Dimensions of Dataspaces

Cornelia Hedeler, Khalid Belhajjame, Alvaro A.A. Fernandes,
Suzanne M. Embury, and Norman W. Paton

School of Computer Science, The University of Manchester
Oxford Road, Manchester M13 9PL, UK
chedeler,khalidb,alvaro,embury,norm@cs.manchester.ac.uk

Abstract. The vision of dataspaces has been articulated as providing various of the benefits of classical data integration, but with reduced up-front costs, combined with opportunities for incremental refinement, enabling a "pay as you go" approach. However, results that seek to realise the vision exhibit considerable variety in their contexts, priorities and techniques, to the extent that the definitional characteristics of dataspaces are not necessarily becoming clearer over time. With a view to clarifying the key concepts in the area, encouraging the use of consistent terminology, and enabling systematic comparison of proposals, this paper defines a collection of dimensions that capture both the components that a dataspace management system may contain and the lifecycle it may support, and uses these dimensions to characterise representative proposals.

1 Introduction

Data integration, in various guises, has been the focus of ongoing research in the database community for over 20 years. The objective of this activity has generally been to provide the illusion that a single database is being accessed, when in fact data may be stored in a range of different locations and managed using a diverse collection of technologies. Providing this illusion typically involves the development of a single central schema to which the schemas of individual resources are related using some form of mapping. Given a query over the central schema, the mappings, and information about the capabilities of the resources, a distributed query processor optimizes and evaluates the query.

Data integration software is impressive when it works; declarative access is provided over heterogeneous resources, in a setting where the infrastructure takes responsibility for efficient evaluation of potentially complex requests. However, in a world in which there are ever more networked data resources, data integration technologies from the database community are far from ubiquitous. This stems in significant measure from the fact that the development and maintenance of mappings between schemas has proved to be labour intensive. Furthermore, it is often difficult to get the mappings right, due to the frequent occurrence of exceptions and special cases as well as autonomous changes in the sources that require changes in the mappings. As a result, deployments are often most successful when integrating modest numbers of stable resources in carefully managed

environments. That is, classical data integration technology occupies a position at the high-cost, high-quality end of the data access spectrum, and is less effective for numerous or rapidly changing resources, or for on-the-fly data integration.

The vision of *dataspaces* [1, 2] is that various of the benefits provided by planned, resource-intensive data integration should be able to be realised at much lower cost, thereby supporting integration on demand but with lower quality of integration. As a result, dataspaces can be expected to make use of techniques that infer relationships between resources, that refine these relationships in the light of user or developer feedback, and that manage the fact that the relationships are intrinsically uncertain. However, to date, no dominant proposal or reference architecture has emerged. Indeed, the dataspace vision has given rise to a wide range of proposals either for specific dataspace components (e.g. [3, 4]), or for complete dataspace management systems (e.g. [5, 6]). These proposals often seem to have little in common, as technical contributions stem from very different underlying assumptions – for example, dataspace proposals may target collections of data resources as diverse as personal file collections, enterprise data resources or the web. It seems unlikely that similar design decisions will be reached by dataspace developers working in such diverse contexts. This means that understanding the relationships and potential synergies between different early results on dataspaces can be challenging; this paper provides a framework against which different proposals can be classified and compared, with a view to clarifying the key concepts in dataspace management systems (DSMS), enabling systematic comparison of results to date, and identifying significant gaps.

The remainder of the paper is structured as follows. Section 2 describes the classification framework. For the purpose of instantiating the framework Section 3 presents existing data integration proposals, and Section 4 presents existing dataspace proposals. Section 5 makes general observations, and Section 6 concludes the paper.

2 The Classification Framework

Low-cost, on-demand, automatic integration of data with the ability to search and query the integrated data can be of benefit in a variety of situations, be it the short-term integration of data from several rescue organisations to help manage a crisis, the medium-term integration of databases from two companies, one of which acquired the other until a new database containing all the data is in place, or the long-term integration of personal data that an individual collects over time, e.g., emails, papers, or music. Different application contexts result in different dataspace lifetimes, ranging from short-, medium- to long-term (*Lifetime* field in Table 1).

