Chapter 7: Dataspaces

Cornelia Hedeler[1], Khalid Belhajjame[1], Norman W. Paton[1], Alessandro Campi[2], Alvaro A.A. Fernandes[1], and Suzanne M. Embury[1]

[1] School of Computer Science, University of Manchester, UK
{chedeler, npaton, alvaro, embury}@cs.manchester.ac.uk
[2] Dipartimento di Elettronica e Informatzione, Politecnico di Milano, Italy
campi@elet.polimi.it

Abstract. The vision of dataspaces is to provide 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. As such, dataspaces join a long stream of research activities that aim to build tools that simplify integrated access to distributed data. To address dataspace challenges, many different techniques may need to be considered: data integration from multiple sources, machine learning approaches to resolving schema heterogeneity, integration of structured and unstructured data, management of uncertainty, and query processing and optimization. Results that seek to realize the different visions exhibit considerable variety in their contexts, priorities and techniques. This chapter presents a classification of the key concepts in the area, encouraging the use of consistent terminology, and enabling a systematic comparison of proposals. This chapter also seeks to identify common and complementary ideas in the dataspace and search computing literatures, in so doing identifying opportunities for both areas and open issues for further research.

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

S. Ceri and M. Brambilla (Eds.): Search Computing, LNCS 5950, pp. 114–134, 2010.

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* [16,18] 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. As such, a dataspace can be seen as a data integration system that exhibits the following distinguishing features: (i) low/no initialisation cost, (ii) support for incremental improvement, and (iii) management of uncertainty that is inherent to the automatic integration process, but could also be present in the integrated data itself.

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. [23,35]), or for complete dataspace management systems (e.g. [10,27]). 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. In the context of search computing, the chapter identifies issues that occur in dataspaces that are also relevant to search computing, such as uncertainty, and explores how dataspace concepts may be relevant to multi-domain search and vice versa.

The remainder of the chapter is structured as follows. Section 2 describes the classification framework by introducing the various dimensions that are used to characterise data integration and dataspace proposals. For the purpose of instantiating the framework Section 3 describes existing data integration and dataspace proposals in the context of the classification framework. Section 4 discusses open issues for dataspaces and search computing, and Section 5 concludes the chapter. This paper extends Hedeler *et al.* [19] by extending the dimensions used in the classification, increasing the number of proposals included in the classification,

and by including a discussion of the relationship between dataspaces and search computing.

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 Tables 1 and 2).

Figure 1 shows the conceptual life cycle of a dataspace consisting of phases that are introduced in the following. Dataspaces in different application contexts may only need a subset of the conceptual life cycle. The phases addressed are listed in *Life cycle* in Tables 1 and 2 with the initialisation, test/evaluation, deployment, 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 [27] 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,

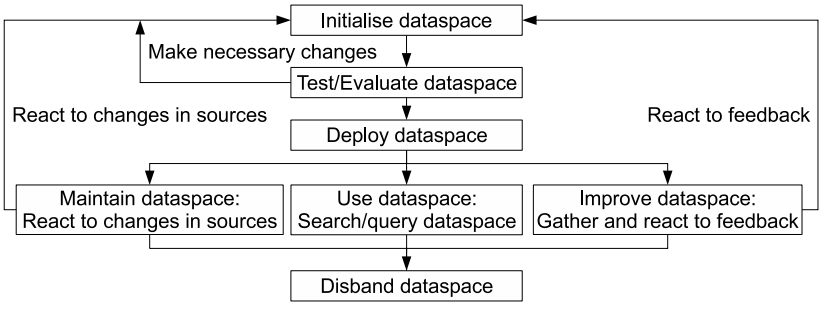

Fig. 1. Conceptual life cycle of a dataspace

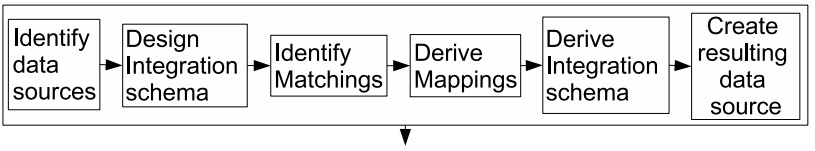

Fig. 2. Initialisation of a dataspace

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 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 [16,18] 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 (e.g., the matchings identified). As others have proposed (e.g., [30]) 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.

