A Survey on Accessing Dataspaces

Yihan Wang
Tsinghua University, Beijing, China
yihanwang@tsinghua.edu.cn

Shaoxu Song*
Tsinghua University, Beijing, China
sxsong@tsinghua.edu.cn

Lei Chen
The Hong Kong University of Science and Technology
leichen@cse.ust.hk

ABSTRACT

Dataspaces provide a co-existence approach for heterogeneous data. Relationships among these heterogeneous data are often incrementally identified, such as object associations or attribute synonyms. With the different degree of relationships recognized, various query answers may be obtained. In this paper, we review the major techniques for processing and optimizing queries in dataspaces, according to their different abilities of handling relationships, including 1) simple search query without considering relationships, 2) association query over object associations, 3) heterogeneity query with attribute correspondences, and 4) similarity query for similar objects. Techniques such as indexing, query rewriting, expansion, and semantic query optimization are discussed for these query types. Finally, we highlight possible directions in accessing dataspaces.

1. INTRODUCTION

A dataspace system [8, 10] processes data, with various formats, accessible through many systems with different interfaces, such as relational [11], sequential [30], XML [32], RDF [31], etc. Unlike data integration over DBMS, a dataspace system does not have full control on its data, and gradually integrates data as necessary.

Dataspace is a data co-existence approach, which provides base functionality over various data sources, regardless of how integrated they are. A dataspace system may return best-effort approximate answers, from multiple sources, where a set of correct semantic mappings have not been applied. In addition, dataspace query answering also takes into consideration a sequence of earlier queries leading up to it, not only for optimizing potential future queries (see [26]), but also for creating better semantic integration between sources in a dataspace [10].

Dataspace systems could apply existing approximate or uncertain mappings and keyword search techniques, however, as also indicated in [10], these works need to be generalized considerably to cases where we do not have semantic mappings of sources and where the data models of the sources differ. In particular, since dataspaces are loosely coupled, rather that providing exact answers, the goal of dataspaces is best-effort query answering.

Some of the challenges in accessing dataspaces have been (partially) addressed, such as studying a sequence of earlier queries (for query optimization and potentially better semantic integration), or ranking answers from multiple sources with various levels of semantic mappings. Other challenges remain open, e.g., handling inconsistencies in dataspaces. (See a detailed discussion in Section 7).

In this paper, we focus on search queries for accessing dataspaces, i.e., returning items already existing in the dataspace. Since well-established semantic mappings are often unavailable in dataspaces, more complex SPJ queries with transformations and views are not considered in this survey. Being aware of different levels of connections, we categorize recently proposed techniques for searching dataspaces into four potential categories.

Simple Search Query

Since the relationships between data objects may barely be obtained, it is necessary to provide an elementary way to access all the data in a dataspace. Simple search query, e.g., keyword queries (with or without attribute names) or attribute predicate queries, is a good choice to meet such requirements [16]. It returns a set of dataspace resources (objects) that directly match the query predicates without considering associations. Inverted index is thus naturally extended to dataspaces for efficient keyword query processing (see details in Section 3).

Association Query

A number of associations among objects may be naturally embedded in dataspace initialization, such as the tree relationships between file objects in a

*Corresponding Author

personal information management dataspace. The object associations could also be incrementally specified, e.g., all the persons graduated from the same university, known as *association trails*. These associations between objects are often modeled in a graph structure. Association query, performed on the association graph, returns not only the objects matching the query specified contents, but also those related objects connected via associations. Extensions on index for associations are discussed as well in Section 4.

Heterogeneity Query

Besides associations among objects, relationships between heterogeneous attributes could also be considered when querying a dataspace. For example, the matching between attributes "manu" and "prod" indicates that both of them specify similar information about manufacturers or producers. Such attribute matching is often recognized by schema matching techniques [18]. As mentioned, in dataspaces, a pay-as-you-go style [12] is usually applied to gradually identify attribute matching (according to users' feedback when necessary). The heterogeneity query extends query results by considering the matching relationships between heterogeneous attributes. For instance, a heterogeneity query on attribute manu searches not only the mentioned manu but also the identified prod attribute. Query rewrite is often employed for such query expansion (see details in Section 5).

Similarity Query

Rather than simply specifying keywords, a more advanced query may pose an object and return dataspace objects that are similar to the query object, known as similarity query. To accelerate similarity query processing, semantic query optimization [3, 13] can be employed. It relies on data dependencies introduced in dataspaces for query rewriting (in Section 6).

The remainder of this paper is organized as follow. In Section 2, we introduce models for representing dataspaces. The aforesaid four types of queries are discussed from Sections 3 to 6, respectively. Finally, we summarize this paper in Section 7 and discuss possible future directions.

2. MODELING

Owing to heterogeneity, dataspaces need to employ an elementary model to represent the most common part of data from various sources, and provide a very basic accessing way to begin with.

2.1 Data Model

The data model indicates how the contents of objects in dataspaces are organized. Usually, the objects consist of values on several attributes.

2.1.1 Triple Store

Since the objects in a dataspace are collected from various heterogeneous sources with distinct attributes, it is obviously inappropriate to manage the data as tables with fixed columns like the relational model. Instead, the data are often modeled as a collection of triples [12, 5], in the form of

$$\langle object, attribute, value \rangle.$$

For example, Table 1 presents 6 triples, recording the (School)Color attribute values of 6 objects.

Table 1: Triples in a dataspace

	$\langle object, attribute, value \rangle$
t_0	⟨Wisconsin, SchoolColor, Cardinal⟩
t_1	⟨Cal, SchoolColor, Blue⟩
t_2	⟨Washington, SchoolColor, Purple⟩
t_3	⟨Berkeley, Color, Navy⟩
t_4	⟨UW-Madison, Color, Red⟩
t_5	⟨Stanford, Color, Cardinal⟩

It is worth noting that the triples can be converted to graph representation. In order to present examples more intuitively, we use graph representation by default in the remainder of this paper.