Figure 1 shows the conceptual life cycle of a dataspace consisting of phases that are introduced in the following. Dataspaces in different application contexts will only need a subset of the conceptual life cycle. The phases addressed are listed in *Life cycle* in Table 1 with the initialisation, test/evaluation, deployment,

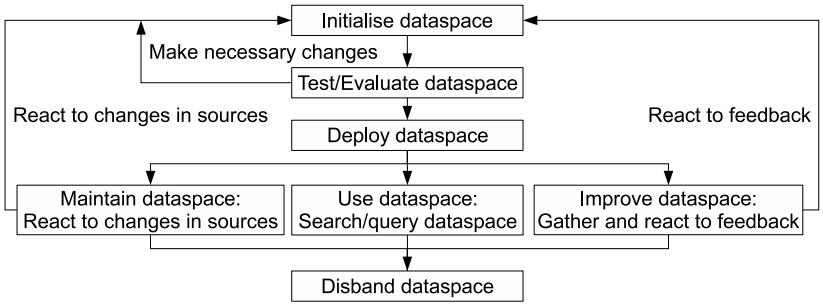

Fig. 1. Conceptual life cycle of a dataspace.

maintenance, use, and improvement phases denoted as *init*, *test*, *depl*, *maint*, *use*, and *impr*, respectively.

A dataspace, just like any traditional data integration software, is initialised, which may include the identification of the data resources to be accessed and the integration of those resources. Initialisation may be followed by an evaluation and testing phase, before deployment. The deployment phase, which may not be required, for example, in the case of a personal dataspace residing on a single desktop computer, could include enabling access to the dataspace for users or moving the dataspace infrastructure onto a server. As the initialisation of a DSMS should preferably require limited manual effort, the integration may be improved over time in a pay-as-you-go manner [6] while it is being used to search and query the integrated data resources. In ever-changing environments, a DSMS also needs to respond to changes, e.g., in the underlying data resources, which may require support for incremental integration. The phases *Use*, *Maintain* and *Improve* are depicted as coexisting, because carrying out maintenance and improvement off-line would not be desirable. For clarity, the figure does not show any information flow between the different phases, so the arrows denote transitions between phases.

In the remainder of this section, the initialisation, usage, maintenance and improvement phases are discussed in more detail with a view to eliciting the dimensions over which existing dataspace proposals have varied. The dimensions are partly based on the dataspace vision [1, 2] and partly on the characteristics of dataspace proposals.

2.1 Initialisation Phase

Figure 2 presents a more detailed overview of the steps that may be part of the initialisation phase. In the following, each of these steps is discussed in more detail and the dimensions that are used to classify the existing proposals are introduced. For each step, the dimensions are either concerned with the process (e.g., identifying matchings) and its input, or with the output of the process

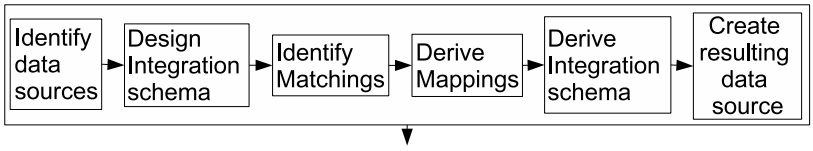

Fig. 2. Initialisation of a dataspace.

(e.g., the matchings identified). As others have proposed (e.g., [7]) we distinguish between *matchings*, which we take to be correspondences between elements and attributes in different schemas, and *mappings*, which we take to be executable programs (e.g., view definitions) for translating data between schemas.

Identify data sources. A DSMS can either provide support for the integration of data sources with any kind of content (*Cont* field in Table 1) or it can provide support for a specific application (*app_sp*), making use of assumptions that apply for that particular application. General support is denoted by *gen* in Table 1. Examples of specific applications include the life sciences, personal information and enterprise data. Furthermore, the data sources to be integrated can be of different types (*Type* field in Table 1). Examples include unstructured (*unstr*), semi-structured (with no explicit schema) (*s_str*) or structured (with explicit schema) (*str*). The data sources can also differ in their *location*: they can be local (*loc*) or distributed (*distr*).

