ABSTRACT

We describe a family of heuristics-based clustering strategies to support the merging of XML data from multiple sources. As part of this research, we have developed a comprehensive classification for schematic and semantic conflicts that can occur when reconciling related XML data from multiple sources. Given the fact that element clustering is compute-intensive, especially when comparing large numbers of data elements that exhibit great representational diversity, performance is a critical, yet so far neglected aspect of the merging process. We have developed five heuristics for clustering data in the multi-dimensional metric space. Equivalence of data elements within the individual clusters is determined using several distance functions that calculate the semantic distances among the elements.

The research described in this article is conducted within the context of the Integration Wizard (IWIZ) project at the University of Florida. IWIZ enables users to access and retrieve information from multiple XML-based sources through a consistent, integrated view. The results of our qualitative analysis of the clustering heuristics have validated the feasibility of our approach as well as its superior performance when compared to other similarity search techniques.

Keywords: Element matching, information integration, object clustering, reconciliation, XML

1 Introduction

Information integration continues to be an important problem especially when it involves information from a wide range of sources. Interesting sources can range from simple data files containing Web pages, e-mail, etc., to conventional database systems managing inventory and customer data, for example. Integrating all of these data sources and providing a single interface and representation for users is a challenging problem since the sources are autonomous (i.e., independently managed), distributed (i.e., located in different places), and heterogeneous in nature (i.e., using different software, data model, schema, etc. to manage the data). Overcoming these challenges is the task of integration systems.

Figure 1 motivates the need for an integration system. Imagine an online shopping system where customers can browse data on publications (e.g., books and articles) and related products (e.g., computer
The data is stored in heterogeneous sources such as digital libraries, online catalogs, and other e-commerce sites as shown on the bottom of Figure 1.

**Figure 1:** Conceptual view of an integration system accessing data from multiple, heterogeneous information sources. The integration system provides data translation (via source wrappers), data fusion and query support (via the mediator), and caching of frequently used data (via the data warehouse).

Typically, each source uses different tools, data models and schemas to create and manage their data. A consequence of the use of different data models and systems to represent information is that sources provide different query capabilities. In addition to the differences in the data models, the integration systems must deal with the fact that a real-world entity can be represented differently across data sources.

For instance, in Figure 1, the electronic catalog for an online source may represent author names as single character strings, whereas a digital library may represent them as three distinct strings: first-name, middle-name, and last-name. In another example, the price of a book as listed in a publisher’s database may include shipping costs, whereas the e-commerce site for a retailer may separate base price and tax. In yet another example, book information as provided by the publisher may provide two authors, while the same book in one of the online sources may list only one author. The first example illustrates a difference in the structure of the data; the second example shows a difference between the two domains, while the last example describes a difference in the content of the two data items in questions. Differences in the structure and the domain of the data can be detected if we have knowledge about the schema. Differences in the content of the data can be detected if we have knowledge about the associated constraints. Detecting such
differences or discrepancies is important when we want to reconcile related data from multiple sources, as
is the focus of this article.

Besides collecting and reconciling related data from multiple, heterogeneous sources, other goals of
integration systems are to provide a single interface and efficient data access for users. In our sample online
shopping system, for example, users want to issue queries without having to know which sources are
available, where they are located, how they represent their data, and how they are queried. Thus, users
prefer the illusion that all the data is being retrieved from a single large repository. In Figure 1, this
integrated repository corresponds to the data warehouse storing frequently asked queries and their results.

The problems of information integration have been studied in federated database systems [58] and
mediation systems [13, 26, 31, 42] for integrating relational and object-oriented sources. However,
integrating sources containing semistructured data [60], which exhibit varying format and structure and
occur frequently during e-commerce transactions (e.g., contracts) as well as in electronic catalogs, has not
been well studied. Semistructured data or self-describing data collectively refer to information whose
contents and structure are flexible and thus cannot be described and managed by the more rigid traditional
data models (e.g., relational model). Semistructured data can be represented using the Object Exchange
Model (OEM) [49] or the Extensible Markup Language (XML) [65], for example.

The research described in this article is couched within the framework of the Information Integration
Wizard (IWIZ) project, which has been developed in the Database Research and Development Center at the
University of Florida [30, 32]. IWIZ allows end-users to access and retrieve information from various
sources without knowledge about the location, API, and data representation. At present, IWIZ information
sources are assumed to contain semistructured data in XML format. In the future, IWIZ will provide access
to information in other formats such as text files, relational databases, and image archives. The three major
IWIZ components are (1) wrappers [61] to restructure the schema of the results that are being returned
from the sources; (2) a mediator [50, 57] to reconcile and cleanse the results from the wrappers; and (3) the
warehouse [34, 52] to process queries whose results have been previously fetched from the sources.

In this article, we describe our solution to data reconciliation in the mediator. We assume the data to
be reconciled are represented as XML elements. Our novel solution is implemented in the form of a data
merge engine which is capable of producing an integrated result efficiently and accurately by removing
duplicates and resolving inconsistencies among related data items that have been retrieved from the underlying sources. We introduce a multi-strategy clustering model to support the automatic detection of similar elements that represent the same real-world entity. Our model and underlying technologies are general enough to be applicable in a wide range of integration scenarios.

In order for any reconciliation algorithm to be practical, performance is an important factor since mediation-based query processing is a runtime activity. Specifically, the potentially large number of data items and the representational diversity of related items that can occur in different sources [37] have resulted in a number of requirements that reconciliation algorithms must satisfy [55]. The three most important ones are:

1. **Qualitative Comparability**: The result produced by the reconciliation algorithm should be comparable in correctness to the result produced by a detection process carried out manually.

2. **Computational Complexity**: The time and space complexity of the reconciliation algorithm should be polynomial.

3. **Scalability**: The reconciliation algorithm should be scalable with respect to the number of data items to be compared, the number of features per item, and the number of data sources which participate in the integration system.

We will revisit these requirements at the end of the article to demonstrate that our approach represents a viable solution to data reconciliation in integration systems.

The rest of this article is organized as follows. Section 2 provides a brief survey of the research most closely related to our work. Section 3 provides the definition of important terms that will be used in this article. Section 4 outlines our classification scheme for XML data and formulates the element-matching problem. Section 5 illustrates our framework for reconciling and cleansing XML data from multiple heterogeneous sources. Section 6 describes our multi-strategy clustering model including the underlying distance functions and heuristics, which have been implemented and tested in our IWIZ mediator framework. Section 7 describes the results of the qualitative analysis of our clustering heuristics. Conclusions and future research opportunities are described in Section 8.
2 Related Research

The research related to this work falls into the following four categories: (1) information integration and sharing systems; (2) semantic heterogeneities and conflict resolution; (3) data matching and similarity measure; and (4) data clustering methods. We summarize the state-of-the-art in these categories in the following sections.

2.1 Information Integration and Sharing Systems

Information integration and sharing refer to the process of retrieving related information from multiple information contexts, combining it using a predefined global schema or view, and using the combined information in a context that is different from the original one. There are two generally accepted approaches to integrating and sharing information from heterogeneous sources: The data warehousing approach (e.g., WHIPS [40], STRUDEL [25], MOMIS [9], OASIS [54] and Xyleme [1, 67]), which uses a logically centralized, persistent storage to manage frequently accessed data for faster retrieval, and the mediation approach (e.g., TSIMMIS [12], Infomaster [20], WHIRL [15], MIX [6], Tukwila [29], and ObjectGlobe [11]), which supports on-demand querying of the underlying sources.

Many components and capabilities in IWIZ resemble those found in other existing mediation-based integration systems: they are XML-based, use a single, predefined global view that clients can browse and query through a graphical user interface, and rely on wrappers and mediators for translating, restructuring, and reconciling source information. On the other hand, existing mediation projects have so far focused mostly on efficient query processing inside the mediator and neglected largely the processing of data that are being returned from the sources. In the IWIZ project, we have focused our attention on automatic resolution of conflicts that can occur between related data in different contexts in order to produce a single integrated result with as little human intervention as possible. Given the rapid increase of information available online, automatic conflict resolution in integration systems will reduce their dependency on domain experts and in turn increase the scalability and flexibility of applications using these systems.

2.2 Semantic Heterogeneities and Conflict Resolution

Before one can share information, one must resolve conflicts that can occur between related data in different contexts. The conflicts are due to heterogeneities, which exist at various levels of abstraction.
Primarily, one is faced with heterogeneities at the system level (e.g., differences in the underlying hardware, network protocol, operating system), at the data management level (e.g., differences in the data model, access commands), and at the semantic level (e.g., differences in the way related or similar data are represented).

There are three types of conflicts: **Structural conflicts** refer to discrepancies in the structural representations of related data in different sources. **Domain conflicts** refer to discrepancies between the domains of similar or related data elements in different sources. Structural and domain conflicts both arise during the conversion of data from the source context into the target context and are resolved by source wrappers [62]. **Data conflicts** refer to problems related to identifying data elements, which represent related or the same real-world entity. Data conflicts arise when attempting to merge the translated and restructured data elements from the sources into a unified result and are resolved in the mediator. Given the widespread use of XML to represent semistructured data, we have devised a new classification scheme [51] for structural, domain and data conflicts among XML-based data sources, which forms the basis for the work described in this article.

