L}_1, \mathrm{L}_2: \text{LABEL} \mid \\ \quad\quad\quad (\mathrm{L}_1 \in CS_1 \wedge \mathrm{L}_2 \in CS_2 \wedge Parent(a) = \mathrm{L}_1 \wedge Parent(b) = \mathrm{L}_2) \bullet \mathrm{L}_1 = \mathrm{L}_2 \Rightarrow \\ \quad\quad\quad\quad (\exists GIS: \text{SCHEMA}; \exists \mathrm{L}: \text{LABEL} \mid \mathrm{L} \in \textit{NonTerminal-Element} \wedge \mathrm{L} \in GIS) \bullet \\ \quad\quad\quad\quad\quad a, b \in Child(\mathrm{L}) \wedge \exists \alpha \in Key \bullet \alpha = (a \cup b)) \end{array} \tag{12}$$

7.2 The Integration Algorithm

Let d be the degree of each element, and let p and q represent the parents of elements. $\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}$ and $\mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}}$ (where, $0 \leq i$, $j \leq d$) represent elements of the local schemas S_1 and S_2 respectively. For example, $\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}$ represents the ith element whose parent is P in S_1, and $\mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}}$ represents the jth element whose parent is q in S_2. Let $L_{\mathrm{EP}} \subseteq L$ 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 $p, q = \varnothing$. In this framework, matching of elements is based on names and structure of schema elements. Let E_{LS} be the sum of all the pairs of edges in (S_1, S_2), then,

$$\mathrm{E}_{\mathrm{LS}} \leq \sum_{i=1}^{d} \sum_{j=1}^{d} (\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}}) \tag{13}$$

The integration algorithm determines if $Equal(\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}})$ where $\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}} \in S_1$ and $\mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}} \in S_2$. The problem at hand is to integrate two regulated schemas S_1 and S_2 into *GIS*. The integration process is of the form:

$$\Upsilon_{GIS}: \psi(\Sigma) \times \Re \times \Sigma^{\sim} \rightarrow GIS \tag{14}$$

where Σ is a set of elements of the pair $(\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}})$ such that $\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}} \in S_1$ and $\mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}} \in S_2$ and $Equal(\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}})$. $\Sigma^{\sim}$ is a set of elements from S_1 and S_2 such that for all elements in S_1 and S_2 the constraint $\neg Equal(\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}})$ holds. The union of Σ and $\Sigma^{\sim}$ represents the set of local schemas S_{LS}. $\psi(\Sigma)$ is the correspondence assertion between pairs of elements from S_1 and S_2, and $\Re$ is a set of correspondence rules for generating the *GIS* from Σ and $\Sigma^{\sim}$. The elements of Σ are merged into *GIS* according to their respective assertions, while elements of $\Sigma^{\sim}$ are merged into *GIS* such that the local relationships between those elements and their parents are maintained. 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. Three operations dominate the algorithm, namely; 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 $(\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}})$ 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 (Υ_i). The *Merge* operation produces the correspondence rules (Υ_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 $(\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}})$ 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 $(\mathrm{L}_1{}^{\mathrm{p}}{}_{\mathrm{i}}, \mathrm{L}_2{}^{\mathrm{q}}{}_{\mathrm{j}})$ 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 $\Theta(\mathrm{E}_{\mathrm{S}})$, and the cost of enqueuing and dequeuing those elements is $O(\mathrm{E}_{\mathrm{S}})$. 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).

8. Output Process

The goal of the output process is to have a transparent global integrated schema, *GIS* that represents all the schemas in the data sources. The nature of the output shows the ability of the integration algorithm to resolve heterogeneities. An integration system that resolves heterogeneities takes away the burden of identifying assertions from the users.

Definition 13. A *Global Integrated Schema* (*GIS*) over a set of local schemas $S_{LS} \mid S_{LS} = \bigcup_{i=1}^{n} S_i$ is a well-formed *Del-G* schema where $\forall L_1{}^p{}_i$: LABEL $\bullet$ ($\exists$ S_{LS}: SCHEMA $\bullet$ $L_1{}^p{}_i \in S_{LS} \wedge Parent(L_1{}^p{}_i) = p) \Rightarrow \forall L^r{}_j$: LABEL $\bullet$ ($\exists GIS$: SCHEMA $\bullet$ $L^r{}_j \in GIS \wedge L_1{}^p{}_i = L^r{}_j$).

An effective integration methodology must guarantee that the *GIS* is a non-redundant, unified representation of all the data sources. In the subsection that follows, we show that the *GIS* correctly represent all the local schemas.

