Computer Technology and Application 3 (2012) 418-424

Modelling Dataspace Entity Association Using Set Theorems

Shibwabo Bernard Kasamani[1], Ateya Ismail Lukandu[1] and Wanyembi Gregory[2]

1. Faculty of Information Technology, Strathmore University, Nairobi 00200, Kenya

2. Department of Computer Science, Masinde Muliro University of Science and Technology, Kakamega 50100, Kenya

Received: May 17, 2012 / Accepted: June 13, 2012 / Published: June 25, 2012.

Abstract: The development of dataspace support systems is far from reality as individuals and enterprises are faced with the huge challenge of data management. Critical to this is the need to provide a model that represents the relationships between the entities collaborating in a dataspace. A dataspace is a new abstraction and target architecture to data management that does not require up-front semantic data integration. This paper models a dataspace using the set theory with entity mappings. A technique for identity resolution and pay-as-you-go data integration is explained. In order to provide a strong degree of assurance, the authors subject the model to certain real world entities that might form part of a global dataspace.

Key words: Dataspaces, entity collaboration, integration, geo data, data management.

1. Introduction

The overall observation supports the fact that large volumes of data are continuously being stored in data repositories around the world [1]. As data is continuously stored in data stores around the world, the need for effective and efficient techniques of data management is growing. Data appear in myriad of forms some in structured sources, e.g., Database Management Systems (DBMS) and some not. There is the demand to provide coherence between these sources. These data sources are becoming a part of a dataspace. Such a new abstraction is described in Ref. [2] as a new abstraction to data integration.

Despite of traditional (enterprise) databases with a given schema the goal is to manage a rich collection of structured, semi-structured, and unstructured data, spread in more enterprise repositories and on the Web. To control such data space of course does not mean other data integration approach. Data in data space rather coexists; semantic integration is not a necessity here, in order to operate parts of the system. Fig. 1 adopted from Ref. [2] shows a categorization of current solution of data management in two dimensions. Administrative proximity indicates how close various data sources are in terms of administrative control. “Near” means that sources have the same or at least coordinated control. Semantic integration is a measure, how closely the schemas of different data sources match [3].

A complete dataspace should be a plug and play architecture that is customizable (can be modeled) to the domain of interest. The concept of a domain is often ignored thought very important both in efficiency and clarity.

A dataspace should contain all of the information relevant to a particular organization regardless of its format and location, and model a rich collection of relationships between data repositories. Hence, the authors model a dataspace as a set of participants and relationships [3].

Corresponding author: Shibwabo Bernard Kasamani, M.Sc., research fields: data integration, dataspaces, spatial infrastructure. E-mail: bshibwabo@strathmore.edu.

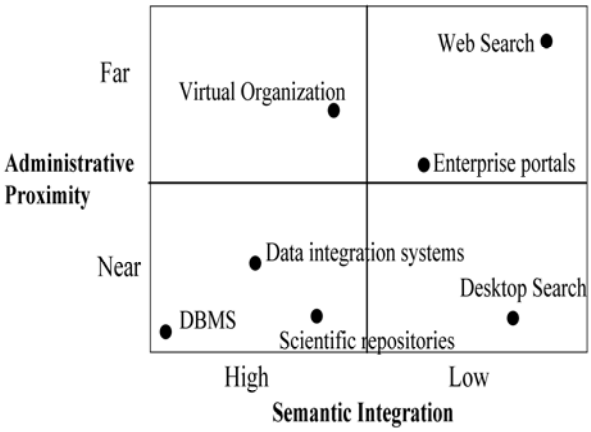

Fig. 1 A space of data management solutions.

The participants in a dataspace are the individual data sources: They can be relational databases, XML repositories, text databases, web services and software packages. They can be stored or streamed (managed locally by data stream systems), or even sensor deployments [3].

Some participants may support expressive query languages, while others are opaque and offer only limited interfaces for posing queries (e.g., structured files, web services, or other software packages). Participants vary from being very structured (e.g., relational databases) to semi-structured (XML, code collections) to completely unstructured. Some sources will support traditional updates, while others may be append-only (for archiving purposes), and still others may be immutable [4].