2.1.2 Resource Views

The triple store scatters attribute values of an object, which may not be efficient in object oriented retrieval. The iMeMex Data Model (iDM), specialized for personal data management [4, 20], introduces an object oriented data model, by grouping the attribute-value pairs of an object together, known as *resource views*.

Table 2: Components of a resource view RV_i

Component	Description
$RV_i.name$	Name of a resource view RV_i
$RV_i.tuple$	Set of attribute value pairs $(att_0 : value_0), (att_1 : value_1), \ldots$
$RV_i.content$	Finite byte sequence of content

Table 2 lists the components that can be attached to a resource view RV_i. Besides the set of attribute-value pairs stored in $RV_i.tuple$, other components,

such as a finite byte sequence of content (a file in personal information management), could be further attached to a resource view.

2.2 Object Association

As mentioned in the introduction, we cannot identify ahead of time all the associations of objects in dataspaces. Ad hoc recognized associations among objects are managed in an extraordinary manner.

2.2.1 Association Graph

In iDM [20], the associations are modeled as a graph, $G := (\mathcal{RV}, E)$, where $\mathcal{RV} := \{RV_1, \ldots, RV_n\}$ denotes a set of n resource views (nodes). E is a sequence of directed edges between resource views.

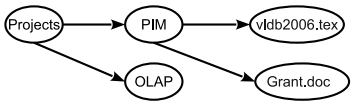

Figure 1: An example graph of associations among resource views (objects)

For instance, Figure 1 presents an example association graph of 5 resource views. The edges denote the associations between the corresponding resource views, e.g., (Projects→PIM) indicates that PIM is a project under the directory of all Projects in a personal file management dataspace.

2.2.2 Association Triple

The association between objects in a dataspace can be represented as a collection of triples as well, in the form of $\langle object, association, object \rangle$ [5]. Unlike the edges without any label in the aforesaid association graph, a type is indicated in the association triple for the association between two objects.

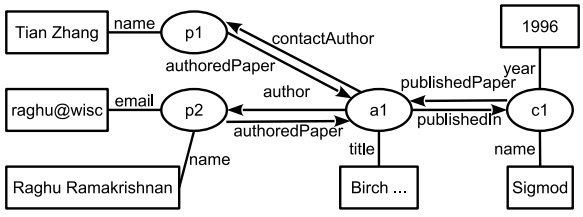

Figure 2: An example triple base

Figure 2 illustrate an example dataspace of 4 objects (p1, p2, a1 and c1, denoted by ellipses). Rectangles attached to each object denote its attribute values, e.g., "Tian Zhang" attached to p1 through an edge with label "name", representing an attribute value pair (name:Tian Zhang) of object p1. Each association triple corresponds to an edge in Figure 2 with a label indicating its association type. For example, ⟨a1, contactAuthor, p1⟩ means that a1 is connected to p1 as a contactAuthor.

2.2.3 Association Trail

Rather than representing specific single associations, an *association trail* [19], denotes a group of associations by a join predicate θ. Instead of originally embedded in data, such associations can be gradually declared.

Let $Q_L(G)$ and $Q_R(G)$ be two collections of objects specified by queries Q_L and Q_R, respectively, over an association graph G. An association trail $\Phi : Q_L \overset{\theta(l,r)}{\Longrightarrow} Q_R$ represents all the associations from objects in $Q_L(G)$ to objects in $Q_R(G)$ according to $\theta(l, r)$. It conceptually introduces in the association graph a directed edge from left to right and labeled Φ, for each pair of nodes given by $Q_L \bowtie_\theta Q_R$, namely intensional edge. Association trails cover relational and non-relational theta-joins as special cases. A bidirectional association trail $\Phi : Q_L \overset{\theta(l,r)}{\Longleftrightarrow} Q_R$ represents associations in both directions.

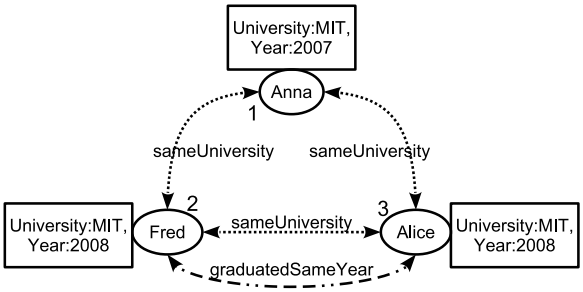

Figure 3: Example association trails

Consider the example in Figure 3. Let ellipses denote objects, while the rectangle attached to each object lists its attribute-value pairs. A bidirectional association trail is given by

$$sameUniversity : class = person \overset{\theta(l,r)}{\Longleftrightarrow} class = person,$$
$$\theta(l, r) : (l.university = r.university).$$

Arrows with dotted lines, labeled sameUniversity, represents 3 intensional edges introduced by the association trail. For instance, the intensional edge between Anna and Fred indicates that they share the same *university* attribute value.

2.2.4 Other Associations

Besides the aforesaid general associations between objects, a special category of "referenceOf" relationships in personal data are considered, namely *Context-Based Reference (CR)* [14]. Such associations are generated by user behaviors. Similar to association trails, the CR associations could be introduced gradually and represented in association graphs as well.

2.3 Semantic Correspondences

While associations represent the relationships between objects, the attribute correspondences indicate the identity relationships of attributes.

2.3.1 Attribute Synonyms

Synonym correspondence between two attributes (e.g., manu vs. prod) can be identified by schema matching techniques in data integration (see [18] for a survey). In dataspaces, the synonym correspondence between attributes are often incrementally recognized in a pay-as-you-go style [12]. Automated mechanisms such as schema matching and reference reconciliation provide initial correspondences, termed candidate matches, and then user feedback is used to incrementally confirm these matches (when necessary).