Integration schema and its design/derivation. The *process* of obtaining the integration schema can either be manual (*man*), i.e., it is designed, or it can be derived semi-automatically (*s_aut*), e.g., requiring users to select between alternatives, or automatically (*aut*) without any manual intervention. A variety of information can be used as *Input* for designing or deriving the schema, which is depicted by the different locations of the *Design* and *Derive* steps in Figure 2. The schema can be designed using *schema*, or instance (*inst*) information from the sources. Matchings (*match*) or mappings (*map*) can also be used as input. Even when all the available information is taken into account to derive the integration schema automatically, there may still be some uncertainty as to whether the schema models the conceptual world appropriately. This degree of *uncertainty* associated with the schema can be represented as a *score*, which can be, e.g., probabilities or weights derived from inputs. The resulting schema can simply be a union schema, in which source-specific concepts are imported directly into the integration schema, or a schema that merges (e.g. [8]) the source schemas with the aim of capturing more specific the semantic aspects that relate them. The different *types* of resulting schemas are denoted as *union* and *merge* in Table 1, respectively. Integration schemas can also vary in their *scope*. To be able to model a wide variety of data from a variety of domains, generic models (*gen*), such as resource-object-value triples, can be used. In contrast to those, domain-specific models (*dom_sp*) are used in other proposals.

Matchings and their identification. Matchings can vary with respect to their *endpoints*: they can either be correspondences between the source schemas (*src-src*) or between source schemas and the integration schema (*src-int*). The *process* of identifying the matchings can either be manual (*man*), semi-automatic (*s_aut*) or automatic (*aut*). When matchings are identified automatically, some *uncertainty* is often associated with them. This can be represented by the use of *scores*. The identification process may require a variety of different *inputs*, e.g., the *schemas* to be matched, instances that conform to the schemas (*inst*), and training data (*train*), e.g., when machine learning techniques are applied.

Mappings and their identification. Unlike matchings, we take mappings as always been expressed between the sources schemas and the integration schema, not between source schemas. The *process* to derive the mappings can either be manual (*man*), semi-automatic (*s_aut*) or automatic (*aut*). Similar to matchings, the mappings can be associated with a degree of *uncertainty* about their validity, which can be represented by *scores*. The *inputs* to the derivation process may include the *schemas* to be mapped, instances that conform to the schemas (*inst*), matchings (*match*), and training data (*train*), for example, when machine learning techniques are used.

Resulting data resource. The resulting data resources over which queries are expressed can vary with respect to their materialisation (*Materialis.*): they can either be virtual (*virt*), partially materialised (*p_mat*) or fully materialised (*f_mat*). *Uncertainty* that is associated with the content can be denoted by *scores*. During the creation of the integrated database, duplicates (*dupl*) and conflicts (*confl*) can either be reconciled (*Reconciliation*) or be allowed to coexist.

2.2 Usage phase: search/query and their evaluation

Searches and queries can be specified (*Specification*) as a workload in advance (*in_adv*) of data integration taking place, or they can be specified after the integration, at runtime (*run*). Specifying queries in advance provides the potential for optimising the integration specifically for a particular workload. Different *types* of searches/queries can be supported by the dataspace: exploratory searches, e.g, browsing (*browse*), which are useful either if the user is unfamiliar with the integration schema, or if there is no integration schema. Other types include keyword search (*key*), select- (*S*), project- (*P*), join- (*J*), and aggregation (*aggr*) queries. An aim for a dataspace is to provide some kind of search at all times [1]. Query *evaluation* can either be complete (*compl*) or partial (*part*), e.g., using top-k evaluation approaches or approaches that are able to deal with the unavailability of data sources [1]. If multiple sources are queried, the results have to be combined (*Combine results*), which may be done by forming the *union* or merging (*merge*) the results, which may include the reconciliation of duplicates and/or conflicts. To indicate the *uncertainty* associated with the results of a query, they may be annotated with *scores* of some form, or they may be *ranked*.