Two of the main services that a Dataspace Support Platform (DSSP) will support are search and query. While DBMSs have excelled at providing support for querying, search has emerged as a primary mechanism for end users to deal with large collections of unfamiliar data. Search has the property that it is more forgiving than query, being based on similarity and providing ranked results to end users, and supporting interactive refinement so that users can explore a data set and incrementally improve their results. A DSSP should enable a user to specify a search query and iteratively refine it, when appropriate, to a database-style query. A key tenet of the dataspaces approach is that search should be applicable to all of the contents of a dataspace, regardless of their formats [4].

Universal search and query should extend to meta-data as well as data. Users should be able to discover relevant data sources and inquire about their completeness, correctness and freshness. In fact, a DSSP should also be aware of gaps in its coverage of the domain [3]. The paper is organized as follows: Section 2 reviews related literature particularly sets and dataspaces; section 3 introduces dataspace entity relationships; section 4 presents results and discussions; section 5 gives conclusions.

2. Related Works

2.1 Binary Operations on Sets

A set with a binary operation is a fundamental concept in algebra. A binary operation on a set S is a rule that assigns to each ordered pair (a, b), where a and b are elements of S, exactly one element, denoted by ab, in S. A set S with a binary operation is said to be closed under the operation and in general if H is a subset of S then H is closed if ab is in H for all a and b in H. Quite often a binary operation on a set is referred to as multiplication [5].

An operation on S is commutative if $xy = yx$ for every x and y in S. An operation on S is associative if $x(yz)= (xy)z$ for every x, y and z in S. A semigroup is a set S with an associative operation. If a is an element of a semigroup and n is a natural number then an is defined to be the product $aaa...a$, of n factors. An element x of S is said to be an idempotent if $xx = x$, an element e of S is said to be an identity if $ex = x$ and $xe = x$ for all x in S, and an element z of S is said to be a zero if $zx = z$ and $xz = z$ for all x in S.

An isomorphism between S and S' is a one-to-one function ϕ mapping S onto S' such that $\phi(xy) = \phi(x)\phi(y)$ for all x and y in S. If there exists an isomorphism between S and S', then S and S' are said to be isomorphic, denoted $S \approx S'$ [5].

An anti-isomorphism between S and S' is a one-to-one function ϕ mapping S onto S' such that

$\phi(xy) = \phi(y)\phi(x)$ for all x and y in S. If there exists an anti-isomorphism between S and S', then S and S' are said to be anti-isomorphic, denoted $S \approx a\ S'$ [5].

2.2 Useful Theorems

The following theorems about sets, S and S', closed under operations are well known and easy to prove:

Theorem 1. If an identity exists in S then it is unique.

Theorem 2. If a zero exists in S then it is unique.

Theorem 3. In a set S of more than one element, an identity and a zero have to be distinct.

Theorem 4. If there exists an isomorphism between S and S' and the operation on S is associative then the operation on S' is also associative.

Theorem 5. If there exists an anti-isomorphism between S and S' and the operation on S is associative then the operation on S' is also associative.

Theorem 6. If S is a finite set, the operation on S is defined by an operation tables A and A^T (the transposition of A) defines an operation on the set $S' = S$, then there exists an anti-isomorphism between S and S'.

A summary of these theorems can be explained further using an Example. Consider a set $S = \{x, y\}$ with two elements. The number of different binary operations on this set is 16. The corresponding operation tables are presented in Table 1.

By studying the binary operation tables, the followings can be concluded:

- Eight operations are commutative. See Tables 1a-1b, 1g-1j and 1o-1p;
- Eight operations are associative. See Tables 1a-1b, 1d, 1f-1j and 1p;
- Six operations are both commutative and associative. See Tables 1a-1b, 1g-1h, 1j and 1p;
- For four operations there exists an identity in S. See Tables 1b, 1g-1h and 1j;
- For four operations there exists a zero in S. See Tables 1a-1b, 1h and 1p;
- For two operations there exist both an identity and a zero in S. See Tables 1b and 1h.

Table 1 Binary operations.