Two attributes A, B having synonym correspondence are denoted by $A \leftrightarrow B$, e.g., manu $\leftrightarrow$ prod. There may exist multiple attributes having synonym correspondences to an attribute A. A synonym table is introduced in [5] for attributes with correspondence. If attribute A is referred to as $A_1, \ldots, A_n$ in different data sources, having $A \leftrightarrow A_1, \ldots, A \leftrightarrow A_n$, the canonical name of A is chosen as one of $A_1, \ldots, A_n$.

2.3.2 Trails

Instead of the symmetric synonyms relationships between attributes, the concept of *trail* is also proposed to specify asymmetric correspondences [20].

A unidirectional trail is denoted as $\Psi : Q_L \rightarrow Q_R$. It means that the query on the left Q_L induces the query on the right Q_R, i.e., whenever we query for Q_L we should also query for Q_R.

For example, a trail

$$\Psi_1 : //*.tuple.created \rightarrow //*.tuple.date$$

indicates that whenever querying objects (resource views) on attribute *created*, it also needs to query objects on attribute *date*.

A bidirectional trail is denoted as $\Psi : Q_L \leftrightarrow Q_R$. It further indicates that the query on the right Q_R induces the query on the left Q_L.

Indeed, trails can specify correspondences between more general queries. For example, they could be used to transform a simple keyword query into a query to the mediated data source. Consider a trail "$dataspace \rightarrow //Projects/PIM/*$". With such a trail, the query will not only return resources containing keyword *dataspace*, but also "vldb2006.tex" and "Grant.doc" in Figure 1 (although the keyword *dataspace* does not appear in these documents).

Traditional data integration approaches, such as GAV, LAV, and GLAV, need high upfront effort to semantically integrate all source schemas and provide a mediated schema. Trails provide a declarative mechanism to enable semantic enrichment of a dataspace in a pay-as-you-go fashion.

2.3.3 Probabilistic Mapping

Instead of the synonym correspondence based on certain schema mapping, a probabilistic mapping describe a probability distribution of a set of possible schema mapping [21].

A probabilistic mapping is defined as $(S, T, \mathbf{m})$, where S and T are relations belonging to source schema and target schema respectively, and $\mathbf{m}$ is a set whose elements m_i consist of a one-to-one mapping between S and T and a probability, indicating the probability of each mapping.

For example, consider three possible mappings in Table 3. Each possible mapping consists of a set of attribute correspondences, and is associated with a corresponding probability. Note that the sum of all probability is 1.

It is worth noting that even though the studies [6, 7] focus now on attribute correspondences, probabilistic schema mappings could be much more complex semantic correspondences, e.g., more general GLAV mappings.

Table 3: Probabilistic schema mapping

	Possible Mapping	Prob
m_1	{(pname, name), (email-addr, email), (current-addr, mailing-addr), (permanent-addr, home-addr)}	0.5
m_2	{(pname, name), (email-addr, email), (permanent-addr, mailing-addr), (current-addr, home-addr)}	0.4
m_3	{(pname, name), (email-addr, mailing-addr), (current-addr, home-addr)}	0.1

3. SIMPLE SEARCH QUERY

The primary and easy way for most people accessing dataspaces is the keyword predicate query [5]. Each predicate is of the form (attribute : keyword), denoted by $(A : K)$, where A is an attribute name and K is a keyword in the value of attribute A. For example, a keyword predicate query could be:

$$\{(title : Birch), (author : Raghu)\}.$$

To answer the keyword predicate query, we consider both the query and objects as sets of *items* of

attribute keyword pairs. An object O (or a query Q) in dataspaces is thus a set of items $\{(A_1 : K_1), (A_2 : K_2), \ldots, (A_{|O|} : K_{|O|})\}$. For example, an object with attribute value $(manu : Apple\ Inc.)$ can be represented by a set of items $\{(manu : Apple), (manu : Inc.)\}$, if each word is considered as a keyword.

The keyword predicate query returns the objects in the dataspace that match most items. Query answers are ranked by the following matching score in descending order $score(Q, O) = |Q \cap O|$.

A similar Filter operator is also considered in [11]. It returns resources (objects) satisfying the given conditions, which can be specified as a set of (attribute : keyword) pairs as well.

3.1 Index

Inverted lists are utilized for indexing dataspaces [5]. The *inverted index*, also known as *inverted files* or *inverted lists*, consists of a vocabulary of items and a set of inverted lists. Each item e corresponds to an inverted list of object IDs, where each ID reports the occurrence of item e in that object.

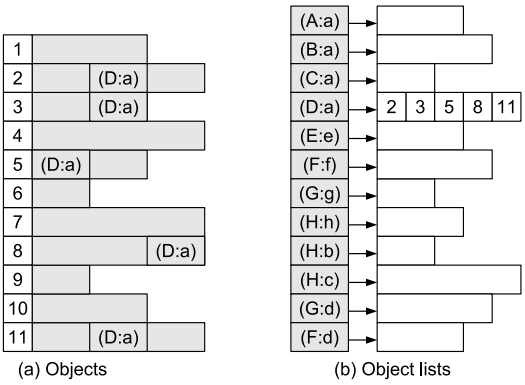

Figure 4: Indexing by attribute inverted lists (ATIL) for items of attribute-keyword pairs

The items of objects in dataspaces are indeed attribute-keyword pairs. The *attribute inverted lists* (ATIL) are lists of objects where the corresponding item (attribute-keyword pair) appears [5]. Figure 4 shows an example dataspace $\mathcal{S}$ which consists of 11 objects with a vocabulary $\mathcal{I}$ of 12 items. For each item (an attribute-keyword pair), we have a pointer referring to a specific list of object IDs, where the item appears. For instance, consider the inverted list of item (D : a) in Figure 4(b). It indicates that the keyword a on attribute D appears in the objects 2, 3, 5, 8, 11, as presented in in Figure 4(a). In real implementation, each object ID in the list is associated with a value, which denotes the weight of the item (D : a) in that object.