8.1 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 $\bigcup_{i=1}^{n} S_i$. The solution for Q is the computed union of all the partial solutions of q_i.

To enable us prove the correctness of the integration system, we define the following restricting operators; the operator ($\lhd$) restricts the domain, and the operator ($\rhd$) restricts the range of relations. These operators enable us construct sets of objects that satisfy the function Υ_{GIS}. We define the operators as functions.

$$\_\lhd\_ : \mathbb{P}S_{LS} \times (S_{LS} \leftrightarrow GIS) \rightarrow (S_{LS} \leftrightarrow GIS)$$

$$\forall S_i: \mathbb{P}S_{LS}, \Upsilon_{GIS}: (S_{LS} \leftrightarrow GIS) \bullet \qquad (15)$$

$$S_i \lhd \Upsilon_{GIS} = \{(L_i{}^p{}_j, L^r{}_k) \mid L_i{}^p{}_j \in S_i \wedge (L_i{}^p{}_j, L^r{}_k) \in Gr(\Upsilon_{GIS})\}$$[3]

The restriction imposed by the operator ($\lhd$) implies that every element in S_{LS} is mapped to an element in *GIS* such that the domain of Υ_{GIS} is the whole of S_{LS}. $S \lhd R$ filters $(L_i{}^p{}_j, L^r{}_k) \in Gr(\Upsilon_{GIS})$ for which $L_i{}^p{}_j \in S_i$ and produces the result relation.

$$\_\rhd\_ : \mathbb{P}GIS \times (S_{LS} \leftrightarrow GIS) \rightarrow (S_{LS} \leftrightarrow GIS)$$

$$\forall G: \mathbb{P}GIS, \Upsilon_{GIS}: (S_{LS} \leftrightarrow GIS) \bullet \qquad (16)$$

$$\Upsilon_{GIS} \rhd G = \{(L_i{}^p{}_j, L^r{}_k) \mid L^r{}_k \in G \wedge (L_i{}^p{}_j, L^r{}_k) \in Gr(\Upsilon_{GIS})\}$$

The restriction imposed by the operator ($\rhd$) implies that every element in *GIS* is mapped to an element in S_{LS} such that *GIS* is the range of Υ_{GIS}. $\Upsilon_{GIS} \rhd G$ filters $(L_i{}^p{}_j, L^r{}_k) \in Gr(\Upsilon_{GIS})$ for which $L^r{}_k \in G$ and produces the result relation.

Definition 14. A *query* Q on a schema *S* is a regular language over Σ, 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 there is a *Path-Label* $pl \in S$ such that pl **in** l_i and l_i **in** pl (where "**in**" is the sequence inclusion).

Recall that 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;

[3] $Gr(R) \mathrel{\widehat{=}} \{\langle x, y\rangle \mid x \; R \; y\}$ [Sch94]

$$\bigcup_{i,j=1}^{n} q_i(S_j) \sqsubseteq \mathrm{Q}(GIS) \Rightarrow \mathrm{Q}(GIS) \sqsubseteq \bigcup_{i,j=1}^{n} q_i(S_j) \bullet \mathrm{Q}(GIS) \equiv \bigcup_{i,j=1}^{n} q_i(S_j) \tag{17}$$

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

Corollary 1. Suppose $\mathrm{Q}(GIS) \equiv \bigcup_{i,j=1}^{n} q_i(S_j)$, where $\mathrm{Q}(GIS) = (u, l_i, v_1)$ and $\bigcup_{i,j=1}^{n} q_i(S_j) = (u_1, l_k, w)$, then $(u, l_i, v_1) \subseteq (u_1, l_k, w) \wedge (u, l_i, v_1) \supseteq (u_1, l_k, w)$.