Data Sources. A DSMS can either provide support for the integration of data sources with any kind of content (*Cont* field in Tables 1 and 2) or it can provide support for a specific application (*app_sp*), in which case assumptions that apply to that particular application can be of benefit during the initialisation phase. Utilising domain knowledge, e.g., in form of a predefined integration schema, or utilising domain knowledge during matching and mapping generation, to create domain specific dataspace solutions could result in a higher quality of the initial integration and may require less improvement. However, those solutions could be hard to port into different domains. In contrast, general purpose dataspace solutions can be applied to any domain, but may be burdened by lower quality of the integration, requiring more user feedback for improvement of the integration. General support is denoted by *gen*. 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 Tables 1 and

2). 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 integration 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. [31]) the source schemas with the aim of capturing more specifically the semantic aspects that relate them. The different *types* of resulting schemas are denoted as *union* and *merge* in Tables 1 and 2, 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. Various data models can be used to represent the integration schema. Multiple models are used by the dataspace proposals discussed here, ranging from specific models, such as the relational model, to supermodels that subsume several specific models (e.g., [2]). 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.

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*). The identification process may require a variety of different *inputs*, e.g., the *schemas* to be matched, instances (*inst*) that conform to the schemas (which may be utilised instead of or in addition to schema information to infer matches between the schemas), and training data (*train*), e.g., when machine learning techniques are applied.

Mappings and Their Identification. Like matchings, mappings can also vary with respect to their *endpoints* (*src-src* or *src-int*). The *process* to derive the mappings can either be manual (*man*), semi-automatic (*s_aut*) or automatic (*aut*). The *inputs* to the derivation process may include the *schemas* to be mapped, instances that conform to the schemas (*inst*), matchings (*match*) and/or training data (*train*), for example, when machine learning techniques are used, or a *query*.

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

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

Dimension	DB2 II[17]	Aladin [24]	SEMEX [12,25]	iMeMex[10] iTrails[34]	PayGo[27]	UDI[35,13,14,36]	[23]	Quarry [20]
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
Initialisation								
Data sources; identification								
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
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	
Process	man	aut	aut	s_aut	aut	aut	aut	
Input		schema/ inst	schema/ inst	schema/ inst	schema/ inst/train	schema	schema/ inst	
Mappings; identification								
Endpoints	src-int	src-int	src-int	src-int/ src-src	src-int	src-int, src-int	src-int	src-int
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
Reconcil.	NA	dupl	dupl			dupl	dupl	
Usage: 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
Evaluation	compl	compl	compl	partial	compl	compl	partial	compl
Comb. res.				union	union	merge	merge	union
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

Table 2. Properties of the initialisation, usage, maintenance, and improvement phase of existing data integration and dataspace proposals (cont.)