To date, most of the research in this area has centered on identifying structural and domain conflicts in structured data sources. Batini et al. [7] have analyzed conflicts based on the E/R model and proposed a framework for the problem of schema integration. Dayal and Hwang [16] proposed a generalization technique to resolve discrepancies and inconsistencies found in multiple heterogeneous databases, including the use of an extended functional data model to model the information in the individual database. Kent [38] applied an object-oriented database programming language to resolve domain and structure mismatches. Lakshmanan [41] and Miller [45] introduced an SQL-based language to resolve schematic discrepancies. Siegel and Madnick [59] and Sciore et al. [56] resolved semantic conflicts based on the context information (e.g., units and scales) describing the characteristics of data. Fankhauser et al. [23, 33] introduced IRO-DB framework to resolve schematic discrepancies between relational and object-oriented database systems. Li and Clifton [43] resolved a schematic discrepancy found in heterogeneous databases by using neural networks which use schema information (i.e., a database description) to match corresponding attributes.
As far as resolving data conflicts is concerned, Kashyap and Sheth [35] proposed a formal model for measuring similarity between objects in databases. Dey et al. [17-19] proposed probabilistic and distance-based decision models to detect matching data elements in two databases. The worst-case complexity of their matching procedure, which is based on the Hungarian method [48], is $O(N^3)$ where $N$ is the cardinality of the larger of the two databases. Even though the algorithm runs in polynomial time, it is not practical for matching large data sets. Moreover, based on the experiments conducted by the authors, up to 35% of the data elements are matched incorrectly. Like Dey et al., we aim to develop methodologies for efficient data merging. However, the average running time of our method is faster by the order of the total number of data elements to be compared. In addition, using our clustering method, we found that in certain cases the number of matching errors is less than 5% of the total number of comparisons, depending on cluster size and expected number of duplicates in the underlying data sets.

2.3 Data Matching and Similarity Measures

Data matching refers to the process of comparing and identifying related data items. Data matching is an important part of many tasks such as clustering, similarity search, and information integration. In the clustering process, data items need to be compared before related data can be clustered. In similarity search, a given data item needs to be compared to other data items in a set before the items that are similar to a given data item can be found. In information integration, items from multiple sources need to be compared and related data items need to be identified before combining them.

Matching of data requires similarity measures such as correlation coefficients, distance coefficients, association coefficients, and probability similarity coefficients [2] for comparing data items in the multi-dimensional space. In computer science, the distance coefficients, also known as distance functions, have had widespread use and most of them are metrics. A similarity measure is a metric if it satisfies the following properties:

Let $x$, $y$ and $z$ be data items to be measured and $d(x, y)$ be a similarity measure or the distance from data item $x$ to data item $y$. Then $d$ must satisfy the following properties:

1. Symmetry: \[ d(x, y) = d(y, x) \]
2. Non-Negativity: \[ 0 \leq d(x, y) < \infty, \text{ where } x \neq y \text{ and } d(x, x) = 0 \]
3. Triangular Inequality: \[ d(x, y) \leq d(x, z) + d(z, y) \] (2.1)
The distance functions most commonly used for similarity searches are the Euclidean, Cosine, Manhattan, and Minkowski distances. These distance functions are used to compare data items in a vector space, which requires a transformation of each data item into a vector. In other words, a data item with $k$ features (or attributes) must be mapped into a $k$-dimensional feature vector before a special distance function computes the distances between two feature vectors. An alternative to the matching of data items is to compare them in a metric space in which the feature transformation is not needed. In this case, only a distance function that satisfies the three metric properties is needed.

Furthermore, a data item can contain one or more components or sub-items. We refer to such a data item as a complex data item. Comparing complex data requires a distance function that is defined for all features and sub-items of the data item under consideration. Dey et al [17] suggest several types of distance functions for different types of feature values. Zobel and Moffat [68] provide a set of standard similarity measures that include combining and weighting functions.

Matching related data can be viewed as a part of the well-known similarity search problem: *Given an object, find other objects that are similar to the given object.* Spatial access methods (SAMs) have been introduced to support similarity searches by indexing the spatial objects such as multi-dimensional points, lines, rectangles and other geometric objects [21]. SAMs provide index structures, which enable fast responses to range and nearest neighbor queries. In our work, we use SAMs as auxiliary data structures to cluster related heterogeneous data during reconciliation.

Several spatial access methods have been proposed, for example, the R-tree [28] and its variants [22], the TV-tree [44], the X-tree [8], the SR-tree [36], the SS-tree [66], the Metric-tree [63], the MVP-tree [10], and the M-tree [14], to name a few. The X-tree, the TV-tree, the SS-tree, the SR-tree as well as the R-tree and its variants are suitable for indexing spatial objects in the vector space. The Metric-tree, the MVP-tree, and the M-tree are suitable for indexing spatial objects in the metric space.

One of the access methods considered in our research is the M-tree. Like B-trees and R-trees, M-trees grow in a bottom-up fashion and are balanced. However, the M-tree differs from the B-tree in the fact that a node in an M-tree is split when it is 100% full whereas a node in a B-tree is split when its capacity reaches a predefined threshold (usually 70% of the total node capacity). The M-tree differs from the R-tree
in the fact that the M-tree uses a sphere-shaped routing object to identify objects that are close to one another, whereas the R-tree uses a minimum-bounding rectangle.

In the M-tree, objects are organized into M-tree nodes (or clusters) based on a decomposition strategy. A common strategy used in metric-trees\(^2\) [14] is the *Generalized Hyperplane Decomposition* [63]: Given a set of objects, arbitrarily pick two different objects, for example, \(x\) and \(y\), from the set of objects. The set is partitioned into a subset which contains every object \(z\) that is closer to \(x\) than to \(y\), a second subset which contains every object \(z\) that is closer to \(y\) than to \(x\), and a third subset which contains the remainder of the objects in the original set.

In our research, we use the M-tree data structure in one of the clustering heuristics since it is simple, easy to implement, and allows matching of elements in the metric space. As described above, the metric space is more appropriate for comparing elements with variable structure (such as XML elements) since unlike in the vector space, they do not have to be transformed into \(k\)-dimensional vectors first (where \(k\) is fixed and represents the number of attributes of each element).

### 2.4 Data Clustering Methods

Clustering of data refers to the grouping of data items into clusters such that data items in one cluster are more similar to one another (high affinity) and those in separate clusters are less similar to one another (low affinity). Data clustering has been widely used in many areas such as statistical data analysis [4], information retrieval [64], and data mining [24]. Three excellent reviews of clustering methods can be found in [2, 46, 53].

Different clustering methods have been developed for different applications. For example, in information integration, Samatova et al. [55] present a hierarchical clustering method to build a global hierarchical cluster by merging clustering hierarchies, which are generated locally at each of data sources. Each data source contains different data items with the same set of features. The data items are assumed to be homogeneous for the type of features and for the units of measurements of those features. Like Samatova et al., we aim to develop a framework for efficient data merging. However, we allow data items

---

1 Here, an “object” refers to a thing that constitutes a particular subject matter, much like we have used the term “data item” so far.
2 “Metric-trees” are trees that compare objects in the Metric space.
from different sources to be heterogeneous in terms of their feature types as well as in the number of features for each type.

**Figure 2:** Conceptual integration of elements that represent the same book entity.

### 3 Terminology

For the remainder of this article, we focus on the reconciliation of related data in an integration system. Before defining the important problems for data reconciliation, we first define the important terms that will be used for the rest of this article. Unless stated otherwise, we will use “**entity**” when referring to an object in the real-world, “**item**” (or data item) when referring to data that represents a particular entity, “**element**” (or data element) when referring to a fragment of a particular item. Each item (element) has features that describe properties of the entity. Figure 2 depicts a sample book entity, a book item, and five book elements. The book item and book elements are represented in XML format. In an integration system, the data items are gathered from various sources that contain elements which can have (1) the same set of features but for a different item (horizontal fragmentation), (2) different sets of features on the same item (vertical fragmentation), or (3) a combination of the two (block fragmentation). In addition, those sources
can provide the same element (i.e., elements can be fully or partially replicated either in the same source or across different sources).

We define base element to be an element that is originally gathered from a source. We also define compound element to be an element that is derived from two or more elements by joining them on some common feature values. Two elements are related if they have a common ‘subset’ of features; otherwise, they are unrelated. Note that elements that are related can be either base or compound elements. In information integration, only related elements are considered because unrelated elements represent different types of entities. We say elements that are related are equivalent if they are fragments of the ‘same’ item (i.e., they represent the same entity); otherwise they are nonequivalent. Elements that are equivalent are duplicates of each other if they have the same collection of features and the same value for each feature. In Figure 2, all elements are related since they have a common subset of features (e.g., Title). The elements in the first four sources are equivalent because they are fragments of the same book item. The element in the fourth source is a duplicate of the element in the third source whereas the element in the fifth source is nonequivalent to the elements in the first four sources.

In the semistructured data model, an entity is represented as an element which can have a set of attributes describing its properties and a list of (sub)elements describing parts of information. In XML, the element that has no attribute and does not contain any other elements but contains a single value that represents information is called simple element; otherwise it is called complex element.

4 The Element Matching Problem
We are now ready to define the element-matching problem which is central to data reconciliation. The element-matching problem is defined as the problem of identifying equivalent elements which originate from (or have been created in) different contexts in different sources. More formally:

Given $N$ base elements that are gathered from various sources, find a set of clusters $C = \{C_1, C_2, \ldots, C_n\}$ where $n \leq N$, such that $C_i$, $1 \leq i \leq n$, is a cluster that contains equivalent elements.