Table 1. Properties of the initialisation and usage phase of existing data integration and dataspace proposals

Dimension	DB2 II[9]	Aladin [10]	SEMEX [11, 12]	iMeMex[5], iTrails[13]	PayGo[6]	UDI[3]	Roomba [4]	Quarry [14]
Life time/Life cycle								
Lifetime	long	long	long	long	long	long	long	long
Life cycle	init/use/ maint	init/use/ maint	init/use	init/use/ maint/impr	init/use/ maint/impr	init	impr	init/use
Data sources								
Cont	gen	app_sp	app_sp	app_sp	gen	gen	gen	gen
Type	s_str/str	s_str/str	unstr/ s_str/str	unstr/ s_str/str	str	str		s_str
Location	distr	distr	loc/ distr	distr	distr	loc		loc
Integration schema; design/derivation								
Type	union/merge	union	merge	union	union	merge	union	union
Scope	dom_sp	dom_sp	dom_sp	gen	dom_sp	dom_sp	gen	gen
Uncertainty						score		
Process	s_aut/man	s_aut	man	aut	aut	aut	aut	aut
Input						schema/ match	schema/ inst	schema/ inst
Matchings; identification								
Endpoints	src-int	src-src	src-int	src-src	src-src	src-src, src-int	src-src	
Uncertainty				score	score	score	score	
Process	man	aut	aut	s_aut	aut	aut	aut	
Input		schema/ inst	schema/ inst	schema/ inst	schema/ inst/train	schema	schema/ inst	
Mappings; identification								
Uncertainty				score		score		
Process	man		aut	man		aut		
Input			match	schema/ inst		schema/ match		
Resulting data resource; creation								
Materialis.	virt/p_mat	mat	mat	virt		virt	virt	f_mat
Uncertainty						score		
Reconcil.	NA	dupl	dupl			dupl	dupl	
Search/query; evaluation								
Specification	in_adv/run	run	run	run	run	run	in_adv	run
Type	SPJ/aggr	browse/ key/SPJ	browse/ key SP	browse/ key/SPJ	key	SP(J)	key/S	browse/ SP
Uncertainty		ranked			ranked	score		
Evaluation	compl	compl	compl	partial	compl	compl	partial	compl
Comb. res.					union	merge	merge	union

2.3 Maintenance and Improvement Phase

The maintenance phase deals with the fact that the underlying data sources are autonomous [1], and the improvement phase aims to provide tighter integration over time [1]. The steps in both phases are comparable to the steps involved in the initialisation phase, however, additional inputs may need to be considered. Examples include user feedback, as well as previous matchings and mappings that may need to be updated after changes in the underlying schemas.

Despite the general awareness that a DSMS needs to be able to cope with evolving data sources and needs to improve over time, only limited results have been reported to date, making it hard to consolidate the efforts into coherent dimensions. In the following we suggest a set of dimensions, that may be used to characterise future research efforts (see also Table 2).

Table 2. Properties of the maintenance and improvement phase of existing data integration and dataspace proposals

Dimension	DB2 II [9]	Aladin [10]	SEMEX [11, 12]	iMeMex[5], iTrails[13]	PayGo[6]	UDI[3]	Roomba [4]	Quarry [14]
Maintenance								
Changes	add/ src_inst		add/ src_inst	add				
Reuse			match/map/ int_sch		match/map/ int_sch			
Improvement								
Approach					alg_match/ exp_user			exp_user
Stage_feedb								match
Stage_impr								match

Maintenance: For effective maintenance, a DSMS needs to be able to cope with a number of different *changes*, including adding (*add*) and removing (*rem*) of resources. A DSMS also needs to be able to cope with changes in the underlying sources, e.g. changes to the instances (*src_inst*) or the schemas (*src_sch*), as well as changes to the integration schema (*int_sch*). Ideally, a DSMS should require little or no manual effort to respond to those changes. It may also be beneficial to *Reuse* the results of previous integration tasks, e.g., previous matchings (*match*), mappings (*map*), integration schemas (*int_sch*), or even user feedback (*feedb*) when responding to source changes.

Improvement: Improvement may be achieved in a number of ways (*Approach*), including the use of different or additional approaches to those used during initialisation for deriving matchings (*a_match*), mappings (*a_map*), or the integration schema (*a_int*). Furthermore, user feedback can be utilised, which could be implicit (*imp_user*) or explicit (*exp_user*). In cases where user feedback is considered, this could be requested about a number of different stages (*Stage_feedb*). This includes requesting feedback on the matchings (*match*), mappings (*map*), integration schema(s) (*int_sch*), reformulated queries (*ref_query*), query results (*res*) or the ranking of the results (*res_ran*). The feedback obtained may not only be used to revise the stage about which it was acquired, but it may also be propagated for improvement at other stages (Stage_impr). The values for this property are the same as for *Stage_feedb*.