(a)

	x	**y**
x	x	x
y	x	x

(b)

	x	**y**
x	x	x
y	x	y

(c)

	x	**y**
x	x	x
y	y	x

(d)

	x	**y**
x	x	x
y	y	y

(e)

	x	**y**
x	x	y
y	x	x

(f)

	x	**y**
x	x	y
y	x	y

(g)

	x	**y**
x	x	y
y	y	x

(h)

	x	**y**
x	x	y
y	y	y

(i)

	x	**y**
x	y	x
y	x	x

(j)

	x	**y**
x	y	x
y	x	y

(k)

	x	**y**
x	y	x
y	y	x

(l)

	x	**y**
x	y	x
y	y	y

(m)

	x	**y**
x	y	y
y	x	x

(n)

	x	**y**
x	y	y
y	x	y

(o)

	x	**y**
x	y	y
y	y	x

(p)

	x	**y**
x	y	y
y	y	y

2.3 Dataspace Design

Ref. [3] models a dataspace as consisting of participants and relationships (Fig. 2). A dataspace should contain all of the information relevant to a particular organization regardless of its format and location, and model a rich collection of relationships between data repositories.

To summarize, the distinguishing properties of dataspace systems, Ref. [5] outlines the following:

- A DSSP must deal with data and applications in a wide variety of formats accessible through many systems with different interfaces. A DSSP is required to support all the data in the dataspace rather than leaving some out, as with DBMSs;
- Although a DSSP offers an integrated means of searching, querying, updating, and administering the dataspace, often the same data may also be accessible

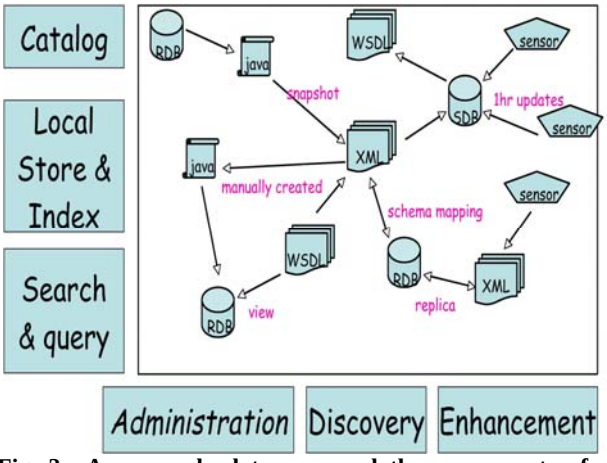

Fig. 2 An example dataspace and the components of a dataspace system.

and modifiable through an interface native to the system hosting the data. Thus, unlike a DBMS, a DSSP is not in full control of its data;

• Queries to a DSSP may offer varying levels of service, and in some cases may return best-effort or approximate answers. For example, when individual data sources are unavailable, a DSSP may be capable of producing the best results it can, using thedata accessible to it at the time of the query;

• A DSSP must offer the tools to create tighter integration of data in the space as necessary.

The next section presents an approach to building relationships between dataspace participants.

3. Dataspace Entity Relationships

3.1 The Hybrid Dataspace Model

Dataspace entities in principal can be modeled to take a hybrid relationship which combines both hierarchical and network model in order to be sufficient. An example is that a page belongs to a document but it is also true to state that a page belongs to a website. Moreover, a website may also consist of documents. This can be represented as shown in Fig. 3.

Fig. 3 also indicates that entities in a dataspace belong to a larger dataspace or properly expressed as subsets of the universal set denoted as U. Based on the observations presented by Fig. 3, a model outlined in Fig 4 can be derived.

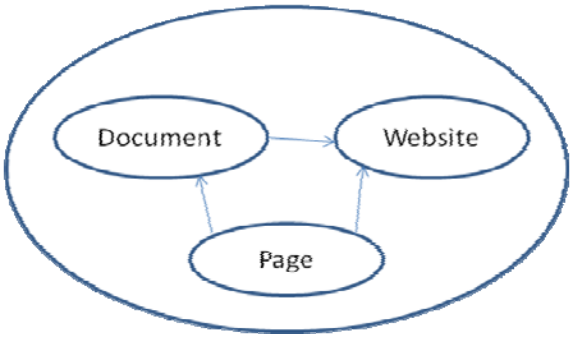

Fig. 3 Modelling a dataspace of three entities.