Dimension	Q [37]		Cimple [11,29]	CopyCat [22]	Octopus [7]
Life time/Life cycle					
Lifetime	medium/ long		long	short	short
Life cycle	init/use/ impr		init/use/ maint/impr	init/use/ impr	init/ use/ impr
Initialisation					
Data sources; identification					
Cont	app_sp (gen)		app_sp	gen	gen
Type	s_str/str		str	s_str/str	s_str/str
Location	distr		distr	distr	distr
Integration schema; design/derivation					
Type	union		merge	union	merge
Scope	gen		dom_sp	dom_sp	dom_sp
Process	aut		man	s_aut	s_aut
Input	schema/ match			schema/ inst/ match	schema/ inst
Matchings; identification					
Endpoints	src-src		src-src/ src-int	src-src	src-int
Process	s_aut		s_aut	s_aut	s_aut
Input	schema/ inst		schema/ inst	schema/ inst	schema/ inst
Mappings; identification					
Endpoints	src-src		src-int	src-int	src-int
Process	aut		man	s_aut	s_aut
Input	schema/ match/ query			schema/ inst	schema/ inst
Resulting data resource; creation					
Materialis.	virt		p_mat	f_mat	f_mat
Reconcil.			dupl		dupl
Usage: Search/query; evaluation					
Specification	in_adv/ run		run	in_adv	in_adv
Type	key		key/ SPJ	VQL	key
Evaluation	compl		compl	compl	compl
Comb. res.	union		merge	union	merge
Maintenance					
Changes			src_sch/ src_inst	add	add
Reuse			int_sch	map	
Improvement					
Approach	exp_user	exp_user	exp_user	exp_user	exp_user
Stage_feedb	map	res/ res_ran	match	int_sch/ map	int_sch/ map
Stage_impr	map	map	match	int_sch/ map	int_sch/ map

(*f_mat*). During the creation of the integrated database, duplicates (*dupl*) entities and conflicts (*confl*) can either be reconciled (*Reconciliation*) using, e.g., record linkage approaches, 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*) or visual query languages (*VQL*), 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. A common aim for a dataspace is to provide some kind of search at all times [16]. 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 [16]. 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 entities and/or conflicts using, e.g., record linkage approaches.

2.3 Maintenance and Improvement Phase

The maintenance phase deals with the fact that the underlying data sources are autonomous [16], and the improvement phase aims to provide tighter integration over time [16]. 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 Tables 1 and 2).

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*) (e.g., [3]) 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*.

2.4 Uncertainty

For the purpose of this survey, we use the term uncertainty to cover various aspects, such as the trustworthiness of sources, or the robustness of algorithms which, e.g., could be represented as probabilities or scores associated with the resulting matchings or mappings. When data from a variety of sources is integrated, uncertainty may be introduced at various stages of the initialisation and maintenance phases, and may have an impact on the usage and improvement phases. As uncertainty plays a role across all phases of the dataspace life cycle, it is discussed separately here. Table 3 classifies proposals that handle uncertainty explicitely in terms of the dimensions.

When the uncertainty that is intrinsic to the integration process is made explicit, all the *concepts* that are produced during initialisation can be annotated with uncertainty information. The *concepts* include: *source data*, which in itself could be of uncertain quality; *data sources*, which for example could be ranked with respect to their relevance to a given query; the *matchings* identified, which may be computed using algorithms that use partial information; the *mappings*, which may be derived from uncertain matchings or, similar to matchings, they may be derived using incomplete information; the *integration schema*, which may be one of many alternative integration schemas that can be derived from mappings and as such may not model the conceptual world appropriately; and the *resulting data resource*, which may have uncertainty associated with its integrated content due to the uncertainty associated with the integration process itself. The uncertainty that may be accumulated throughout the various stages of the intialisation phase may then manifest itself in the usage phase. As such, *query results* or their *rankings* may be annotated with uncertainty information or quality measures. In addition, certain properties of the *query* itself may be uncertain, e.g., it may be uncertain whether a structured query that is derived from a keyword query [37] is an appropriate representation of the query the user had in mind.

Uncertainty can be represented by various *kinds of annotation*, which include: *scores*, which, for example, can be used to represent a preference; *prob*abilities, which can be used to express the probability that a concept is relevant; *precision/recall* measures; or by *ranking* values without providing addition quality measures. However, the quality measures associated with a concept could have various meanings, for example, the ranking of query results could mean that the results ranked higher are more relevant to the query or that they are more likely to be part of the correct answer as the matchings, mappings, etc. that have been utilised to obtain the answers have less uncertainty associated with them than the mappings used to obtain the lower ranked results. As such, we also clarify for each proposal that represents uncertainty explicitly the *meaning* of the quality measures associated with the concepts.