3.2 Compression

The heavy cost of inverted index based query processing arises from two aspects, 1) I/O cost of reading inverted lists from disks, and 2) aggregation of these inverted lists for ranking results. To reduce the cost of retrieving and merging lists, compression of inverted lists is studied [22]. The basic idea is to store the merged lists for items that appear together frequently in queries or dataspace objects.

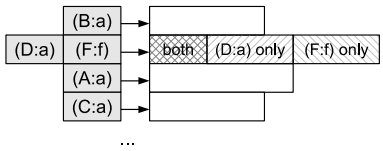

Figure 5: Compressing inverted lists on items (an item corresponds to only one list)

For instance, in Figure 5, the inverted lists of items, attribute-keyword pairs (D:a), (F:f), are combined together as a big single list. We have several sections in each compressed list, such as the sublist of tuples with both (D:a) and (F:f), the sublist with only (D:a), and the sublist with only (F:f).

3.3 Materialization

Materialized views in relational databases are often utilized to find equivalent view-based re-writings of relational queries [9], such as conjunctive queries or aggregate queries in databases. The concept of materialization is also extended to dataspaces for exploring query optimization opportunities [26].

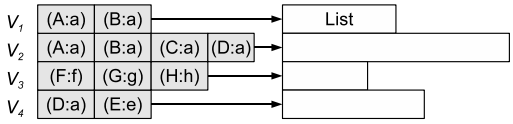

Figure 6: Materializing views of items (an item may be materialized multiple times)

In Figure 6, we show an example of materialized lists of item views. For instance, the first view, denoted by $V_1 = \{(\mathsf{A} : \mathsf{a}), (\mathsf{B} : \mathsf{a})\}$, materializes the merge results of lists corresponding to item (A : a) and (B : a) in the example of Figure 4. Without materialization, we need two random disk accesses for query predicates (A : a) and (B : a), respectively. In contrast, by materialized views, only one random disk access is needed for the same query. Moreover, besides reducing the I/O cost, we also avoid the aggregation of the aforesaid two separate inverted tuple lists in the query.

4. ASSOCIATION QUERY

Next, we discuss the queries specifying association requirements of objects. Again, both data models, triple store and resource view, support keyword predicate queries.

4.1 Association Search Query

4.1.1 Neighborhood Keyword Query

Neighborhood keyword query extends traditional keyword query by taking associations into account, that is, it also explores associations between data items. A neighborhood keyword query specifies a set of keywords, and the result consists of (1) relevant instance, which contains at least one of the query keywords; (2) associated instance, which is associated with a relevant instance.

For example, consider a query whose keyword is "*Birch*" in Figure 2. Instance a_1 is a relevant instance as it contains "*Birch*" in the title attribute, and p_1, p_2 and c_1 are associated instances as they have association with a_1.

4.1.2 Association Predicate Query

Besides the predicates on attribute values, predicates on associations can also be specified [5], of the form $(R : K)$, where R in the predicate is an association name. Objects satisfy the predicate if they have associations of type R with objects that contain the keyword K in attribute values.

For example, a query "Raghu's Birch paper in Sigmod" can be described with three predicates:

$$\{(title : Birch), (author : Raghu), \\ (publishedIn : Sigmod)\}.$$

The query is satisfied by object a1 in Figure 2, which has attribute value $(title : Birch)$, and in association with c1 (containing keyword $Sigmod$) in type $publishedIn$ and p2 (containing keyword $Raghu$) with type $author$.

4.1.3 Indexing Associations

Similar to indexing attribute keywords, *attribute-association inverted lists* (AAIL) are introduced for indexing association information (together with the aforesaid attribute-keyword information). Suppose that an object O has an association R with objects $O_1, \ldots, O_n$ in the dataspace, and each of $O_1, \ldots, O_n$ has the keyword K in one of its attribute values. An inverted list will be generated for the association:value item $(R : K)$, which contains n objects $O_1, \ldots, O_n$ in the list. For instance, in Figure 7, object $a1$ appears in the list of $(publishedIn : 1996)$, since $a1$ is in association (with type $publishedIn$) to another object ($c1$) containing keyword 1996 in its attribute ($year$), as illustrated in Figure 2.

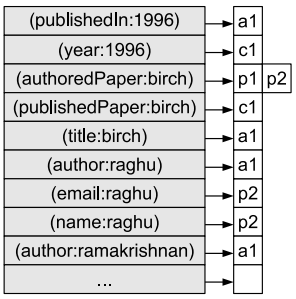

Figure 7: Attribute-association inverted lists (AAIL) for both associations and attribute-keyword items (for the example dataspace in Figure 2)

An association predicate query $\{(R : K1), \ldots, (R : K_n)\}$ can be answered over the AAIL. For example, when searching for "Raghu's papers", the query contains an association predicate $(author : Raghu)$. Based on the AAIL in Figure 2, it returns object $a1$.

4.2 Association Trail Query

Other than query over arbitrary individual associations, [19] considers queries with specific groups of associations, specified by association trails.

4.2.1 Neighborhood Query

Given a query Q (of attribute-keyword items) and an association trail Φ_i, the association trail query results are given by $(Q \cap Q_L^i) \rtimes_{\theta_i} Q_R^i$, where Q_L^i and Q_R^i are the queries on the left and right sides of trail Φ_i, respectively, and θ_i is the θ-predicate of Φ_i.

For example, consider the dataspace in Figure 3 and an association trail

$$sameUniversity : class = person \overset{\theta(l,r)}{\Longleftrightarrow} class = person, \\ \theta(l, r) : (l.university = r.university).$$

A query $Q = \{(name : Anna)\}$ with the association trail returns not only object 1 in Figure 3 (matching the query content), but also objects 2 and 3 who share the same university (identified by $\theta(l, r)$).

When considering a set of association trails $\Phi^* := \{\Phi_1, \ldots, \Phi_n\}$, the query results could be the union of the corresponding results given by Φ_i.