In order to reconcile related data in an integration system using clustering, the ideal set of clusters should fulfill the following two requirements:
1. The clustering process including the analysis of the base elements and the clustering itself should be fast, proportional to $O(N)$ or $O(N^2)$ but not higher than $O(N^2)$, because the time and space cost will be prohibitive for large datasets.

2. The number of mismatched elements for each cluster should be comparable to the error rate obtained when matching elements manually.

Element matching is made difficult by the existence of data conflicts that occur between related elements and that prevent a straightforward comparison of elements based on their feature values. Recall that structural and domain conflicts both arise during the conversion of data from the source context into the target context and are resolved by source wrappers [62].

Given the difficulty of the problem and the fact that we are using heuristics rather than exact procedures to determine matches, we also need to measure the matching error (clustering error) and find ways to reduce it if necessary. We start by illustrating some of the data conflicts that may occur between related elements in different domains and outline our approach to measuring matching errors. Finally, we introduce a data integration framework to support automatic reconciliation of heterogeneous data and describe how we can detect similar elements using our multi-strategy clustering model.

![Figure 3: Two conceptual representations of the same book in different sources.](image-url)
### 4.1 Data Conflicts

Data conflicts arise when attempting to merge the translated and restructured elements from the sources into a unified result. To illustrate some of the potential conflicts, consider the two book elements in two different sources as shown in Figure 3. Although both books represent the same real-world entity, they obviously exhibit several discrepancies. For example, in source 1 the ISBN number is unknown, and the values for the publisher subelements differ, etc. Note, although both book elements have the same underlying structure\(^3\), namely author, title, ISBN and publisher, they may differ in their constraints. For example, the schema in source 1 may specify that the structure of a book element must contain at least one author, whereas book elements represented in source 2 may have exactly one author. Matching the elements that represent the same real-world entity but differ in their contents is a challenging problem.

To solve the element-matching problem, the following types of conflicts need to be resolved:

- **Synonym conflict:** It arises when two different values have the same meaning or refer to the same real-world entity. For example, in Figure 3, the names of the second author, “Abraham Silberschatz” and “A. Silberschatz,” represent a synonym conflict since they refer to the same person but are represented by different strings.

- **Spelling conflict:** It arises when the same data value is spelled differently in different contexts. For example, in source 2 in Figure 3, the name of the first author of the book, “Henry,” is misspelled as “Herry”. The spelling conflict can be considered a special case of the synonym conflict since the misspelled term and its correct form are semantically equivalent.

- **Acronym conflict:** It arises when a data value is an acronym for another. For example, in Figure 3, the publisher “MGH” is an acronym for “McGraw-Hill.” The acronym conflict can be considered a special case of the synonym conflict since the acronym and its full name are semantically equivalent.

---

\(^3\) This is the result of the schema restructuring performed by wrappers, which transform elements conforming to a source schema into elements conforming to the global schema.
• *Homonym conflict:* It arises when values of the same data type refer to different things. For example, in Figure 3, the second author in source 1 and the first author in source 2 have the same local identifier (number=”4”), but they refer to two different real-world individuals.

• *Missing value conflict:* It arises when a data value is missing or when two elements representing the same real-world entity have different number of subelements. The data value that is missing can be thought of as having a NULL value. For example, in Figure 3, there is a missing value conflict between the ISBN numbers, since the ISBN for the book in source 1 is missing.

We resolve data conflicts using the IWIZ integration framework, which will be described in Section 5, and a set of distance functions, which are a part of our multi-strategy clustering model that will be described in Section 6. Due to space limitations, we only show selected data conflicts, for example, between sub-elements such as author, ISBN, and publisher. The conflicts that can occur between attributes or between attributes and elements are not shown here. Those conflicts can be resolved by using the same approach as is used for resolving conflicts between sub-elements. Other conflicts such as generalization/specialization, attribute/element naming conflicts, etc., which have not been mentioned here, can be resolved using restructuring and conversion rules [61]. An exhaustive classification of structural and domain conflicts among related XML data sources is provided in our technical report [51].

### 4.2 Matching Performance

Before we describe our clustering model and underlying heuristics in detail, we define the two types of errors that can occur when attempting to cluster data.

1. *False alarm,* which occurs when our heuristic indicates that two related elements are equivalent, although they are actually nonequivalent in the real world;

2. *False dismissal,* which occurs when the clustering heuristic indicates that two related elements are nonequivalent, although they represent the same item or entity in the real world.

The number of false alarms is derived from the total number of comparisons between elements representing different real-world entities but which have been arranged into the same cluster. The number of false dismissals is derived from the total number of comparisons between elements representing the same real-
world entities but which have been arranged into different clusters. The total error is obtained by adding the number of false alarms and false dismissals.

Figure 4: Three clustering scenarios and the resulting errors: (A) no clustering, (B) the ideal clustering, (C) a clustering that decreases false dismissals but increases false alarms.

Figure 4 illustrates three different possibilities for how two different elements each with its own copy, can be arranged into different clusters. In the figure, circles represent data elements, rectangles represent clusters and lines represent comparisons (solid lines represent false dismissals, dotted lines represent false alarms). In scenario A, when no clustering is used and no duplicates are removed, we end up with four clusters each of which contains one data element. In this case, there are no false alarms but two false dismissals due to the grouping of each duplicate into its own cluster.

One way to remove false dismissals is to combine the data elements of the same type into the same cluster by applying a clustering heuristic. The ideal clustering is shown in scenario B, where our clustering heuristic returns a single copy of each element (both errors are zero). In scenario C, when a heuristic is applied to reduce false dismissals (i.e., by clustering some of the elements together), the number of false alarms increases. This is due to incorrect matches between an element of type 2 and the two elements of type 1. Although the clustering heuristic was able to reduce the number of false dismissals by one, it increased the number of false alarms to two.

The problem of reducing both false dismissals and false alarms at the same time in order to reduce the total error is a challenging problem. In most applications, including integration systems, false alarms are considered more serious errors than false dismissals since they result in the erroneous removal of data
from the result. Hence, our goal is to design a clustering heuristic that provides the greatest possible reduction of false dismissals while at the same time limiting the number of false alarms as much as possible.

5 Data Reconciliation and Integration Framework

In this section, we describe the IWIZ integration framework, which enables semiautomatic reconciliation of heterogeneous query results from multiple sources. A conceptual overview of the IWIZ architecture is shown in Figure 5. Storage components include the sources, the data warehouse, and the metadata repository. Control components include the querying and browsing interface, the warehouse manager, the mediator, and the wrappers. In addition, there is information not explicitly shown in the figure, which includes the global schema, the queries and the data. The global schema, which describes the structure of all internal data, is represented as a Document Type Definition (DTD); future versions of IWIZ will use an XML Schema Description (XSD). The definitions of concepts and terms used in the schema are stored in the global ontology. The global schema and the ontology are stored in the metadata repository. Internally, all data are represented using XML’s Document Object Model (DOM). Data are manipulated through queries expressed in XML-QL, which was one of the many query languages for XML when we started this project.

![Figure 5: Schematic description of the IWIZ architecture and its main components.](image)
5.1 Front-end: User Views
Users interact with IWIZ through QBI, which provides a conceptual overview of the source contents in the form of the global IWIZ schema and shields users from the intricacies of XML and XML-QL. QBI translates user requests into equivalent XML-QL queries, which are submitted to the warehouse manager. In case when the query can be answered by the warehouse, the answer is returned to QBI. Otherwise, the query is processed by the mediator, which retrieves the requested information from the relevant sources through the wrappers. The contents of the warehouse are updated whenever a query cannot be satisfied or whenever existing content has become stale. Our update policy assures that over time the warehouse contains as much of the requested result set as possible to answer the majority of the frequently asked queries.

5.2 Back-end: Schema Restructuring
The requested information is gathered from the underlying sources through the source-specific wrappers. Each wrapper maps the data model used in the associated source into the data model used by IWIZ. In addition, it has to determine the correspondence between concepts used to describe the source schema and those used to describe the global IWIZ schema and carry out the structural conversions. These structural conversions are captured in form of mappings which are generated when the wrapper is configured. To generate the mappings, the wrapper uses the explicit source schema defined in the form of DTD (or XSD) as well as a local ontology. This local ontology describes the meaning of the source vocabulary in terms of concepts of the global ontology. If the underlying sources have no explicitly defined DTD (or XSD), one must first be inferred by the wrapper [3, 27].

At run-time, the wrapper receives XML-QL queries from the mediator and transforms them into equivalent queries that can be executed on the XML source documents using the wrapper’s own query processor; note, we are assuming that our sources have no query capabilities of their own. The result of the query is converted into an XML document consistent with the global IWIZ schema and returned to the mediator. In IWIZ, the wrapper reorganizes the structure of data and maps the terms used to describe the source schema to the terms used to describe the global IWIZ schema. It does not reconcile data content in the query result. The reconciliation of data content is the task of the mediator. The IWIZ wrapper automates
much of the setup and conversion specification generation with minimal human intervention. Details describing the IWIZ wrapper design and implementation can be found in [61].

5.3 Middleware: Data Reconciliation

Data reconciliation takes place inside the IWIZ mediator which consists of two major components: the Query Rewrite Engine (QRE) [57] and the Data Merge Engine (DME) [50]. The tasks of the QRE are to (1) re-write an input query against the global IWIZ schema into a set of subqueries suitable for execution by the source wrappers, (2) generate a query plan that describes the execution sequence for combining partial results from individual sources, and (3) invoke the source wrappers. The tasks of the DME are to (1) collect the results that are being returned from the sources, and (2) execute QRE’s query plan on the results and produce an integrated answer to the user query with all inconsistencies resolved. DME uses our multi-strategy clustering model (described in detail in Sec. 6) to reconcile conflicts and inconsistencies during element matching. We briefly describe how the reconciliation process works in DME.