The annotation representing uncertainty can be *propagated* through various of the *operations* of the initialisation phase, such as, *identify matchings*, *derive mappings*, *derive integration schema*, and *create resulting data resource*. The operations utilised during the usage phase, such as *answer query* and *combine*

Table 3. Handling of uncertainty by existing data integration and dataspace proposals

Concept	Proposal	Kind of annotation	Meaning	Propagation	Propag. function
Data sources	PayGo[27]	ranking	relevance to query	combine results	in-built
	Octopus [7]	score/ ranking	relevance to query		
Matchings	iMeMex[10,4], iTrails[34]	prob	likelihood that results obtained are correct	derive mappings/ derive integration schema	in-built
		weights	relevance of matching to query	derive mappings/ derive integration schema	in-built
	PayGo[27]	weights	distance between schemas	derive mappings/ derive integration schema	in-built
	UDI[35,13,14,36]	prob	probability that matching is correct	derive mappings/ derive integration schema/ answer query	in-built
	Roomba [23]	score	confidence of match being correct		
	Q [37]	costs	bias against using matching for query as it produces worse answers from the user's point of view when used to answer a query	derive mappings/ answer query	in-built
	Cimple [11,29]	score	confidence that match is correct	answer query	in-built
	CopyCat [22]	score	relevance to integration operation	derive mappings/ answer query	in-built
Mapping	Octopus [7]	score	relevance to query and table to be joined with		
Query	Roomba [23]	score	expected result quality		
Query results	UDI[35,13,14,36]	score	scores of mappings used		
	Q [37]	score	cost of the query that produced result		
	Cimple [11,29]	score	scores from matchings used		
	CopyCat [22]	score	score of query that produced result		
User	Cimple [11,29]	score	trustworthiness of user		

results, can also propagate uncertainty. The *propagation function* that determines how the uncertainty is propagated can either be a predetermined *built-in* function, such as the sum or product of all the scores associated with the input concept, or can be *user-defined*, e.g., allowing the user to assign different importance in the form of weights to certain inputs. For example, users may choose to trust information coming from particular sources more than from others, and may want to encode their preference in the propagation function.

2.5 Human-Computer Interface

Another aspect that plays a role across the various phases of the dataspaces life cycle is the Human-Computer Interface that is provided to enable the user to interact with the system (see Table 4 for properties of proposals that provide a description of their user interface, and Table 5 for properties of proposals that describe the query inputs and outputs). As users have varying backgrounds,

Table 4. Human-Computer Interfaces provided by existing data integration and dataspace proposals

Concept	Proposal	Kind of user	Input	Optional/ Mandatory
Matchings	Roomba [23]	domain expert	annotate	optional
	Cimple [11,29]	domain expert	provide/ edit/ annotate	mandatory/ optional
Mappings	DB2 II [17]	database expert	provide	mandatory
	iMeMex[10,4], iTrails[34]	database expert	provide	mandatory
	Q [37]	domain expert	edit	optional
	CopyCat [22]	domain expert	provide/ edit/ annotate	mandatory
	Octopus [7]	domain expert	edit/ annotate	mandatory
Integration schema	SEMEX [12,25]	domain expert	edit	optional
	Cimple [11,29]	domain expert	provide	mandatory
	CopyCat [22]	domain expert	provide/ edit/ annotate	mandatory
	Octopus [7]	domain expert	edit/ annotate	mandatory
Ranked query results	Q [37]	domain expert	annotate	optional

Table 5. Query Interfaces provided by existing data integration and dataspace proposals