Proof. $\mathrm{Q}(GIS) \equiv \bigcup_{i,j=1}^{n} q_i(S_j)$ means that (u, l_i, v_1) satisfies $\mathrm{Q}(GIS)$, which implies that there is a *Path-Label* pl_i in *GIS* such that pl_i **in** l_i and l_i **in** pl_i. Notice that $pl_i \in GIS$ implies that there exist $pl_k \in S_{\mathrm{LS}}$ such that pl_i **in** $pl_k \wedge pl_k$ **in** $pl_i \Rightarrow pl_i \in S_{\mathrm{LS}}$ since every element in S_{LS} is represented at most once in *GIS* (from (15)). Also, $pl_i \in S_{\mathrm{LS}}$ implies that $\bigcup_{i,j=1}^{n} q_i(S_j)$ satisfies (u, l_i, v_1) such that $\bigcup_{i,j=1}^{n} q_i(S_j) = (u, l_i, v_1)$ (by the *closed domain assumption*). Similarly, $\bigcup_{i,j=1}^{n} q_i(S_j) = (u_1, l_k, w)$ means that (u_1, l_k, w) satisfies $\bigcup_{i,j=1}^{n} q_i(S_j)$, which implies that there is a *Path-Label* pl_k in S_{LS} such that pl_k **in** l_k and l_k **in** pl_k. Observe that $pl_k \in S_{\mathrm{LS}}$ implies that there exist $pl_i \in GIS$ such that pl_i **in** $pl_k \wedge pl_k$ **in** $pl_i \Rightarrow pl_k \in GIS$ (since *GIS* is the range of Υ_i from (16)). Every *Path-Label* $pl_i \in GIS$ is unique (by Definition 10). So, if pl_i **in** $pl_k \wedge pl_k$ **in** pl_i and $pl_k \in GIS$ contradicts the uniqueness of pl_i. Therefore, $i = k$ such that $pl_i = pl_k$.

An interesting implication of Corollary 1 is that the *GIS* correctly represents all the elements in the local schemas $\bigcup_{i=1}^{n} S_i$.

9. Conclusions

Web data integration is a complex and detailed process that requires unambiguous explanations to elucidate the underlying concepts. Despite recent advances in information and communication technologies most existing integration systems still involve substantial manual input, and adopt *ad hoc* approaches to data integration without necessarily considering the theoretical foundations and formalisms of such approaches, thus leading to minimal successes. In this paper, we leverage a flexible semistructured data model and a common ontology to provide scalability and correctness of the integration model. Our formalism reduces the complexity of the integration process and guarantees that local schemas are correctly represented in a platform independent *Del-G* format. Our approach scales over multiple data sources since the local schemas are well-formed and the integration methodology only involves simple matching of terms based on semantic names. We also showed that the integration process correctly generates the global integrated schema. This paper represents a shift in the traditional approach to data integration. Plans to introduce more algebraic constructs to enhance the specification of the entire integration process are underway.

Acknowledgements
The authors would like to thank Dr. Peter C. J. Graham and the anonymous reviewers. Their valuable comments and suggestions greatly improved the quality of this paper.

REFERENCES

Adiele, C., *Integration of Web Data Sources for E-Commerce Transactions*, Ph.D. Thesis, Department of Computer Science, The University of Manitoba, November 2004.

Adiele, C. and S. A. Ehikioya, "Dynamic Identification of Correspondence Assertions for Electronic Commerce Data Integration", *Proceedings of the IEEE International Conference on Information Technology (ITCC 2004)*, pp. 223-227, Las Vegas, NV, April 2004.

Alagar, V. S. and K. Periyasamy, *Specification of Software Systems*, Springer-Verlag Inc., New York, 1998.

Bryant, A. and B. Colledge, "Trust in Electronic Commerce Business Relationships", *Journal of Electronic Commerce Research*, Vol. 3, No. 2, pp. 32-39, 2002.

Bukhres, O., J. Chen, A. Elmagarmid, X. Liu, and J. Mullen, "InterBase: A Multidatabase Prototype System", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 534-539, Washington, D.C., 1993.

Beneventano, D., F. Guerra, S. Magnani, and M. Vincini, A Web Service Based Framework for the Semantic Mapping Amongst Product Classification Schemas, *Journal of Electronic Commerce Research*, Vol. 5, No. 2, pp. 114-127, 2004.

Buneman, P., S. Davidson, M. Fernandez, and D. Suciu, "Adding Structure to Unstructured Data", *Proceedings of the International Conference on Database Theory*, Delphi, Greece, 1997.

Batini, C., M. Lenzerini, and S. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys*, Vol. 18, No. 4, pp. 323-364, December 1986.

Chau, P. Y. K., Inhibitors to EDI Adoption in Small Businesses: An Empirical Investigation, *Journal of Electronic Commerce Research*, Vol. 2, No. 2, pp. 78-88, 2001.

Chawathe, S., H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman and J. Widom, "The TSIMMIS Project: Integration of Heterogeneous Information Sources", *Proceedings of IPSJ Conference*, pp. 7-18, Japan, 1994.