To reconcile data, the DME needs a merge specification, which is generated at built-time and validated by domain experts using the results of one or more probe queries as the “training set.” The specification contains (1) one or more distance functions to compare the elements in the data set, (2) for each component of a particular element type, a weighting factor that indicates its importance when computing the distance between elements, and (3) a threshold indicating the largest distance between two elements for which the elements are still considered equal and should be clustered together. The merge specification is a part of the mediator metadata.

![Figure 6: Run-time Control Flow among Components of the DME (inside the mediator).](image-url)
Figure 6 shows the run-time control flow among the DME modules. Dark shaded entities represent software modules; lighter shaded entities represent input/output: the entities shown as flowchart documents represent XML documents containing the elements that are returned from the sources. The rectangular and circular entities represent internal data structures (e.g., query plan and merge specification). The block arrows represent the entry and exit points for DME.

DME works as follows: The individual query results from the wrappers are forwarded to the Data Cleaner for pre-processing. The task of the data cleaner is to pre-process the individual query results before they are joined and merged. It performs several important functions:

- It converts data values of type string into all upper or all lower case.
- It removes extra white spaces that may appear in string values.
- It spell-checks string values against an internal and optional (external) dictionary.
- It removes the suffixes/prefixes of words in order to transform them into their root form (e.g., “kindness” is transformed into “kind”).
- It removes common function words (e.g., the, of, as, etc.) to simplify detection of synonyms, acronyms, and spelling conflicts.

The cleansed data is sent to the Data Joiner, which uses the query plan provided by QRE to join the appropriate elements. The joins are performed using the available join attributes (joinable elements) to form a complete answer set (if possible). The goal is to combine the partial information from multiple sources into a complete, consistent, and correct answer. For example, in the bibliography example, the joiner may combine the author and address values from one source with the relevant title and publication information from another. It is important to note that the Data Joiner does not detect and remove duplicate elements. More specifically, the Data Joiner combines fragments of vertically decomposed data.

The Cluster Generator uses one of several clustering techniques to identify equivalent elements and to resolve data conflicts in the joined answer set. The Cluster Generator compares the elements using the distance functions that are defined for different data types in the merge specification (top right corner in Figure 6). Next, it clusters the data items and generates a clustering tree using one of the heuristics presented in Section 6. Each leaf of the tree represents a cluster containing a set of elements that are equivalent under the clustering heuristic and merge specification.
The clustering tree is used by the Data Unifier to merge elements and form to a complete item, if possible. For each cluster in the clustering tree, the Data Unifier merges those elements for which the distance between them is less than the threshold defined in the merge specification. Merging of elements is done by removing all but one of the copies of the same data item. Figure 7 shows the algorithm for merging of elements.

```
/* Variables */
E : a particular type of element which defined in the global schema,
S : a set of M types of sub-element that is defined under E in the global schema,
A : a set of K attributes of E which is defined in the global schema.
C : a particular cluster containing a set of N element instances of type E that are being unified.

Unify (C)
1: If E is simple /* element that contains a single value */
2:   For each element instance Ci, where 1 ≤ i ≤ N, do
3:     hash value of it.
4:   Return the set of hash keys which are unrepeatable.
5: Otherwise, /* if E is complex */
6:   For each element instance Ci, where 1 ≤ i ≤ N, do
7:     count the number of attribute and sub-elements.
8:   Pick the element, say E[x], that has highest number of attributes and sub-elements, as the host element.
9:   For each attribute A[k], where 1 ≤ k ≤ K, do
10:     For each element instance Ci, where 1 ≤ i ≤ N, do
11:       Let AV be the value set containing the values that have the highest frequency
12:       If AV has only one value,
13:          /* The attribute value of the host element is set to be AV. */
14:            E[x].A[k].value = AV[1];
15:       else if E[x].A[k].value is not in AV,
16:          Let AV[y] be the attribute value that appears in the element that has the next highest number
17:           of attributes and sub-elements
18:            E[x].A[k].value = AV[y];
19:   Else /* the element value of the host element is set to be E[x]. */
20:     For each sub-element of type S[j], where 1 ≤ j ≤ M, do
21:       If S[j] is simple and has constraint of exactly-one
22:         For each element instance Ci, where 1 ≤ i ≤ N, do
23:           Count frequency for all possible values.
24:           Let EV be the value set containing the values that have the highest frequency
25:           If EV has only one value,
26:              /* The element value of the host element is set to be EV. */
27:                E[x].S[j].value = EV[1];
28:           else if E[x].S[j].value is not in EV,
29:              Let EV[z] be the value that appears in the element that has the next highest number
30:                of attributes and sub-elements
31:                  E[x].S[j].value = EV[z];
32:           else if S[j] is complex and has constraint exactly-one
33:              For each element instance Ci, where 1 ≤ i ≤ N, do
34:                E[x].S[j].value = Uniﬁy (D);
35:           else /* S[j] is either simple or complex and has other constraints */
36:              For each element instance Ci, where 1 ≤ i ≤ N, do
37:                Calculate the distance between the sub-elements of Ci and the sub-elements of E[x].
38:                  if distance > unifying_threshold,
```

Figure 7: Algorithm for unifying equivalent elements.

The algorithm classifies the types of element which can be either simple or complex. For simple element types, the values of the elements appearing in the cluster are hashed; the set of hash keys which are
unrepeatable is returned. For complex element types, one of the elements in the cluster is chosen as the *host* element, which is used as the starting element for data unification. In Figure 7, we pick the element that contains the highest number of attributes and sub-elements to be the host element. This is based on the widely used assumption that the elements containing more information tend to be more complete. We can also consider other strategies for choosing the host element by taking into account the availability or reliability of the underlying data sources.

Finally, the *Data Filter* module applies any remaining predicates from the original user query on the cleansed, joined, and merged data and returns the final result to the user. For example, some of the data items may contain extra information that was needed during the join and clustering steps but should not be part of the answer. This filtering step is equivalent to projection in relational query languages.

6 Multi-Strategy Clustering Model

As part of the IWIZ mediator, we have developed a multi-strategy clustering model (MCM) to support automatic reconciliation of heterogeneous data from different sources, typically as the result of a query. Specifically, MCM resolves data conflicts by determining the similarity of elements, which is used to remove duplicates (in case of exact matches) or to resolve inconsistencies and missing data values (in case of partial matches).

MCM addresses the case when the data sets to be reconciled are partially overlapping, i.e., the data sets contain elements that are related or equivalent. The other cases where the data sets are completely unrelated or duplicated can be reconciled trivially. MCM includes *distance functions* for measuring the similarity of data elements, *data structures* and *operations* for representing and manipulating clusters (*clustering trees*), and a set of *clustering heuristics* for dividing the input data elements into clusters.

6.1 Distance Functions

In this section, the term “object” is used to refer to both the abstract and instance levels of a particular element. In case we want to differentiate between the two, the terms “object” and “object instance” are used, respectively. Like elements, objects can be simple or complex. A simple object contains only a single value; a complex object contains a set of attributes or one or more nested objects which can be either simple or complex.
Choosing an appropriate distance functions is important since it is used to measure the (dis-)similarity between two objects. Distance functions are sensitive to the application domain and need to be chosen based on the types of comparisons that are made to help solve the data conflicts. For clustering heuristics that operate in the metric space, the distance function must satisfy the properties (2.1) described in Section 2. Distance functions can be classified into two categories: Primitive and complex distance functions.

6.1.1. Primitive Distance Functions

Primitive distance functions measure the (dis-)similarity between (values of) two attributes or two simple objects. The definition of a primitive distance function depends on the types of the input values. For binary data types (e.g., yes/no, female/male), the distance function is defined to be 0 in case of a match and 1 otherwise [4]. For numeric data types (e.g., integer, float), the distance function is defined as the absolute value of the difference of the data values (normalized on a 0-1 scale). For nominal data types (e.g., color), the distance function is defined to be 0 or 1 depending on whether the data values match or not. An alternative distance function for data values of type nominal may be obtained from the user. For example, the user may define the distance between a ‘red’ and a ‘brown’ hat to be 0.1, the distance between a ‘red’ and a ‘blue’ hat to be 0.9, and the distance between a ‘brown’ and a ‘blue’ hat to be 0.5. For string data types, the distance between two strings may be defined as their edit distance, which can be obtained either by counting the number of insertions, deletions, and changes of the characters that separate the two strings, or by finding the longest common subsequence of characters. Alternatively, the distance between two strings can be based on the occurrences of typographical errors or on synonym and acronym relationships that may exist between the strings.

Our multi-strategy clustering model provides four default primitive distance functions for binary, numeric, nominal and string data types. In addition, it allows user-defined functions to support a variety of data types and constraints that can be defined using XML Schema. Since most XML data that we are reconciling are string-based, the most-frequently-used distance function is based on edit distances. Let \( S_i \) and \( S_j \) be two strings to be matched and \(|S_i|\) be the number of characters in \( S_i \). The edit distance we are using is defined as follows:

\[
\text{distance}(S_i, S_j) = \frac{|S_i| + |S_j| - 2*|\text{LCS}(S_i, S_j)|}{\text{Max}(|S_i|, |S_j|)}
\]  

(6.1)
where LCS($S_1$, $S_2$) is the longest common subsequence of characters which are in both $S_1$ and $S_2$, and Max($|S_1|$, $|S_2|$) is the maximum number of characters of $S_1$ and $S_2$.