Proposal	Query input	Query output
DB2 II [17]	structured	results
Aladin [24]	keywords/ structured	ranked results
SEMEX [12,25]	keywords/ structured	results/ browse
iMeMex[10,4], iTrails[34]	keywords/ structured	results/ browse/ provenance
PayGo[27]	keywords	ranked results
UDI[35,13,14,36]	structured	ranked results
Roomba [23]	keywords/ structured	results
Quarry [20]	structured	results/ browse
Q [37]	keyword	ranked results
Cimple [11,29]	keyword/ structured	ranked results/ browse
CopyCat [22]	visual query language	results/ provenance
Octopus [7]	keyword	results

knowledge and experience, interfaces should be designed for different *kinds of users*. For the purpose of the classification framework, we only focus on *domain experts* who are familiar with the domain which is described by the information to be integrated and queried and *database experts* with a good understanding, for example, of the source schemas, the integration schema and how they relate to each other. Throughout the initialisation and improvement phase, users may want to or may be encouraged to provide information at various stages. The *input* provided by users may include *providing* the concepts in question as input, *editing* those suggested by the integration system, or *annotating* them with quality measures, for example, by indicating which were expected by the user (true positives), or which were not expected (false positives). *Concepts* that users may provide as input, edit or annotate include *matchings*, *mappings*, the *integration schema*, *query results* and *ranked query results*. Providing the information or annotation can either be *mandatory* or *optional*.

To cater for different kinds of users but also different degrees of integration, different interfaces for querying the integrated sources as well as viewing and possibly exploring the results may have to be provided. The *query input* can be provided using a *visual query language*, *keywords* or *structured queries*, such as

the select, project and join queries mentioned earlier in the usage phase. The *query output* could consist simply of the *results* or the *ranked results*, or could in addition include some *provenance* information that can be explored by the user to identify the source of the information returned. In addition, a means may be provided to *browse* the results and their associations with other information, for example by providing links that users can follow.

3 Data Integration Proposals

For the purpose of comparison, this section uses the framework to characterise and describe a number of dataspaces proposals, and in addition the data integration facilities of DB2 [17] as an example of a classical data integration approach. The proposals were classified according to the dimensions in Section 2. Only published proposals were chosen for which sufficient implementation detail is available to enable them to be classified according to the framework presented in Section 2.

DB2 [17] 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 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.

ALADIN [24] 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 scientists 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 [12,25] integrates personal information. A domain model, which essentially can be seen as a *merged* integration schema, is provided *manually* upfront, 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 [10,4] 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 [34]. 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 [27] 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 [26]. 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 [27] 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 [35,13,14,36] 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 [32]. 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 [23] 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 [20] 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.

Q [37], the query system of ORCHESTRA [21], a collaborative data sharing system, covering the three phases intialisation, usage and improvement, uses a *generic* graph structure to store the schemas and *matches* between schema elements, which are derived *semi-automatically* and *annotated* with costs representing the bias of the system against using the matches. *Mappings* in the form of query templates are derived from keyword queries posed by the user and matched against the schemas and matches. Multiple mappings are ranked by the sum of the cost associated with the matches utilised for the mapping, and may be edited and made persistent by the user for further reuse and parameterisation by him and other users. Tbe parameterised queries are executed and the results ranked by the cost associated with the query that produced them and annotated with *provenance* information that enables the propagation of user feedback from the ranked query results to the corresponding mapping. Users may provide *feedback* on the results and their ranking, indicating which results should be removed, or how results should be ranked, which is propagated to the ranking of the producing mappings and the costs of the matchings utilised. The feedback is used to

adjust the costs of the matchings and thus the ranking of the mappings used to answer the query in an attempt to try and learn the model the user has in mind.