4.2.2 Indexing Association Trails

The most intuitive strategy for processing association trail queries is to explicitly materialize all intensional edges in the graph, through a join index [28]. For instance, Figure 8 lists all the objects 2 and 3, which have association to object 1 specified by association trail 1 (sameUniversity). Similarly, objects 1 and 3 have association to object 2 as well.

OidLeft	OidRight	trailList
1	2	{1}
	3	{1}
2	1	{1}
	3	{1, 2}
3	1	{1}
	2	{1, 2}

trailID	trail label
1	sameUniversity
2	graduatedSameYear

Figure 8: Association trail index

At query time, it looks up materialization to obtain the neighborhoods for each object returned by the original query Q. A query $Q = \{(name : Anna)\}$ thus returns object 1 (containing the specified keyword in Q) as well as objects 2 and 3 with the specified association 1 (sameUniversity) to object 1.

lookupEdges

OidLeft	OidRight	trailList
2	1	{1}
3	1	{1}
	2	{2}

trailID	trail label
1	sameUniversity
2	graduatedSameYear

normalEdges

OidLeft	OidRight	trailList
1	2	{1}
	3	{1}
2	3	{2}

Figure 9: Grouping-compressed index (GCI) for association trail

Association (join) relationships can be grouped together as cliques. The grouping-compressed index (GCI) [19] explicitly represents the edges from a given object (node) O_1 in the clique to all the other objects $\{O_2, \ldots, O_C\}$, and for each remaining node in the clique, represents special lookup edges $\langle O_j, O_1, lookup \rangle$ that state O_j connects to the same objects as O_1. Thus, it represents the information in the clique with C normal edges for O_1 plus $C-1$ edges for the lookup edges of all remaining nodes. In short, a reduction in storage space (and consequently join indexing time) from C^2 edges to $2C-1$ edges.

For example, consider the clique of three objects $\{1, 2, 3\}$ in Figure 3. It corresponds to three groups of edges $\{(1,2),(1,3)\}, \{(2,1),(2,3)\}, \{(3,1),(3,2)\}$, in Figure 8. In the grouping-compressed index as shown in Figure 9, only one group of (normal) edges $\{(1,2),(1,3)\}$ is preserved, while other two groups of edges are represented by lookupEdges $(2,1)$ and $(3,1)$. The lookupEdges denote that objects 2 and 3 share the same connections as object 1 w.r.t. association trail 1 (sameUniversity). The total number of edges (inverted lists) reduce.

4.3 Path Expression Query

Path expressions (as well as keywords and predicates) have been used in XML search engines referring to the NEXI language [27]. The queries have been adapted to dataspaces by iDM [4, 20] and [32]. A most typical path expression query, $//A//B$, returns resource views named B, from which there exists a path to another resource view named A.

The path expression can also be combined with attribute predicates, such as $//A//B[b = 42]$. It further requires $RV.tuple.b = 42$ for the results specified by $//A//B$. For example,

$$//Projects//Grant[created = \text{“2006”}]$$

returns resource views which are documents entitled *Grant* and *created* in 2006 in all the *Projects*. Consequently, the object *Grant.doc* in Figure 1 will be returned, which has a path from *Project* to *Grant*.

A Traverse operator, similar to the path expression query, is also introduced in [11], which is used to find resources referenced by a property. It accepts a sequence of resources with attribute predicates and a set of conditions. The returned resources satisfy the conditions and have association to the given resources successively, i.e., there is a path from the first resource to the result.

5. HETEROGENEITY QUERY

Dataspace queries could be extended to not only objects with associations, but also attributes with correspondence owing to information heterogeneity.

5.1 Attribute Hierarchy Query

Owing to information heterogeneity, an attribute (such as *name*) may correspond to multiple descendants in hierarchies, e.g., $firstName$, $lastName$, $nickName$, etc. A query $(name : Tian)$ may refer to $firstName$, $lastName$, or $nickName$.

To support such hierarchies in attribute heterogeneity, a natural idea is to index dataspaces with duplication. *Attribute inverted lists with duplication* (Dup-ATIL) [5] is constructed as follows. If the keyword K appears in the value of attribute A_0, and A is an ancestor of A_0 in the hierarchy, then there is a list for $(A : K)$. It records the number of occurrences of K in values of the attribute A and A's sub-attributes. Consequently, a predicate query with the Dup-ATIL is answered in the same way as we use the ATIL.

Indexing Attributes with Hierarchy Path

The size of Dup-ATIL could be very large if the attribute hierarchy contains long paths from the root

attribute to the leaf attributes and most values in the dataspace belong to leaf attributes.

A more concise way is to generalize the inverted lists without introducing duplicates [5]. *Attribute inverted lists with hierarchies* (Hier-ATIL) is constructed by extending the attribute inverted list as follows. Let $A_0, \ldots, A_n$ be attributes such that for each $i \in [0, n-1]$, attribute A_i is the super-attribute of A_{i+1}, and A_0 does not have super-attribute. It introduces a hierarchy path $A_0// \ldots //A_n$ for attribute A_n. For each keyword K in the value of attribute A_n, there is an inverted list generated for $(A_0// \ldots //A_n : K)$. Each object in the list denotes that the keyword K appears in the attribute A_n of the object.

The attribute-keyword predicate query $(A : K)$ can be answered by considering all the inverted lists whose hierarchy paths contain the attribute A.

5.2 Attribute Synonym Query

Besides attribute hierarchies, a more general form of information heterogeneity is the attribute synonyms, $A \leftrightarrow B$, in more arbitrary attribute pairs.

Query Rewrite with Canonical Name

To accommodate synonyms, a straightforward idea is to introduce a synonym table for attribute and association names [5]. If attribute A is referred to as $A_1, \ldots, A_n$ in different data sources, it chooses the canonical name of A as one of $A_1, \ldots, A_n$.