Evincible Software, "E-Sign Transactions Platform: The Enterprise Architecture to Implement Electronic Transactions Requiring Electronic Signatures", *White Paper*, October 22, 2003, Available at: http://www.evincible.com/resources/eSign%20Transaction%20Platform.pdf

Fensel, D., *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag, 2001.

Haas, L. M., D. Kossman, E. L. Wimmers and J. Yang, "Optimizing Queries Across Diverse Data Sources", *23rd Conference on Very Large Database Systems*, Athen, Greece, 1997.

Haas, L. M., R. J. Miller, B. Niswonger, M. T. Roth, P. M. Schwarz and E. L. Wimmers, "Transforming Heterogeneous Data with Database Middleware: Beyond Integration", *IEEE Data Engineering Bulletin*, pp. 31-36, 1991.

Larson, J. A., S. B. Navathe, and R. Elmasri, "A Theory of Attribute Databases with Application to Schema Integration", *IEEE Transaction on Software Engineering*, Vol. 15, No. 4, pp. 449-463, 1989.

Lee, M., S. Bressan, G. H. Goh and R. Ramakrishnan, "Integration of Information from Disparate Sources: A Short Survey", *Workshop on Logic of Programming and Distributed Knowledge Management*, pp. 155-158, UK, April 1999.

Maamar, Z., "Commerce, E-Commerce, and M-Commerce: What Comes Next", *Communications of the ACM*, Vol. 46, No. 12, pp. 251-257, December 2003.

The Metadata Coalition, "Metadata Interchange Specification" *Technical Report* Version 1.1, August 1997. Available at: http://www.mdcinfo.com/MDIS/MDIS11.html

Microsoft Corporation, "BizTalk Framework 2.0 – Independent Document Specification", *Technical Report*, December 2000. Available at: http://www.microsoft.com/biztalk/default.htm

Papakonstantinou, Y., H. Garcia-Molina, and J. Widom, "Object Exchange Across Heterogeneous Information Sources", in Yu, P. S., and Chen, A. L. P., Editors, 1995, *11th International Conference on Data Engineering*, March 1995, pp. 251-260, Taipei, Taiwan.

Scheurer, T., *Foundations of Computing*: *System Development with Set Theory and Logic*, Addison-Wesley Publishers, Reading, MA, 1994.

Sipser, M., *Introduction to the Theory of Computation*, PWS Publishing Company, Boston, MA., 1997.

Sheth, A. and G. Karabatis, "Multidatabase Interdependencies in Industry", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 483-486, Washington, D.C., 1993.

Sheth, A. P. and J. A. Larson, J. A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Survey*, Vol. 22, No. 3, pp. 184-236, September 1990.

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*, pp. 54-59, Tucson, May 1997.

Swatman, P. M. C. and P. A. Swatman, "Electronic Data Interchange: Organisational Opportunity, Not Technical Problem", *Proceedings of the 2nd Australian Database-Information Systems Conference* "*DBIS'91*", pp. 290-307, Sydney, Australia, February 1991.

Tygar, J. D., "Atomicity in Electronic Commerce", *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing* (*PODC '96*), pp. 8-26, New York, May 1996.

Uniform Code Council Inc., "SIL – Standard Interchange Language", *Technical Report*, January 1999. Available at: http://www.uc-council.org/e-commerce/ec-sil-general-overview.html

VeriSign® Fraud Protection Services, "Securing the Enterprise Payments Network", *White Paper*, (Accessed on December 22, 2003), Available at: http://www.verisign.com/resources/wp/fraud/PreventingFraud.pdf

W3C, "E-Business XML (ebXML): Enabling a Global Electronic Marketplace", 2002. Available at: http://www.ebxml.org

APPENDIX A

Appendix provides a glossary of the notations used in this paper. This glossary is based on simple set notations and predicate logic as described in [Alagar and Periyasamy, 1998; Scheurer, 1994].

Symbols	Meaning
$\mathbb{P}$	Power set
$\mathbb{P}_1$	Non-empty power set
$\mathbb{N}_1$	A set of positive integers
$\mathbb{N}$	A set of integers
$\wedge$	Logical "and"
$\vee$	Logical "or"
$\neg$	Logical "not"
$\Rightarrow$	Implies that (if then)
$\Leftrightarrow$	If and only if
$\mid$	Satisfying
$\bullet$	Such that
$\widehat{=}$	By definition, equal to
$\in$	Set membership
$\rightarrow$	Member to member association in a function
$\forall$	Universal quantifier
$\exists$	Existential quantifier
$\lhd$	The operator restricts the domain of a function
$\rhd$	The operator restricts the range of a function