3 Data integration proposals

For the purpose of comparison, this section uses the framework to characterise the data integration facilities of DB2 [9], as an example of a classical data integration approach; values of the dimensions are provided in Tables 1 and 2.

DB2 [9] follows a database federation approach. It provides uniform access to heterogeneous data sources through a relational database that acts as mediation middleware. The integration schema, which could be a *union* schema, or a *merged* schema defined by views which need to be written *manually*. Data sources are accessed by wrappers, some of which are provided by DB2 and some of which may have to be written by the user. A wrapper supports full SQL

and translates (sub)queries of relevance to a source so that they are understood by the external source. Due to the *virtual* nature of the resulting data resource, changes in the underlying data sources may be responded to with limited manual effort. In summary, DB2 relies on largely manual integration, but can provide tight semantic integration and powerful query facilities in return.

4 Dataspace proposals

This section describes a selection of dataspace proposals. An overview of their properties can be found in Tables 1 and 2.

ALADIN [10] supports semi-automatic data integration in the life sciences, with the aim of easing the addition of new data sources. To achieve this, ALADIN makes use of assumptions that apply to this domain, i.e., that each database tends to be centered around one primary concept with additional annotation of that concept, and that databases tend to be heavily cross-referenced using fairly stable identifiers. ALADIN uses a *union* integration schema, and predominantly *instance-based* domain-specific approaches, e.g., utilising cross-referencing to discover relationships between attributes in entities. The resulting links are comparable to *matchings*. *Duplicates* are discovered during *materialisation* of the data resource. Links and duplicate information are utilised for *exploratory* and *keyword* searches and may help life scientist to discover previously unknown relationships. To summarise, ALADIN provides fairly loose integration and mainly exploratory search facilities that are tailored to the life sciences domain.

SEMEX [11, 12] integrates personal information. A domain model, which essentially can be seen as a *merged* integration schema, is provided *manually* up-front, but may be extended manually if required. Data sources are accessed using wrappers, some provided, but some may have to be written manually. The schemas of the data sources are *matched* and *mapped automatically* to the domain model, using a bespoke mapping algorithm that utilises heuristics and *reuses* experience from previous matching/mapping tasks. As part of the *materialisation* of the resulting data resource, *duplicate* references are reconciled, making use of domain knowledge, e.g., exploiting knowledge of the components of email addresses. SEMEX provides support for *adding* new data sources and *changes* in the underlying data, e.g., people moving jobs and changing their email address or phone number, which require domain knowledge to be resolved, e.g., to realise that it is still the same person despite the change to the contact details. SEMEX, therefore, can be seen as a domain-specific dataspace proposal that relies on domain knowledge to match schemas to the given integration schema and reconcile references automatically.

iMeMeX [5] is a proposal for a dataspace that manages personal information; in essence, data from different sources such as email or documents are accessed from a graph data model over which path-based queries can be evaluated. iMeMeX provides low-cost data integration by initially providing a *union* integration schema over diverse data resources, and supports incremental refinement through the *manual* provision of path-based queries known as iTrails [13].

These trail definitions may be associated with a *score* that indicates the *uncertainty* of the author that the values returned by an iTrail is correct. As such, iMeMeX can be seen as a light weight dataspace proposal, in which uniform data representation allows queries over diverse resources, but without automation to support tasks such as the management of relationships between sources.

PayGo [6] aims to model web resources. The schemas of all sources are integrated to form a *union* schema. The source schemas are then matched *automatically* using a schema matching approach that utilises results from the matching of large numbers of schemas [15]. Given the similarity of the schemas determined by matching, the schemas are then clustered. *Keyword* searches are reformulated into structured queries, which are compared to the schema clusters to identify the relevant data sources. The sources are ranked based on the similarity of their schemas, and the results obtained from the sources are *ranked* accordingly. PayGo [6] advocates the *improvement* of the semantic integration over time by utilising techniques that automatically suggest relationships or incorporate user feedback; however, no details are provided as to how this is done. In summary, PayGo can be seen as a large-scale, multi-domain dataspace proposal that offers limited integration and provides keyword-based search facilities.