In the index, when a keyword K appears in a value of the A_i attribute, there is an inverted list for $(A : K)$. The object in the list for $(A : K)$ denotes the occurrence of K in its attribute $A_1, \ldots,$ or A_n.

To answer a predicate query with attribute predicate $(A_i, K), i \in [1, n]$, we transform it into a keyword search for $(K : A)$. For example, suppose that *author* is considered as a canonical name for *author* and *authorship*. The predicate $(authorship : Tian)$ will be transformed into $(author : Tian)$.

Query Rewrite with Predicate Expansion

The attribute synonym relationship is often incrementally determined, in a pay-as-you-go style. To avoid updating the existing index (w.r.t. synonym attributes), we may also consider to expand the query with synonym attributes [26], rather than replacing them with canonical names.

Consider a query $Q = \{(A_1 : K_1), \ldots, (A_{|Q|} : K_{|Q|})\}$ specifying predicates on a set of attribute-keyword pairs. The expanded query $\hat{Q}$ of Q is

$$\hat{Q} = \{(B_i : K_i) \mid B_i \leftrightarrow A_i, (A_i : K_i) \in Q\} \cup Q.$$

For example, we consider a query

$$Q = \{(\mathsf{manu} : \mathsf{Apple}), (\mathsf{post} : \mathsf{Infinite})\}.$$

The query evaluation searches not only in the manu and post attributes specified in the query, but also in the attributes prod and addr according to the attribute synonym correspondences manu $\leftrightarrow$ prod and addr $\leftrightarrow$ post, respectively, having

$$\hat{Q} = \{(\mathsf{manu} : \mathsf{Apple}), (\mathsf{prod} : \mathsf{Apple}), (\mathsf{post} : \mathsf{Infinite}), (\mathsf{addr} : \mathsf{Infinite})\}.$$

5.3 Trail Query

Rather than simple attribute pairs or hierarchies, trails specify more complicated relationships among attributes of query answers [20]. Query processing in the presence of trails first detects whether a trail should be applied to a given query Q. The matching of a unidirectional trail is performed on the left side of the trail. If the left side of a trail was matched by a query Q, the right side of that trail will be used to compute the transformation of Q. Finally, the original query Q is merged with the transformation as a new query. The new query extends the semantics of the original query based on the information provided by the trail definition.

Query Rewrite with Trails

Let Ψ_i^L and Ψ_i^R denote the left and right sides of a trail Ψ_i, respectively. The rewrite processing consists of three phases: Matching, Transformation, and Merging.

(1) Matching. A trail Ψ_i matches a query Q whenever its left side query, Φ_i^L, is contained in Q as a subset.

(2) Transformation. The query Q is transformed by substituting Ψ_i^L with Ψ_i^R, denoted by $Q_{\Psi_i}^T$.

(3) Merging. The transformed query $Q_{\Psi_i}^T$ is merged with the matched subexpression.

Consequently, the query is expanded not only on Ψ_i^L but also the related Ψ_i^R.

For example, consider a query

$$Q := //Projects//Grant[created = \text{“2006”}],$$

and a trail

$$\Psi_1 := // * .tuple.created \rightarrow // * .tuple.date.$$

The trail states that when querying the *created* attribute of an object (resource view), it should also consider the query on the *date* attribute. Ψ_1 matches Q as its left side query Ψ_1^L is contained in Q. After transformation, we have

$$Q_{\Psi_1}^T := // * [date = \text{“2006”}].$$

The final merged query $Q^*_{\{\Psi_1\}}$ is

$$Q^*_{\{\Psi_1\}} := //Projects//Grant[created{=}\text{“2006” OR } date{=}\text{“2006”}].$$

5.4 Query with Probabilistic Mapping

Instead of certain attribute synonym correspondence, the matching of attributes (especially the one generated by auto-matching tools) is often uncertain. Query answering is thus performed on such probabilistic mapping [6, 7, 21].

Consider the possible mappings in Table 3. By-table semantics [6, 7] could be considered for query answering based on p-mapping, i.e., one mapping for all tuples. A query Q with attribute-keyword predicate $\{(mailing\text{-}addr : Sunnyvale)\}$ will be expanded as

$$\hat{Q} = \{\{(current\text{-}addr : Sunnyvale)\}, \{(permanent\text{-}addr : Sunnyvale)\}, \{(email\text{-}addr : Sunnyvale)\}\}.$$

The probability of an object O matching the query is computed by aggregation w.r.t. possible mappings. Suppose that there is an object O with the following attribute-value pairs

$$O = \{(pname : Bob), (email\text{-}addr, bob), (current\text{-}addr : Sunnyvale), (permanent\text{-}addr : Sunnyvale)\}.$$

As shown, two predicates on attributes *current-addr* and *permanent-addr* match, w.r.t. mappings m_1 and m_2, respectively. Referring to the probabilities of m_1 and m_2 in Table 3, the probability of this object O matching the query Q is 0.9.

Besides the simple predicate queries, more complex SPJ queries could be answered based on probabilistic mapping [6, 7, 21]. As aforesaid, semantic mappings are often not well-established in dataspaces, we do not consider the complex SPJ queries in this survey.

In addition, queries could also be answered based on trails with probability variants, named probabilistic trails [20]. Specifically, a probability value $0 \leq p \leq 1$ is assigned to a trail. The probability reflects the likelihood that the results obtained by the trail are correct. Similarly, there is another variant named scored trails, which bind a score to a trail, reflecting the relevance of the trail.

6. SIMILARITY QUERY

Other than given a set of attribute-keyword predicates, a similarity query poses a query object with attribute-value pairs, and returns dataspace objects that are similar to the query object.

Similarity Query Answering

Given a query of attribute value pairs, $Q = \{(A_1 : W_1), \ldots, (A_{|Q|} : W_{|Q|})\}$, the similarity query returns a set of objects, which are similar to Q on each attribute w.r.t. attribute similarity function [25].