### 6.1.2. Complex Distance Functions

A complex distance function measures the (dis-)similarity between two complex objects. It can be derived recursively from the distance between the sub-objects. Before defining a distance function between two complex objects, we introduce some important terminology:

- $O^r$ represents a complex object of type $r$ (e.g., book).
- $O^t$ represents a sub-object of $O^r$ whose type is $t$ (e.g., author).
- $T$ represents a set of possible types of sub-object $O^t$ (e.g., \{author, title, year\}).
- $A^k$ represents the $k^{th}$ attribute of $O^r$.
- $m$ is the total number of attributes of $O^r$.
- $W^{rt}$ and $W^{rk}$ represent the weight values associated with $O^t$ and $A^k$ of $O^r$, respectively. As outlined in Section 5, these weight values are adjusted and validated by domain experts at build-time using the results of one or more probe queries.

In general, the distance function, $\Delta$, between two instances of object $O^r$ is defined as:

$$\Delta O^r = \sum_{i \in T} (W^{rt} \Delta O^t) + \sum_{k=1}^{m} (W^{rk} \Delta A^k)$$  \hspace{1cm} (6.2)

If $O^r$ is a complex object, $\Delta O^r$ is defined in a similar way. Otherwise, the object $O^r$ is a primitive object to which a primitive distance function can be applied (e.g., the edit distance function is applied when $O^r$ is of type string). The distance between two attribute values, $\Delta A^k$, is defined as the primitive distance between the corresponding attribute types (e.g., numeric, binary, string).

As outlined in Section 4, each object and attribute represented in XML has a cardinality constraint. Different constraints require different complex distance functions. Due to space limitation, we can only show complex distance functions accommodating two important constraints: optional and multiple sub-objects. Complex distance functions accommodating other constraints can be derived from these constraints and from the general distance function $\Delta$ in Definition 6.2. [50].
The *optional* sub-object constraint refers to the case when an object, \( O' \), may be allowed to have zero-or-one sub-objects of a particular type, \( O_t \). Based on the optimistic assumption that the two object instances that are being compared are related, the distance between them should be mainly derived from the distances between sub-objects that are not missing. Therefore, we define \( \Delta O' \) as follows:

\[
\Delta O' = \begin{cases} 
\theta & \text{If there exists } \Delta O^s \neq \infty \text{, such that } s, t \in T \text{ and } s \neq t \\
\infty & \text{Otherwise}
\end{cases}
\] (6.3)

where \( \theta \) is a positive real number close to zero.

The *multiple* sub-objects constraint refers to the case when an object may be allowed to have multiple (e.g., one-or-more and zero-or-more) sub-objects of a particular type. As shown in Figure 3, each book has two authors. In this case, we need to calculate distances between every author from different sources, and find two appropriate matches of the two authors. The appropriate matches can be obtained by using Hungarian algorithm [39] which is computational expensive for a high cardinality data set. In real-world applications, although the number of authors of a particular book seems to be relatively low, authors may be defined as complex objects, which can have different and multiple sub-objects; hence, the complexity of calculating distances remains an issue. Therefore, to reduce the complexity of the computation, a heuristic can be used by choosing the first \( k \) minimum distances from the set of distances between sub-objects from different sources, where \( k \) is the cardinality of the smaller set of the sub-objects. Such heuristic is used under the optimistic assumption that the two object instances that are being compared are related.

### 6.2 Clustering Trees

The second component of our clustering model is a data structure and operations to represent and manipulate the clusters containing related elements. This data structure is called clustering tree. Our clustering tree has a single root containing a set of cluster nodes. A cluster node can contain other cluster nodes or a set of elements that are related. If the cluster node contains other cluster nodes, it is called a non-leaf or *internal* node. Otherwise, it is a *leaf* node.

The clustering tree is initialized and populated when the clusters have been generated. The shape of the tree depends on the heuristic that is used to generate the clusters. For example, when applying the M-
tree-based heuristic, the resulting clustering tree is a multi-level tree where each cluster node corresponds to a node in an M-tree. The other heuristics described in this article produce single-level trees containing a set of leaf nodes each of which represents a single cluster.

6.3 Clustering Heuristics

We have developed three types of heuristics for building a clustering tree, namely the *All-pair-Comparison-based heuristic* (AC), the *Selected-Comparison-based heuristic* (SC), and the *M-Tree-based heuristic* (MT). In the case of SC, we have developed two additional variations to overcome potential problems caused by its fixed cluster size and non-overlapping clusters. Each type of heuristics has two important input parameters:

- $N$ represents the number of elements to be matched, and
- $k$ represents a cluster size.

The cluster size is based on the *expected number of duplicates* (i.e., average number of equivalent instances) for each element in a given set. We can estimate the number of duplicates based on the number of sources that provide the elements. For example, if there are five sources providing elements and each source contains no duplicates, the expected number of duplicates for the whole set is five per element. If sources contain duplicates, we need additional information (e.g., statistics in a database catalog) to calculate the expected number of duplicates. For the rest of the discussion, we assume that this statistical information is available.

In addition to the two parameters, we introduce a set of data structures that store temporary information during clustering. Those data structures are:

- $E$ represents a list of $N$ elements to be matched and $E[i], 0 \leq i < N$, is the $i^{th}$ element in the list.
- $A$ represents an assignment list of size $N$; the value of $A[i], 0 \leq i < N$, indicates to which cluster the $i^{th}$ element $E[i]$ is assigned; $A[i]$ is initialized with a negative number (i.e., we use a positive number or zero as an identification number of a cluster).
- $D$ represents a list of distances between two elements; the size of $D$ is $N^2$ for AC heuristics and $N \times \lfloor N/k \rfloor$ for SC heuristics; and the $D[i]$ is initialized to positive infinity.
- $D^*$ represents a sorted list of distances which are originally stored in $D$. 
• \( I \) represents a list of original indices of values in \( D^* \) before sorting; in other words, if the value of \( I[i] \) is equal to \( j \), the value of \( D^*[i] \) was stored in \( D[j] \).

Now we are ready to formally describe the heuristics we have implemented.

### 6.3.1. All-pair-comparison-based heuristic (AC)

When the AC heuristic is used to build the clustering tree, the distances for every pair of elements must be calculated. The clustering tree is constructed as shown in Figure 8.

```plaintext
1: For each 0 < i < j < N do { evaluate \( D[i*N + j] <- \text{distance}(E[i], E[j]) \) } 
2: Sort the distance list \( D \); the resulting of sorting is stored in \( D^* \) and the list of original indices of values in \( D^* \) before sorting is stored in \( I \). 
3: current_cluster <- 0; 
4: For 0 < i < N^2 do 
   5: if \( D[i] < \text{positive\_infinity} \) 
      6: row <- \( i \) div N; 
      7: col <- \( i \) mod N; 
      // \( E[row] \) and \( E[col] \) have not been assigned to any cluster */ 
      8: if (\( A[row] < 0 \) AND \( A[col] < 0 \)) { 
         9: if size of current_cluster > 0 { 
            10: current_cluster <- current_cluster + 1; 
            11: } 
            // only \( E[row] \) has not been assigned to any cluster */ 
         12: assign \( E[row] \) and \( E[col] \) into the current_cluster and update A 
      13: } /* only \( E[row] \) has not been assigned to any cluster */ 
      14: if (\( A[row] < 0 \) AND \( A[col] >= 0 \)) 
         15: if size of the \( A[col] \)th cluster < k 
            16: assign \( E[row] \) into the \( A[col] \)th cluster and update A 
            // only \( E[col] \) has not been assigned to any cluster */ 
      17: if (\( A[row] >= 0 \) AND \( A[col] < 0 \)) 
         18: if size of the \( A[row] \)th cluster < k 
            19: assign \( E[col] \) into the \( A[row] \)th cluster and update A 
            // if both \( E[row] \) and \( E[col] \) have been assigned to some clusters, each of them is closed to some (other) elements */ 
      20: } 
    21: end for 
   // the unassigned element is assigned into a new cluster */ 
22: current_cluster <- current_cluster + 1; 
23: For 0 <= i < N do { if \( A[i] < 0 \) then assign \( E[i] \) into the current_cluster and update A }
```

**Figure 8:** All-pair-comparison-based heuristic.

Theoretically, the run-time complexity of this algorithm is \( O(N^2 \log N) \) because the time taken in step 2 dominates the time taken in all other steps. For practical purposes, the run-time complexity of the algorithm is close to \( O(N^2) \) because the overall cost of calculating distances for each pair of (complex) objects is more expensive than the overall cost of sorting the distances.

### 6.3.2. Selected-comparison-based heuristic (SC)

Unlike AC, which compares every data element with every other element, SC randomly picks a set of pivot elements from among the data elements and compares all other elements to the pivots. Each pivot
represents a cluster that will be generated. SC is based on a fixed cluster size. Using the SC heuristic, the clustering tree is created as shown in Figure 9.