UDI [3, 16] is a dataspace proposal for integration of a large number of domain independent data sources automatically. In contrast to the proposals introduced so far, which either start with a manually defined integration schema or use the union of all source schemas as integration schema, UDI aims to derive a *merged* integration schema *automatically*, consolidating schema and instance references. As this is a hard task, various simplifying assumptions are made: the source schemas are limited to relational schemas with a single relation, and for the purpose of managing uncertainty, the sources are assumed to be independent. Source schemas are matched *automatically* using existing schema matching techniques [17]. Using the result of the matching and information on which attributes co-occur in the sources, attributes in the source schemas are clustered. Depending on the *scores* from the matching algorithms, matchings are deemed to be certain or uncertain. Using this information, multiple mediated schemas are constructed, which are later consolidated into a single *merged* integration schema that is presented to the user. Mappings between the source schemas and the mediated schemas are derived from the matchings and have *uncertainty* measures associated with them. Query results are *ranked* based on the scores associated with the mappings used. In essence, UDI can be seen as a proposal for automatic bootstrapping of a dataspace, which takes the uncertainty resulting from automation into account, but makes simplifying assumptions that may limit its applicability.

Even though the majority of proposals acknowledge the necessity to improve a dataspace over time, Roomba [4] is the first proposal that places a significant emphasis on the *improvement* phase. It aims to improve the degree of semantic integration by asking users for *feedback* on *matches* and *mappings* between schemas and instances. It addresses the problem of choosing which matches should be confirmed by the user, as it is impossible for a user to confirm all

uncertain matches. Matches are chosen based on their utility with respect to a *query* workload that is provided *in advance*. To demonstrate the applicability of the approach, a *generic* triple store has been used and *instance-based matching* using string similarity is applied to obtain the matches.

Quarry [14] also uses a *generic* triple store as its resulting data source, into which the data is *materialised*. Using a *union* schema, the data from the data sources coexists without any semantic integration in the form of matchings or mappings. So called signature tables, which contain the properties for each source, are introduced and it is suggested that signature tables with similar properties could be combined. Quarry provides an API for browsing the integrated data and for posing select and project queries.

5 Discussion

To make dataspaces valuable, steps that have been proven to be difficult and/or time consuming and that, to date, have predominantly been carried out manually or semi-automatically, need to be automated. Such steps include designing an integration schema, deriving the matchings between schemas, and deriving mappings between the source and integration schemas, which can then be used for query evaluation. Automating those steps results in uncertainty associated with the matchings, the mappings and the integration schema. As a result, techniques need to be developed that either reduce the uncertainty or take it into account, for example during query evaluation. Further uncertainty may be introduced during reformulation of keyword queries into structured queries, and vice versa, and during reference reconciliation either at the time the target data resource is materialised or when the results from subqueries issued to multiple sources are combined.

Since the dataspace vision has only recently emerged, and since it opens a broad research space covering multiple areas including, among others, data integration, information retrieval, and uncertainty management, existing proposals vary greatly over the dimensions presented in this paper. However, the following general observations can be made, with reference to Tables 1 and 2:

Union schemas are more common than merged schemas. In most proposals to date, the integration schema used by the user to formulate the query is obtained by unioning the source schemas. This choice is driven by the fact that the creation of a merged schema often raises conflicts, the resolution of which requires human intervention. Although there has been some work describing [18] and automating the collection [4] of uncertain information in merged schemas, early dataspace proposals provide limited support for managing the uncertainty associated with merged schemas, and in most cases steer clear of the issue altogether. Furthermore, classical work on data integration typically does not address uncertainty, for the simple reason that this has been dealt with manually at the bootstrapping phase.

Improvement as a means for dealing with or reducing the impact of uncertainty. Although incremental improvement was presented in the datas-

pace vision as the means for reducing uncertainty, only two of the proposals described in Section 4 consider this issue. This may be explained by the fact that early proposals for dataspace are initially addressing challenges at the bootstrapping phase, rather than looking for means that facilitate the improvement of an existing dataspace.