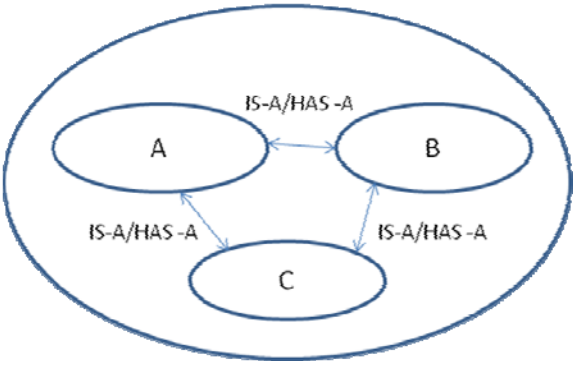

Fig. 4 The hybrid dataspace model.

A critical element to understand the hybrid model is the domain. A dataspace will typically represent a specific domain thereby providing a higher degree of sense in what is being represented. This does not mean that a global dataspace cannot be gotten. The second critical element to understand is that the authors' goal is to achieve co-existence rather than integration. Therefore, they might attempt to give an answer to the following question:

"How is A related to B?"

A dataspace can be represented as a Set and the participants in a dataspace as the Set Elements. This is computationally feasible due to the fact that there exists a practical programming implementation defined as set<Key, Compare, Alloc>. Where:

Key: The set's key type and value type. This is also defined as set::key_type and set::value_type;

Compare: The key comparison function, a Strict Weak Ordering whose argument type is key_type; it returns true if its first argument is less than its second argument, and false otherwise. This is also defined as set::key_compare and set::value_compare;

Alloc: The set's allocator, used for all internal memory management.

It is also possible for an element of a set to be another set. This is important in order to address the challenge of scalability of a dataspace. Set and multiset are particularly appropriate to the set algorithms include, set_union, set_intersection, set_difference, and set_symmetric_difference. The reason for this is twofold.

First, the set algorithms require their arguments to be sorted ranges, and, since set and multiset are Sorted Associative Containers, their elements are always sorted in ascending order. Second, the output range of these algorithms is always sorted, and inserting a sorted range into a set or multiset is a fast operation: The Unique Sorted Associative Container and Multiple Sorted Associative Container requirements guarantee that inserting a range takes only linear time if the range is already sorted [6].

Set has the important property that inserting a new element into a set does not invalidate iterators that point to existing elements. Erasing an element from a set also does not invalidate any iterators, except, of course, for iterators that actually point to the element that is being erased [6].

A C++ implementation of the dataspace follows:

```
struct ltstring
{
bool operator()(const char* sA, const char* sB) const
{
return strcmp(sA, sB) < 0;
}
};

int main()
{
const int N = 3;
const char* a[N] = {"page", "website", "document"};
const char* b[N] = {"honey", "page", "cup"};
set<const char*, ltstring> A(a, a + N);
set<const char*, ltstring> B(b, b + N);
set<const char*, ltstring> C;
cout << "The Set A: ";
copy(A.begin(), A.end(), ostream_iterator<const char*>(cout, " "));
cout << endl;
cout << "The Set B: ";
copy(B.begin(), B.end(), ostream_iterator<const char*>(cout, " "));

cout << endl;
cout << "The Union: ";
set_union(A.begin(), A.end(), B.begin(), B.end(), ostream_iterator<const char*>(cout, " "), ltstring());
cout << endl;
cout << "The Intersection: ";
set_intersection(A.begin(), A.end(), B.begin(), B.end(), ostream_iterator<const char*>(cout, " "), ltstring());
cout << endl;
set_difference(A.begin(), A.end(), B.begin(), B.end(), inserter(C, C.begin()), ltstring());
cout << "Set C (difference of A and B): ";
copy(C.begin(), C.end(), ostream_iterator<const char*>(cout, " "));
cout << endl;
}
```

The arrays that have been defined in the code can easily be replaced with persistent data storage catalog (see Fig. 2) considering the fact that using only arrays for a dataspace may be overkill. In a typical scenario, the participants in a dataspace are entities that are either known or unknown. The process of knowing is understood as learning which forms a subject of future discussion.