Community Information Management systems, such as Cimple [11,29] aim to reduce the up-front cost of data integration by leveraging user feedback from the community. An integration schema is provided *manually*, sources *matched* in a *semi-automatic* manner in which an automatic tool is used as a starting point and users are asked to answer questions, thus confirming or rejecting matches suggested by the automatic tool. The *uncertainty* associated with the matches is propagated through to the query results, which are *annotated* with scores and *ranked*. As Cimple applies a mass collaboration approach and aims to reduce the uncertainty by gathering feedback from users, it is aware of trustworthy and untrustworthy users providing feedback, something not taken into account by other proposals that gather user feedback. It handles feedback by the different classes of users by ignoring feedback from untrustworthy users and taking a majority vote on the feedback from trustworthy users to identify correct matches.

CopyCat [22] follows a more interactive approach to data integration, combining the integration-, usage- and improvement phases by providing a spreadsheet-like workspace in which users copy and paste examples of the data they would like to integrate to answer the queries they have. The user copies data instances from various sources into the spreadsheet, thus specifying the *integration schema* and *mappings* initially manually. The system then tries to learn the schemas of the sources and the semantic types of the data from those examples and uses the learned information to identify *matches* between sources and to suggest *mappings* that reproduce the example tuples provided by the user or that integrate further data, thus making it a *semi-automatic* integration process. Users can provide *feedback* on those suggestions by either ignoring, accepting, *editing* , or *providing* alternatives. To ease the decision process for the user, *provenance* information is provided with the suggested data to be integrated. The user feedback is propagated back through the mappings to the matchings and their scores adjusted accordingly to reflect the user preference which in turn affects the scores and, therefore, the rankings of the mappings.

Similar to CopyCat, Octopus [7] provides the means for integrating multiple sources on the web interactively by providing several operations that can be utilised to create an integrated data source. Using the Search operator, the user states a *keyword query*, for which the system tries to find sources which are *ranked* according to their relevance with respect to the query. If multiple data sources are required to gather the required information, users can use the Extend operator, providing a column of a table with which to join the new table and a keyword stating the information desired. With that information the system tries to find appropriate source tables which are *ranked* according to their relevance with respect to the query and their compatibility with the column provided as input. Throughout the whole integration process, users can provide *feedback* by *editing* or *annotating* in form or rejecting or accepting the suggested source tables.

Both, CopyCat and Octopus do not distinguish between the various phases of the dataspace lifecycle, e.g., initialisation, usage, and improvement. Instead, they promote a seamless combination of initialisation, usage and improvement of the dataspace, albeit with a fair amount of user input required.

4 The Interplay between Dataspaces and Search Tasks

In this section, we investigate the interplay between dataspaces and search tasks. In particular, we show how features that are peculiar to search tasks can be borrowed and adapted in a dataspace context and vice-versa, and pinpoint open issues that arise as a result.

4.1 Performing Search Tasks in Dataspaces

One of the defining features of search tasks is that the sources return streams of ranked results. We refer to sources of this kind using the term *search sources*. Although one could imagine dataspaces queries that involve search sources, the classification and survey presented earlier in this chapter show that existing dataspace proposals do not support them. This raises the question as to how a dataspace system can be adapted to support queries over search sources. In what follows, we discuss issues that have to do with the initialisation and improvement phases of dataspaces when user queries are answered using search sources.

Usage. To support search sources, the query processor of the dataspace system needs to be able to produce results by combining streams of sub-results produced by multiple search sources. Furthermore, the results obtained need to be ranked in the light of the rankings of the sub-results produced by the search sources. In this respect, techniques from the search computing field can be borrowed and adapted for combining and ranking dataspace query results.

Improvement. To perform a search task, the dataspace system needs mappings from the integration schema, which is used by the user to pose queries, to the schemas of search sources. In doing so, the system needs to identify the search sources of relevance to users' queries. The identification of search sources can be performed in incremental manner by seeking feedback from users, e.g., the user can specify whether a result that is obtained from given search sources meets the expectations.

4.2 Using Dataspaces for Mult-domain Search Tasks

Multi-domain search tasks involve retrieving and combining the results obtained from multiple search sources. In what follows, we discuss issues that arise in the context of multi-domain searches, and shows how they can be addressed by adopting the pay-as-you-go philosophy adopted in dataspaces.