1: With a seed, randomly pick \( \lfloor N/k \rfloor \) distinct pivots from \( E \); let \( P \) be the list of pivots;
2: For each \( 0 < i < N \) AND \( 0 < j < \lfloor N/k \rfloor \) do \( \{ D[i, \lfloor N/k \rfloor + j] \leftarrow \text{distance}(E[i], P[j]) \} \)
3: Sort the distance list \( D \); the resulting of sorting is stored in \( D^* \) and the list of original indices of values in \( D^* \) before sorting is stored in \( I \).
4: For \( 0 < i < N \times \lfloor N/k \rfloor \) do
   5: \( \text{row} \leftarrow i \div \lfloor N/k \rfloor \);
   6: \( \text{col} \leftarrow i \mod \lfloor N/k \rfloor \);
   7: if \( A[\text{row}] < 0 \) AND the size of the \( \text{col} \)-th cluster < \( k \) then assign \( E[\text{row}] \) into the \( \text{col} \)-th cluster and update \( A \)
9: end for
10: For \( 0 < i < N \) do { if \( A[i] < 0 \) then assign \( E[i] \) into the \( \lfloor N/k \rfloor \)-th cluster and update \( A \) }

Figure 9: Selected-comparison-based heuristic.

The run-time complexity of this algorithm is \( O(N^2 \log N/k) \) because the time taken in step 2 dominates the time taken in all other steps. For practical purposes, the run-time complexity of the algorithm is close to \( O(N^2/k) \) because the overall cost of calculating distances for each pair of (complex) objects is more expensive than the overall cost of sorting the distances. Note that SC is faster (by a factor \( k \)) than AC.

6.3.3. M-tree-based heuristic (MT)

As mentioned before, this heuristic is based on the M-tree as a clustering tree. The term “heuristic” is used to refer to the algorithm, which constructs the M-tree. Our main rational for implementing the MT heuristic is to use it as a benchmark for the efficiency and accuracy of the other heuristics. Using the MT heuristic, the cluster size is variable but the maximum cluster size, which corresponds to the size of an M-tree node, is fixed. The maximum cluster size is obtained in the same way as the fixed cluster size in AC and SC.

An M-tree consists of a set of nodes each of which contains a list of entries and a pointer to its parent. Each entry in an internal node stores a routing-object that is used for searching a place in the tree where to insert a new object (note that during the clustering process we only need to insert new objects but never remove existing objects). In addition, the entry of the internal node contains a pointer to the root of the sub-tree, the covering radius of the routing-object, and the distance of the routing-object to its parent. The covering radius is the maximum distance from the routing-object to its children. Each entry in a leaf node of an M-tree stores an object or a pointer to the object and the distance from the object to the routing-object stored in the parent node.
Using the M-tree heuristic, the clustering tree is created by inserting \( N \) elements into an empty tree one by one. The insertion procedure works as follows:

1. Let \( e \) be the element to be inserted. Starting from the root, if the root is empty go to step 4. Otherwise, compare \( e \) to the routing-objects to obtain the path to the target node where \( e \) should be stored.

2. If the target node is a non-leaf node, let the target node be the root of the current subtree, and continue with step 1.

3. If the target node is a leaf and is full, the node overflows and must be split. Let \( y \) be the full node. To split \( y \), create a new node, \( z \), and transfer some entries from \( y \) to \( z \) using a partitioning algorithm (e.g., generalized hyperplane described in Section 2). Now \( y \) is a non-empty node.
   a. If \( y \) is the root of the tree, allocate a new root node, \( r \). Then, insert \( y \) and \( z \) into \( r \) (by recursively calling the insertion procedure).
   b. Otherwise, promote \( z \) to the parent of \( y \) (i.e., insert \( z \) into the parent of \( y \) by recursively calling the insertion procedure).

4. If the target node is a leaf and is not full, place \( e \) into one of the entries of the target node.

5. Update the covering radius of each routing-object using backtracking.

The performance of the M-tree heuristic depends on the policy that is used for splitting the tree (step 3 in the insertion algorithm). Ideally, the splitting policy should promote routing objects and partition other objects so that a routing object has a small covering radius and little overlap with other routing objects. Ciacia et al. [14] introduce several strategies for picking routing objects. Our M-tree heuristic uses a random strategy to pick two routing objects from the set of objects in the overflow node, which is fast, simple, and easy to implement. In the same step, we also transfer some of the entries in the overflow node to a new node using the \textit{generalized hyperplane decomposition} described earlier. We chose this decomposition strategy since experiments by Ciacia et al. have shown that it performs well in conjunction with the random strategy for splitting the overflow nodes. The run-time and space complexities of the MT heuristic are \( O(N \log N) \), which is the time it takes to insert \( N \) elements into an M-tree whose height is \( \log N \).
6.4 Variations of SC

In certain situations, using a fixed cluster size may result in a large number of false dismissals; for example, when elements cannot be placed into the cluster they belong because the cluster is already full. Furthermore, it is possible that an element is related to more than one category (e.g., a data mining book can also be considered a database book), and should be placed into more than one cluster. In this situation, a heuristic that supports only non-overlapping clusters will also produce more false dismissals. To remedy the first problem, we have designed a variation called Selected-Comparison-based heuristic with Variable cluster size (SCV). To remedy the second problem, we have designed another variation called Selected-Comparison-based heuristic with Overlapping clusters (SCO).

In both variations, we modify the distance list $D$ to be a distance matrix of size $N \times \left\lceil N/k \right\rceil$ to store distances between an element and a pivot; the $D[i,j]$ is initialized to positive infinity. An additional data structure that we use for each clustering is $minD$, which is a list of size $N$, where $minD[i]$ stores the minimum distance of all distances in the $i^{th}$ row of $D$.

![Algorithm](image)

Figure 10: Selected-comparison-based heuristic with variable cluster size.

6.4.1. SC heuristic with variable cluster size (SCV)

When this heuristic is used to build the clustering tree, it estimates the number of clusters that will be created based on an expected cluster size ($k$). The expected cluster size indicates the average number of elements that can be placed into a cluster and is obtained in the same way as the cluster size in the AC and SC heuristics. Using SCV, the clustering tree is created as shown in Figure 10.

The run-time and space complexity of this algorithm are $O(N^2/k)$. For practical purposes, the time taken in step 1 dominates the time taken in step 2 because the overall cost of calculating distances for each pair of (complex) objects is more expensive than the overall cost of finding minimum distances.
6.4.2. SC heuristic with variable cluster size and overlapping of clusters (SCO)

Using SCO, the clustering tree is created as shown in Figure 11.

```plaintext
1: With a seed, randomly pick \( \lceil N/k \rceil \) distinct pivots from \( E \); let \( P \) be the list of pivots;
2: For \( 0 \leq i < N \) AND \( 0 \leq j < \lceil N/k \rceil \) do \( \{ D[i][j] <- \text{distance}(E[i], P[j]) \} \)
3: For \( 0 \leq i < N \) do \( \text{minD}[i] <- \text{the minimum distance of all distances in the } i^{th} \text{ row of } D. \)
4: For \( 0 \leq i < N \) do
5: For \( 0 \leq j < \lceil N/k \rceil \) do
6: if \( D[i][j] = \text{minD}[i] \) then
7: assign \( E[i] \) into the \( j^{th} \) cluster
9: end for
8: end for
10: end for
```

Figure 11: Selected-comparison-based heuristic with overlapping cluster.

It is important to note that overlapping clusters can reduce the number of false alarms by producing a higher number of correct matches between (equivalent) elements in the same clusters. However, cluster overlap can also produce more false dismissals since more than one copy of an element can be placed in different clusters. The analysis in Sec. 7 will demonstrate that SCO results in a lower overall error when compared to the original SC. Finally, the run-time and space complexities of this algorithm are \( O(N^2/k) \). For practical purposes, the time taken in step 1 is dominating because the overall cost of calculating distances for each pair of (complex) objects is higher than the overall cost of finding minimum distances.

7 Experimental Evaluation

Our qualitative analysis validates the time complexity of the clustering algorithms on different data sets and demonstrates the accuracy of the element matching. All of the experiments were carried out on a Pentium III PC with 256MB of RAM, running Windows NT Workstation 4.0. The algorithms have been implemented in Java (SDK 1.3); other major software tools used in our implementation are the Oracle XML Parser v. 2.0.2.9 [47] and the XML-QL processor v. 0.9 from AT&T Research Lab [5].

7.1 Description of the Test Data

The data used in our experiments is based on a sample bibliography database which was developed as a test database at AT&T Research [5]. Each publication in the bibliography database is represented as an XML element. From this original data set, we created our test data as follows: We selected two subsets and generated duplicate elements as well as modified some of the existing data values (e.g., we randomly added extra characters to some of the string values to simulate inconsistencies and spelling conflicts). Initially, the two subsets contained 20 articles and 20 Ph.D. dissertations, respectively. Each element in a data set was
assigned a unique identifier, which was used to count the number of matching errors. Since the outcome of the experiments was the same for both data sets, we only describe the experiments for the set containing article data. A description of the experiments using the second data set can be found in [50].

```xml
<!ELEMENT Article ( Author+, Title, Year, Month?, Pages?, Note?, Journal )>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Author (Firstname?, Lastname, Address?)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>
<!ELEMENT Journal (Title, Year?, Month?, Volume?, Number?)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Note (#PCDATA)>
<!ELEMENT Number (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Volume (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
```

**Figure 12:** Sample DTD describing the structure of the concept “Articles”.

The structure of Article elements is shown in Figure 12. Each Article contains one or more of Author, one of Title, Year, and Journal, as well as zero-or-one of Month, Pages and Note, respectively. Each Author contains one Lastname as well as an optional Firstname and an optional Address. Each Journal contains one Title and optional Year, Month, Volume and Number.