Dataspaces with short life times are missing. Several applications may generate the need for a dataspace with a short life time. A well known example is that of mashups. The improvement phase of these applications is relatively small compared with dataspace with longer lifetimes. As such, these applications may require dealing with uncertainty to a greater level up-front at bootstrapping time.

6 Conclusions

Dataspaces represent a vision for incremental refinement in data integration, in which the effort devoted to refining a dataspace can be balanced against the cost of obtaining higher quality integration. Comprehensive support for pay-as-you-go data integration might be expected to support different forms of refinement, where both the type and quantity of feedback sought are matched to the specific requirements of an application, user community or individual. Early proposals, however, provide rather limited exploration of the space of possibilities for incremental improvement. A common approach prioritises reduced start-up costs, typically by supporting a *union* integration schema; such an approach provides syntactic consistency, but the extent to which the resulting dataspace can be said to "integrate" the participating sources is strictly limited.

Although there is a considerable body of work outside dataspaces to support activities such as schema matching or merging, early dataspace proposals have made fairly limited use of such techniques. For example, although there has been work on the automated construction of a global model that accommodates uncertainty in matchings [3], this work makes strong simplifying assumptions in terms of the complexity of the models to be integrated. Furthermore, there are no comparable results on automated refinement. As such, although there is certainly a role for different flavours of DSMS, this survey suggests that there is considerable scope for further research into dataspaces, and that proposals to date fall short in different ways of fulfilling the potential of pay-as-you-go data integration.

References

1. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record **34** (2005) 27–33
2. Halevy, A., Franklin, M., Maier, D.: Principles of dataspace systems. In: PODS'06, ACM (2006) 1–9
3. Das Sarma, A., Dong, X., Halevy, A.: Bootstrapping pay-as-you-go data integration systems. In: SIGMOD'08, ACM (2008) 861–874

4. Jeffery, S.R., Franklin, M.J., Halevy, A.Y.: Pay-as-you-go user feedback for dataspace systems. In: SIGMOD'08, ACM (2008) 847–860
5. Dittrich, J.P., Salles, M.A.V.: idm: A unified and versatile data model for personal dataspace management. In: VLDB'06, ACM (2006) 367–378
6. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., Yu, C.: Web-scale data integration: You can afford to pay as you go. In: CIDR'07. (2007) 342–350
7. Miller, R.J., Hernández, M.A., Haas, L.M., Yan, L., Ho, C.T.H., Fagin, R., Popa, L.: The clio project: managing heterogeneity. SIGMOD Record **30** (2001) 78–83
8. Pottinger, R., Bernstein, P.A.: Schema merging and mapping creation for relational sources. In: EDBT'08. (2008) 73–84
9. Haas, L., Lin, E., Roth, M.: Data integration through database federation. IBM Systems Journal **41** (2002) 578–596
10. Leser, U., Naumann, F.: (almost) hands-off information integration for the life sciences. In: CIDR'05. (2005) 131–143
11. Dong, X., Halevy, A.Y.: A platform for personal information management and integration. In: CIDR'05. (2005) 119–130
12. Liu, J., Dong, X., Halevy, A.: Answering structured queries on unstructured data. In: WebDB'06. (2006) 25–30
13. Vaz Salles, M.A., Dittrich, J.P., Karakashian, S.K., Girard, O.R., Blunschi, L.: itrails: Pay-as-you-go information integration in dataspaces. In: VLDB'07, ACM (2007) 663–674
14. Howe, B., Maier, D., Rayner, N., Rucker, J.: Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In: ICDE Workshops, IEEE Computer Society (2008) 270–277
15. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.: Corpus-based shema matching. In: ICDE'05. (2005) 57–68
16. Dong, X., Halevy, A.Y., Yu, C.: Data integration with uncertainty. In: VLDB'07. (2007) 687–698
17. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal: Very Large Data Bases **10** (2001) 334–350
18. Magnani, M., Rizopoulos, N., McBrien, P., Montesi, D.: Schema integration based on uncertain semantic mappings. In: ER'05, Springer (2005) 31–46