A *similarity function* $\theta(A_i, A_j) : [A_i \approx_{ii} A_i, A_i \approx_{ij} A_j, A_j \approx_{jj} A_j]$ specifies a constraint on similarity correspondence of two values from attribute A_i or A_j, according to their corresponding similarity operators $\approx_{ii}, \approx_{ij}$ or $\approx_{jj}$. Here, A_i, A_j are often synonym attributes, and the similarity function often comes together with the attribute matching on how their attribute values should be compared. For instance, a similarity function specified on two attributes (manu, prod) for the example dataspace in Figure 10 can be

$$\theta(\mathsf{manu}, \mathsf{prod}) : [\mathsf{manu} \approx_{\leq 5} \mathsf{manu}, \mathsf{manu} \approx_{\leq 5} \mathsf{prod}, \mathsf{prod} \approx_{\leq 5} \mathsf{prod}].$$

Two objects O_1, O_2 are said to be similar w.r.t. $\theta(A_i, A_j)$, denoted by $(O_1, O_2) \asymp \theta(A_i, A_j)$, if at least one of three similarity operators in $\theta(A_i, A_j)$ evaluates to true. For example, (t_1, t_2) are similar w.r.t. $\theta(\mathsf{manu}, \mathsf{prod})$, since the edit distance of $(t_1[\mathsf{manu}], t_2[\mathsf{prod}])$ is $4 \leq 5$, satisfying the similarity operator $\mathsf{manu} \approx_{\leq 5} \mathsf{prod}$.

t_1:{(name : iPod), (color : red), (manu : Apple Inc.), (addr : InfiniteLoop, CA), (tel : 567), (website : itunes.com)};
t_2:{(name : iPod), (color : cardinal), (prod : Apple), (post : InfiniteLoop, Cupert), (tel : 123), (website : apple.com)};
t_3:{(name : iPad), (color : white), (manu : Apple Inc.), (post : InfiniteLoop), (phn : 567), (website : apple.com)}.

Figure 10: Example dataspace with three objects

Consider a similarity query Q over dataspace $\mathcal{S}$ with a set Θ of similarity functions. It returns all the objects O in $\mathcal{S}$ with similar attribute values to Q, i.e., for each A_i specified in Q, having $(Q, O) \asymp \theta(A_i, B_i)$ for some $\theta(A_i, B_i) \in \Theta$. For example, let

$$\theta(\mathsf{addr}, \mathsf{post}) : [\mathsf{addr} \approx_{\leq 9} \mathsf{addr}, \mathsf{addr} \approx_{\leq 9} \mathsf{post}, \mathsf{post} \approx_{\leq 9} \mathsf{post}]$$

be another similarity function. Consider a query $Q = \{(\mathsf{manu} : \mathsf{Apple}), (\mathsf{post} : \mathsf{InfiniteLoop}, \mathsf{CA})\}$. It returns all the 3 objects in Figure 10 as similarity query answers.

Semantic Query Optimization

Integrity constraints (e.g., FDs) can be utilized to rewrite and optimize queries, which is known as the semantic query optimization [3, 13]. In [24, 25], data dependencies are extended for similarity query optimization in dataspaces.

Data dependencies in dataspaces are generally in the form of $\varphi : \theta(A_i, A_j) \rightarrow \theta(B_i, B_j)$ defined on similarity functions $\theta(A_i, A_j)$ and $\theta(B_i, B_j)$. It states that for any two objects O_1, O_2 that are similar w.r.t. $\theta(A_i, A_j)$, it always implies $(O_1, O_2) \asymp \theta(B_i, B_j)$ as well. For example,

$$\varphi_1 : \theta(\mathsf{manu}, \mathsf{prod}) \rightarrow \theta(\mathsf{addr}, \mathsf{post}).$$

states that if the manu or prod values of two objects are similar, then their corresponding addr or post values should also be similar.

For instance, consider again the query object with (post : InfiniteLoop, CA) and (manu : Apple). As mentioned, the similarity query searches not only in the manu, post attributes specified in the query, but also in the synonym attributes prod, addr according to the similarity functions $\theta(\mathsf{manu}, \mathsf{prod})$ and $\theta(\mathsf{addr}, \mathsf{post})$, respectively. Recall the semantics of the above dependency φ_1. If (manu, prod) of the query object and a data object are found to be similar, then the data object can be directly returned as answer without evaluation on post, addr since their corresponding (post, addr) values must be similar as well. The query efficiency is improved.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we review several cases of accessing dataspaces (listed in Table 4), depending on the relationship information obtained thus far. When a dataspace system is first launched, with relationships barely identified, simple keyword queries apply. With the pay-as-you-go identification of relationships among data in dataspaces, the query answers are enhanced, e.g., trail queries, or probabilistic query answering with the partial probabilistic mapping. The gradually identified attribute relationships further enable the similarity query.

A dataspace system offers a suite of interrelated services and guarantees, where techniques, such as keyword query and probabilistic query answering, could be applied and considerably generalized [8, 10]. While some of the challenges in accessing dataspaces have been (partially) addressed, such as studying a sequence of earlier queries (for query optimization and potentially better semantic integration), or ranking answers from multiple sources with various levels of semantic mappings, the others remain open, e.g., handling inconsistencies in dataspaces.

We list some advice of future directions.

(1) To address inconsistencies in dataspaces, the comparable dependencies [24, 25] could be applied. However, such a notation is defined at schema level (originally for query optimization). Following the same line of extending data dependencies with conditions in databases, known as conditional dependencies [2, 17], we may also study data dependencies declared with conditions (instances) in dataspaces. For example, we may consider a keyword dependency $\phi : ([A_i \leftrightarrow B_i] : K_i) \rightarrow ([A_j \leftrightarrow B_j] : K_j)$, by extending matching keys [23]. It states that if an object contains a keyword K_i in either attribute A_i or its synonym B_i, then it must also have a keyword K_j appearing in either attribute A_j or B_j.