Based on the original subset of 20 different Article instances, we created fifteen sample sets, which contain 50, 60, 70, 80, 90, 100, 150, 200, 300, 400, 800, 1200, 1600, 2000, and 4000 elements, respectively, by generating duplicate elements and modifying some of the existing data values. The average number of duplicates for each data set is shown in column 2 of Table 1. In addition, the edit distance between a data element and its duplicate is two. For each synthetic data set, we count the number of errors (false dismissal and false alarms) that can be observed if each element is placed into a separate cluster. We divide the number of errors by the total number of comparisons and refer to that number as the *base error* shown in column 3 of Table 1. The base error serves as a benchmark when measuring the accuracy of our heuristics relative to the manual matching process, which was simulated through the procedure of counting the number of errors on the synthetic data set.
7.2 Experimental Results and Discussion

We tested all five heuristics on the “Article” data sets described in Table 1. The cluster size varies from 2% to 20% of the total data set. However, here we only show the experimental results for cluster sizes between 2% and 7%, since the number of errors for cluster sizes larger than 7% exceeds the base error. In addition, our experiments have shown that cluster sizes between 2% and 3% provide the most accurate matching results. Finally, we did not perform experiments on the 50-element data set using a 2% cluster size since each cluster would contain only one element.

Table 1: Characteristics of synthesized “Article” data sets.

<table>
<thead>
<tr>
<th>Size of data set</th>
<th>Avg. number of duplicates per element</th>
<th>Base error (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2.5</td>
<td>4.16</td>
</tr>
<tr>
<td>60</td>
<td>3.0</td>
<td>5.20</td>
</tr>
<tr>
<td>70</td>
<td>3.5</td>
<td>4.50</td>
</tr>
<tr>
<td>80</td>
<td>4.0</td>
<td>5.06</td>
</tr>
<tr>
<td>90</td>
<td>4.5</td>
<td>4.99</td>
</tr>
<tr>
<td>100</td>
<td>5.0</td>
<td>4.67</td>
</tr>
<tr>
<td>150</td>
<td>7.5</td>
<td>4.85</td>
</tr>
<tr>
<td>200</td>
<td>10.0</td>
<td>4.93</td>
</tr>
<tr>
<td>300</td>
<td>15.0</td>
<td>4.94</td>
</tr>
<tr>
<td>400</td>
<td>20.0</td>
<td>4.90</td>
</tr>
<tr>
<td>800</td>
<td>40.0</td>
<td>5.08</td>
</tr>
<tr>
<td>1200</td>
<td>60.0</td>
<td>5.03</td>
</tr>
<tr>
<td>1600</td>
<td>80.0</td>
<td>5.00</td>
</tr>
<tr>
<td>2000</td>
<td>100.0</td>
<td>4.98</td>
</tr>
<tr>
<td>4000</td>
<td>200.0</td>
<td>4.99</td>
</tr>
</tbody>
</table>

For heuristics with variable cluster size, the cluster size shown is the expected cluster size, which indicates the average number of elements that can be placed into a cluster. In the heuristics based on SC, the number of clusters that will be generated is the total number of elements in the data set divided by the expected cluster size. The expected cluster size is derived from the expected number of duplicates. For experimental purposes, we assume that the expected number of duplicates is given. In a real integration system, a lower bound for the expected number of duplicates may be the number of sources that contribute data elements to the result.
Figure 13: Time complexities for the clustering heuristics using 2% (left) and 3% (right) of the total data set as cluster size.

The first part of our analysis measures the time complexity. Figure 13 depicts the time complexity for all five heuristics using cluster sizes between 2% and 3% of the total data set. The SC heuristic and its variants (SCV and SCO) perform almost identical in all cases as we have stated in Section 6. In each case, the MT heuristic performs fastest, independent of the data set. As expected, the AC heuristic performs worst for data sets with more than 100 elements.

Figure 14: Time complexities for AC, SC, SCV, and SCO using 2% (left) and 3% (right) of the total data set as cluster sizes.

To better understand the behavior of the different heuristics on small data sets (50-100), Figure 14 shows the time complexities for AC, SC, SCV, and SCO using cluster sizes of 2% and 3% of the total data set. Our first observation is that all heuristics based on SC perform almost identically. The second
observation is that the time complexity of the SC family depends on the cluster size. The third observation is that AC performs fastest for small data sets (e.g., for data sets of size between 50 and 60 elements), since it has the smallest overhead of all heuristics described here (e.g., no pivot is picked). Since we have developed this model to support data reconciliation of query results in integration systems, we expect most data sets to fit into this size range.

![Graph 1](image1.png)

**Figure 15:** Number of false alarms produced by each clustering heuristic using 2% (left) and 3% (right) of the total data set as cluster sizes.

In the second part of the analysis, we concentrate on the accuracy of our heuristics. Figure 15 displays the number of false alarms produced by each heuristic under different cluster sizes. The experiments show that AC produces the fewest number of false alarms for data sets of up to 150 elements with cluster size 2%, and data sets of up to 100 elements with cluster size 3% of the total size. The number of false alarms that are produced by AC increases significantly for large data sets. The reason is that AC partitions the elements based on the sorted order of the distances between two elements. False alarms may occur as follows: the distance between an element and its duplicate is longer than the distance between the element and a non-duplicate element. Therefore, two different elements are forced into the same cluster.

Note for large data sets, MT and SCV provide the fewest number of false alarms when the expected cluster sizes are 3% and 2% of the total size, respectively. Both MT and SCV use variable cluster sizes, which increase the chances that an element and its duplicates are placed into the same cluster. Using SCV, the cluster size determines the number of clusters, each of which is represented by a pivot. When the cluster size increases, the number of clusters decreases, which increases the possibility of false alarms. Using MT,
the number of clusters is independent of the cluster size. An MT node is split when the node is full. However, splitting preserves the property that all elements inside the node are close to each other; hence, fewer false alarms occur.

**Figure 16**: Number of false dismissals produced by each clustering heuristic using 2% (left) and 3% (right) of the total data set as cluster sizes.

Figure 16 shows the number of false dismissals produced by each heuristic using two different cluster sizes. The experiments show that SCV produces the fewest number of false dismissals followed by SCO. In both cases, the cluster size is unbounded, meaning that some clusters can have a larger size than the expected cluster size. This increases the probability of placing an element and its copies into the same cluster, which reduces the number of false dismissals. Using SCO, clusters can also overlap. In general, overlapping clusters can not only reduce the number of false alarms since they allow for more correct matches between the (equivalent) elements in the same cluster, but can also produce more false dismissals since a copy of an element can be placed into more than one cluster. Our results validate this expected behavior.

Another observation is that when the size of the data set increases, the number of false dismissals increases for the AC heuristic. This is because AC places a pair of elements into a cluster in the order of the distances between those two elements. False dismissals may occur as follows: A pair of elements (i.e., an element and one of its duplicates) is placed into a non-empty cluster filling the cluster to capacity (fixed cluster size). Other copies of the element are subsequently forced into in another cluster.
Figures 17 and 18 show the percent errors produced by each heuristic on different cluster sizes. The total error includes the number of false alarms and false dismissals. The experiments show that SCV provides the most accurate matching results across all data sets. It is interesting to note that AC provides the most accurate matching results for data sets of size 70 elements or less, as shown in Figure 18. The reason is that AC has “global knowledge” about the distances between every pair of elements. Those distances are used to place elements into clusters. Note, AC is not appropriate for the data set whose size is larger than 100, since its total error is larger than the base error.
As we can see, AC performs fastest and provides the most accurate result for data sets of 70 elements or less. SCV provides the most accurate result for data sets with more than 70 elements. The remaining analysis will focus on AC for data sets of size 70 and on SCV for data sets of size 400; we believe that most of the data sets gathered from sources in actual integration systems fit into these ranges.

### Figure 19: Matching errors produced by the AC heuristic on a data set of size 70 elements (left) and by the SCV heuristic on a data set of size 400 elements (right).

Figure 19 shows the relationships among false alarms, false dismissals and total error when using AC and SCV with variation in the cluster size for data sets of size 70 and 400 elements, respectively. We can observe that when the cluster size increases, the number of false alarms also increases, whereas the number of false dismissals decreases. The reason is that, in AC, increasing the cluster size forces more elements into the same cluster, which increases the probability that two different elements are co-located. This in return increases the number of false alarms. However, it also increases the probability that an element and its duplicates are co-located, which in return decreases the number of false dismissals. In SCV, increasing the cluster size reduces the number of clusters, which decreases the probability of placing two different elements into different clusters. This increases the number of false alarms. However, it also decreases the probability that an element and its duplicates are placed into different clusters, which decreases the number of false dismissals.

### Table 2: Percentage of matching errors produced by AC on 70 elements with a base error of 4.47% (total of 2,415 comparisons).

<table>
<thead>
<tr>
<th>Expected Cluster size</th>
<th>False alarms (%)</th>
<th>False dismissals (%)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2% of total size (2 elements per cluster)</td>
<td>0.18</td>
<td>3.00</td>
<td>3.18</td>
</tr>
<tr>
<td>3% of total size (2 elements per cluster)</td>
<td>0.18</td>
<td>3.00</td>
<td>3.18</td>
</tr>
<tr>
<td>5% of total size (3 elements per cluster)</td>
<td>2.83</td>
<td>2.85</td>
<td>5.68</td>
</tr>
<tr>
<td>7% of total size (5 elements per cluster)</td>
<td>2.83</td>
<td>2.85</td>
<td>5.68</td>
</tr>
</tbody>
</table>
Table 3: Percentage of matching errors produced by SCV on 400 elements with a base error of 4.96% (total of 79,800 comparisons).