4. Results and Discussion

This paper proposes an approach for the implementation of a dataspace infrastructure to help reduce the time taken to integrate systems. Dataspaces is not a data integration technique but rather a data co-existence approach.

Dataspaces hopes to further reduce data integration time as it attempts to eliminate the need for upfront semantic integration which is associated with human involvement in data integration by having integration on demand.

Some of the current data integration techniques use semantic integration which has the major weakness that it requires full semantic integration of the sources in order to provide useful services [1]. The trends in data integration are clearly summarized by Ref. [7]. Other integration approaches are discussed in Ref. [1]. Also according to Ref. [1], most integration approaches lack the speed, flexibility and economy (integration on-demand) that many organizations need today in a data integration solution.

Dataspaces provide data integration on a pay-as-you-go fashion, this in itself makes data integration more practical as it is more economical and faster to integrate data on a need basis.

Many computer design or pure geometry models deal with continuous motion of sets. First of all, a "distance" between sets is needed. There are several natural candidates for the job, according to the desired properties of the motion model. The most general distance of this kind is the so-called "area-distance" defined in a space M with measure μ as follows: If A and B are measurable subsets of M, then set d(A, B) = μ(A ┌ B), where A ┌ B = (A\B) ∪ (B\A) is the symmetric difference of A and B. This is in fact the distance the authors deal with in the present article. Note that formally speaking, d is not a metric, but only a pseudometric in the family M of measurable subsets of M, nevertheless, it becomes metric after appropriate factorization of in M [8].

The implementation of the dataspace vision is still a task far from reality but there is ongoing work to actualize this infrastructure. It is important to mention that the current buzz words like cloud computing would require data integration as any other system. Cloud computing is about provision of services as opposed to data integration. Data integration is a need for cloud service providers.

A dataspace should provide for data coexistence in any data storage device irrespective of the underlying data model. The goal is to model the integration that exists in the human into the dataspace support system.

5. Conclusions

This paper has shown, through existing literature and further analysis, that data is critical to organizations and that the need for data integration is growing. This need has partly been necessitated by the existence of various data stores that use various underlying models. Lack of integration has negative consequences to the maximization of the value of data in time as organizations worldwide tend to be increasingly dependent on data [1].

Existing literature has shown that current data integration approaches are not sufficient to address the need for data integration. The future of data integration exists in a new abstraction to data management called dataspaces. Dataspaces have actually been described as the final frontier.

A dataspace can be modeled using the concepts borrowed from the mathematical set theorems due to the nature of entities globally where they assume a combination of hierarchical and networked relationship. This paper mainly focuses on relating entities that exist in a dataspace.

In conclusion, the proposed approach has been prepared to address the increasing demand for data integration due to the fact that data is stored in myriad of repositories with multiple underlying data models. It models a dataspace as a set and provides a programming implementation of the same.

References

[1] B. Shibwabo, I. Ateya, Respository integration: The disconnect and way forward through repository virtualization supporting business intelligence, International Journal of Current Research 3 (4) (2011) 015-020.

[2] M. Franklin, A. Halevy, D. Maier, From databases to dataspaces: A new abstraction for information

management, ACM SIGMOD Record 34 (4) (2005) 27-33.

[3] J. Pokorny, Databases in the 3rd Millennium: Trends and Research Directions, Journal of Systems Integration 1 (2010) 3-15.

[4] M. Franklin, A. Halevy, D. Maier, Principles of dataspace systems, in: Proc. of 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006), ACM Press, pp. 1-9.

[5] F. Diego, K. Jónsdóttir, D. Maier, Associative operations on a three-element set, The Montana Mathematics Enthusiast 5 (2&3) (2008) 257-268.

[6] Available online at: http://www.sgi.com/tech/stl/set.html.

[7] P. Ziegler, K. Dittrich, Data integration—problems, approaches, and perspectives, in: J. Krogstie, A.L. Opdahl, S. Brinkkemper (Eds.), Conceptual Modelling in Information Systems Engineering, Springer, Berlin Heidelberg, 2007, pp. 39-58.

[8] S. Stefanov, V. Dragieva, Evolution of sets systems and homotopy groups of spheres, in: Proceedings of the 41st Spring Conference of the Union of Bulgarian Mathematicians, 2012, pp. 202-206.