(2) To answer queries over inconsistent dataspaces, consistent query answering [1] for dataspaces needs to be studied. Existing study [15] could handle uncertainty and inconsistency together, but do not consider the heterogeneous data in dataspaces. In essence, we need to manipulate simultaneously the uncertainty originated from both schema mapping and data inconsistency.

(3) Beyond relational, tree-structured (XML) or graph-structured (RDF), more data types are expected to be supported in dataspaces, e.g., sequential (event logs). While the integration of sequential event data [33, 34] and inconsistency detection [30, 29] have been investigated, searching and consistent query answering over such heterogeneous event sequences, especially ranking together with other data types, remain open.

Acknowledgement

This work is supported in part by the Tsinghua University Initiative Scientific Research Program; China NSFC under Grants 61572272 and 61202008.

8. REFERENCES

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*, pages 68–79, 1999.

[2] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 746–755, 2007.

[3] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query

Table 4: Summary of the surveyed works

Query Case	Method	Detail
Simple Search	index [5]	Extend inverted lists for indexing dataspaces. Cost of list merging could be heavy in query processing
	compression [22]	Compress inverted lists to reduce the cost of retrieving and merging list
	materialization [26]	Materialize inverted lists to reduce the I/O and merging cost. May introduce large additional space cost
Association	association predicate [5]	Consider queries specifying association requirements of objects by keywords or predicates.
	association trail [19]	Consider queries with specific groups of associations, specified by association trails.
	path expression [4]	Consider relationships expressed in paths
Heterogeneity	hierarchy [5]	Handle the situation that an attribute corresponds to multiple descendants in hierarchies
	synonym [5, 26]	Consider attribute synonyms
	trail [20]	Expand query by tail associations
	probabilistic [21]	Enhance semantic mappings by modeling uncertainty
Similarity	dependency [25]	Return similar objects

optimization. *ACM Trans. Database Syst.*, 15(2):162–207, 1990.

[4] J. Dittrich and M. A. V. Salles. idm: A unified and versatile data model for personal dataspace management. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 367–378, 2006.

[5] X. Dong and A. Y. Halevy. Indexing dataspaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 43–54, 2007.

[6] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 687–698, 2007.

[7] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. *VLDB J.*, 18(2):469–500, 2009.

[8] M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[9] G. Gou, M. Kormilitsin, and R. Chirkova. Query evaluation using overlapping views: completeness and efficiency. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 37–48, 2006.

[10] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 1–9, 2006.

[11] B. Howe, D. Maier, N. Rayner, and J. Rucker. Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 270–277, 2008.

[12] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 847–860, 2008.

[13] A. Y. Levy and Y. Sagiv. Semantic query optimization in datalog programs. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA*, pages

163–173, 1995.

[14] Y. Li and X. Meng. Supporting context-based query in personal dataspace. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 1437–1440, 2009.

[15] X. Lian, L. Chen, and S. Song. Consistent query answers in inconsistent probabilistic databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 303–314, 2010.

[16] J. Liu, X. Dong, and A. Y. Halevy. Answering structured queries on unstructured data. In *Ninth International Workshop on the Web and Databases, WebDB 2006, Chicago, Illinois, USA, June 30, 2006*, 2006.

[17] S. Ma, W. Fan, and L. Bravo. Extending inclusion dependencies with conditions. *Theor. Comput. Sci.*, 515:64–95, 2014.

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

[19] M. A. V. Salles, J. Dittrich, and L. Blunschi. Intensional associations in dataspaces. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 984–987, 2010.

[20] M. A. V. Salles, J. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi. itrails: Pay-as-you-go information integration in dataspaces. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 663–674, 2007.

[21] A. D. Sarma, X. L. Dong, and A. Y. Halevy. Uncertainty in data integration and dataspace support platforms. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, Data-Centric Systems and Applications, pages 75–108. Springer, 2011.

[22] S. Song and L. Chen. Indexing dataspaces with partitions. *World Wide Web*, 16(2):141–170, 2013.

[23] S. Song, L. Chen, and H. Cheng. On concise set of relative candidate keys. *PVLDB*, 7(12):1179–1190, 2014.

[24] S. Song, L. Chen, and P. S. Yu. On data dependencies in dataspaces. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 470–481, 2011.

[25] S. Song, L. Chen, and P. S. Yu. Comparable dependencies over heterogeneous data. *VLDB J.*, 22(2):253–274, 2013.

[26] S. Song, L. Chen, and M. Yuan. Materialization and decomposition of dataspaces for efficient search. *IEEE Trans. Knowl. Data Eng.*, 23(12):1872–1887, 2011.

[27] A. Trotman and B. Sigurbjörnsson. Narrowed extended xpath I (NEXI). In *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl Castle, Germany, December 6-8, 2004, Revised Selected Papers*, pages 16–40, 2004.

[28] P. Valduriez. Join indices. *ACM Trans. Database Syst.*, 12(2):218–246, 1987.

[29] J. Wang, S. Song, X. Lin, X. Zhu, and J. Pei. Cleaning structured event logs: A graph repair approach. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 30–41, 2015.

[30] J. Wang, S. Song, X. Zhu, and X. Lin. Efficient recovery of missing events. *PVLDB*, 6(10):841–852, 2013.

[31] W. Zheng, L. Zou, X. Lian, J. X. Yu, S. Song, and D. Zhao. How to build templates for RDF question/answering: An uncertain graph similarity join approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1809–1824, 2015.

[32] M. Zhong, M. Liu, and Y. He. 3sepias: A semi-structured search engine for personal information in dataspace system. *Inf. Sci.*, 218:31–50, 2013.

[33] X. Zhu, S. Song, X. Lian, J. Wang, and L. Zou. Matching heterogeneous event data. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1211–1222, 2014.

[34] X. Zhu, S. Song, J. Wang, P. S. Yu, and J. Sun. Matching heterogeneous events with patterns. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 376–387, 2014.