<table>
<thead>
<tr>
<th>Expected Cluster size</th>
<th>False alarms (%)</th>
<th>False dismissals (%)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2% of total size (8 elements per cluster)</td>
<td>0.63</td>
<td>2.50</td>
<td>3.13</td>
</tr>
<tr>
<td>3% of total size (12 elements per cluster)</td>
<td>1.34</td>
<td>1.70</td>
<td>3.04</td>
</tr>
<tr>
<td>5% of total size (20 elements per cluster)</td>
<td>4.63</td>
<td>1.43</td>
<td>6.06</td>
</tr>
<tr>
<td>7% of total size (28 elements per cluster)</td>
<td>8.74</td>
<td>1.04</td>
<td>9.78</td>
</tr>
</tbody>
</table>

The error percentages produced by AC and SCV as shown in Figure 19 are listed in Tables 2 and 3. The AC heuristic can reduce the total error by 29% with respect to the base error (4.47%) using cluster sizes of 2% and 3% of the total data size; the SCV heuristic can reduce the total error by 36% and 38% with respect to the base error (4.90%) when using a cluster size of 2% and 3% of the total data size, respectively. In addition, during the course of our experiments we have observed that the decrease in the number of false dismissals is greater than the increase in the number of false alarms, which was one of the goals of our research. The reduction in false alarms (false dismissals) was measured by comparing the false alarms (false dismissals) generated by our clustering heuristics with the false alarms (false dismissals) produced when no clustering was used.

Note that each heuristic needs two input parameters, namely a set of data elements and a cluster size. An interesting question is: “How large should the cluster size be to minimize the matching error?” According to tables 2 and 3, both AC and SCV provide the most accurate results when the cluster size is around 2% of the total data set size.

![Figure 20](image.png)

Figure 20: Matching errors produced by SCV heuristic on data sets ranging in size between 400 and 4000 elements using 2% (left) and 3% (right) of each data set as cluster sizes.
Figure 20 displays the number of errors produced by SCV using two different cluster sizes on data sets of size between 400 and 4000 elements. Data sets of these sizes are typical when searching the Web, for example. The result of this last experiment verifies that clusters with size 2% of the total size provide the most accurate result for large data sets.

In accordance with Figure 20, the number of false alarms, false dismissals, and total errors produced by SCV are listed in Table 4. It is interesting to note that on data size of size 2400 elements the SCV heuristic can reduce the total error by 55% with respect to the base error (4.99% as shown in Table 1) using cluster size of 2% of the total size.

Table 4: Percentage of matching errors produced by SCV on data sets ranging in size between 400 and 4000 elements using 2% and 3% of each data set as cluster sizes.

<table>
<thead>
<tr>
<th>Size of data set (items)</th>
<th>Cluster size of 2% of the total size</th>
<th>Cluster size of 3% of the total size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>False alarms (%)</td>
<td>False dismissals (%)</td>
</tr>
<tr>
<td>400</td>
<td>0.63</td>
<td>2.50</td>
</tr>
<tr>
<td>800</td>
<td>0.56</td>
<td>2.59</td>
</tr>
<tr>
<td>1200</td>
<td>0.47</td>
<td>2.61</td>
</tr>
<tr>
<td>1600</td>
<td>0.36</td>
<td>2.52</td>
</tr>
<tr>
<td>2000</td>
<td>0.15</td>
<td>2.56</td>
</tr>
<tr>
<td>2400</td>
<td>0.03</td>
<td>2.27</td>
</tr>
<tr>
<td>2800</td>
<td>0.23</td>
<td>2.32</td>
</tr>
<tr>
<td>3200</td>
<td>0.16</td>
<td>2.55</td>
</tr>
<tr>
<td>3600</td>
<td>0.26</td>
<td>2.52</td>
</tr>
<tr>
<td>4000</td>
<td>0.49</td>
<td>2.54</td>
</tr>
</tbody>
</table>

7.3 Usage Guidelines

Based on the experiments, we make the following two usage recommendations for the different clustering strategies:

1. For a small data set with 70 elements or less, we recommend using the AC heuristic with 2% cluster size, since AC performs fastest and produces the most accurate results.

2. For data sets with significantly more than 70 elements, we recommend using the SCV heuristic with 2% cluster size, since SCV performs fast (faster than AC but slower than MT) and produces the most accurate matching results.

A detailed description of the results and additional recommendations, which have been implemented in the IWIZ prototype system, can be found in [50].
8 Conclusions and Future Research

An important challenge when integrating information from multiple sources, for example in mediation and data warehousing systems, is to resolve the semantic and schematic heterogeneities that exist among related elements efficiently and with as little human intervention as possible. Heterogeneities are due to the fact that a real-world entity can be represented in many different ways in different contexts [37]. Despite the advances in the field of information management, resolving different types of heterogeneities remains a major, ongoing concern. In this article, we have described an approach and infrastructure for identifying and merging related data elements from multiple information sources. Specifically, we have developed a multi-strategy clustering model to perform element matching across related sources containing data represented using XML. We have shown that our model has met three important requirements, namely qualitative compatibility (i.e., the quality of the automatic detection of similar data elements is comparable to the quality of the manual detection process), computational complexity (i.e., our reconciliation algorithm performs in polynomial time and space), and scalability (i.e., our multiple clustering strategy is scalable with respect to the size of data set, the number of data sources and the number of features per element).

The work described in this article contributes to the state-of-the-art in information integration in the following three important ways:

- We have developed a classification scheme for structural and semantic conflicts among related elements in XML-based data sources. The classification serves as a basis for our approach to matching related elements in different contexts.

- We have developed a multi-strategy clustering model (MCM) to match related elements from different contexts. Our model contains a set of distance functions for measuring the semantic distance between any pair of elements, data structures and operations for representing and manipulating the different clustering trees, and five clustering heuristics, for matching related elements and grouping them into clusters depending on the characteristics and distribution of the underlying data.

- We have illustrated how our clustering model can be integrated with other components (data cleaner, data joiner, etc.) to identify and reconcile XML query results from multiple sources inside a mediator. Our test mediator is part of the Integration Wizard (IWIZ) prototype system,
which allows end-users to access and retrieve information from multiple heterogeneous sources through a consistent, integrated view.

The results of our qualitative analysis of the clustering heuristics have shown that computer assisted reconciliation in integration systems is feasible and can significantly enhance usability of integration systems by reducing the amount of human input that is necessary to produce the integrated result. The evaluation of our clustering model is based on (1) time complexity of the algorithms, and (2) accuracy of the result (i.e., the percentage of clustering errors that are produced during the matching). We have tested our clustering model on data sets of different size with varying number of duplicates. For data sets with 70 elements or less the all-pair-comparison-based heuristic (AC) with 2% cluster size produces the most accurate result. For data sets with more than 70 elements, the selected-comparison-based heuristic with variable cluster size (SCV) with 2% maximum cluster size represents the best compromise between speed and accuracy. In comparison, the M-Tree-based heuristic was faster but less accurate.

Since the concept of clustering data elements in the context of data integration is relatively new, there are many opportunities for future extensions to our research. For example, our multi-strategy clustering model currently exists only in the form of a prototype. It is still lacking a rigorous formal representation, which is needed when trying to apply this approach to other domains and data models, for example, to reconcile relational data. Another extension is to experiment with additional clustering heuristics and data structures to see if further improvements in speed and accuracy are possible. Still other optimizations are possible: for example, given the current heuristics, the clustering tree is built under the assumption that the tree structure and all data elements to be reconciled can fit into memory. More research is necessary to develop heuristics and data structures capable of reconciling data elements on disk or other storage media.

Finally, we see important research opportunities in enhancing the usability and scalability of our clustering model and more generally those of the data merge engine (DME) in the IWIZ mediator. In its current form, DME is dependent on domain experts for configuring the various parameters that are needed to cleanse, cluster, and unify related elements. Although our prototype system uses a pre-defined default set, human feedback is needed to fine-tune the clusters in an iterative fashion. Several improvements are
possible. One can include machine-learning techniques capable of remembering the parameter settings for different application domains and using this knowledge to automatically configure DME for future use.

Another way to reduce human effort is to reduce the size of the training set that is currently used during the configuration of the DME during built-time. Rather than using probe queries, which are generated simply based on concepts in the global schema, we believe that data sampling techniques may help generate better training sets. Specifically, we envision that sampling will involve both mediator and wrappers as follows: The mediator initiates probe query requests to each wrapper containing the sampling parameters that specify types and percentage of desired data elements relative to the size of the source. In the wrappers, which have knowledge about the source contents, the sampling parameters are interpreted, the corresponding probe query is generated and executed, and the result is returned back to the mediator. We are in the process of implementing the query-sampling enhancement outlined above and will continue to report on our progress in future conferences and workshops.

Acknowledgements
The authors thank the other members of the IWIZ team for the time and effort spent in designing the overall architecture and implementing the current IWIZ prototype. We also like to thank Profs. Stanley Su, Herman Lam, and Abdelsalam (Sumi) Helal for the many fruitful discussions on the different clustering models. Last but not least, we would also like to thank the anonymous reviewers for their insightful comments, which helped us improve the organization and clarity of our presentation.

References


[35] V. Kashyap and A. Sheth, Semantic and Schematic Similarities between Objects in Databases: A Context-based approach, Department of Computer Science, University of Georgia, Atlanta, Georgia, February 1995.