Query Expression. In a dataspace, the schemas of local data sources are initially integrated using low cost techniques, in particular, schema matching and schema merging algorithms are used for mappings the sources schemas and creating the integration schema (see the *integration schema* dimensions). The system is then improved in the light of feedback provided by the user in an incremental manner. One could envisage adopting a similar approach for easing the specification of multi-domain searches. In particular, the specification of the connections between the search services involved can be automatically derived using, e.g., matching techniques. Because those connections are derived based on heuristics, they may not meet the designer's expectations, which gives rise to the following research issue: *How can the connections suggested by inference tools to link search sources be verified?*

Another issue that arises in search tasks is the specification of queries that capture user's expectations. In dataspaces, the user can pose a structured query, e.g., using SQL, or specify a collection of keywords from which the dataspace system attempts to construct/learn a structured query using as input the source schemas and the mappings that connect the elements of these schemas [28,38] (see the *query type* dimension). *Can a similar approach be adopted for specifying queries in the context of search tasks?* One could envisage the case in which the user specifies a form that captures the elements of the search results the user is after. Using such a form, the system then attempts to construct a query by identifying the sources that provide the elements specified by the user, and connects the schemas of the sources selected using previously specified schema mappings. Of relevance to this problem is the proposal by Blunschi *et al.* [4], which considers indexing support for queries that combine keywords and structure and proposes several extensions to inverted lists to capture structure when it is present. In particular, it takes into account attribute labels, relationships between data items, hierarchies of schema elements, and synonyms among schema elements. We can also foresee the application of techniques taken from different areas in which the problem of search in semistructured or non structured data was already addressed [8,1,6,15]. In general, multiple structured queries are constructed from a set of keywords. The issue that needs to be addressed is, therefore, to identify the queries that closely meet user expectations.

Usage and Improvement. The results returned by each of the sources involved in a multi-domain search task are uncertain; this uncertainty is partly due to the fact that such results are generally obtained by matching a request with the content of the source in question using heuristics. The difficulty then lies in specifying a function whereby the results obtained by combining the sub-results retrieved from the sources involved in the search task can be ranked; this is a *ranking composition* problem [5]. Currently, ranking composition functions are typically manually specified, a task that can be difficult since it involves defining the global ranking of the query results taking into consideration the (possibly different) ranking criteria adopted by the underlying search sources.

A possible solution to the above issue can be borrowed from dataspaces through pay-as-you-go development of ranking composition. The results returned

by evaluating a user query in dataspaces are also uncertain; this uncertainty is partly due to the fact that the mappings used for populating the elements of the integration schema (against which the user queries are posed) are derived based on the results of matching heuristics [33] (see *mappings identification* dimensions). These mappings may not be manually debugged, but rather may be verified by seeking feedback from end users (see *improvement* dimensions). A similar approach can be adopted for specifying ranking composition in the context of multi-domain search tasks. For example, the user can specify that a given result should appear before another one. Using this kind of feedback, the system can then learn the ranking desired by the user. Which mechanism to use for leaning the correct ranking is an open issue. Existing ranking methods and algorithms in the information retrieval literature are potentially relevant for this purpose [9].

5 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. As the large number of dimensions in the classification shows, the decision space facing the designers of dataspaces has many aspects. In this context, a common emphasis has been on reducing start-up costs, for example 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 somewhat 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. Furthermore, there are no comparable results on automated refinement. As such, there is some way top go before the full range of dimensions associated with dataspaces are associated with substantive results, and even where this is the case there will be considerable challenges composing these results to provide dataspace deployments that meet diverse user requirements. However, dataspaces provide an overall vision that promises to enable the wider application of information integration techniques, by balancing the costs of integration activities with their benefits. The challenge of providing appropriate data integration at manageable cost seems to be of widespread relevance in widely different contexts, including personal, group, enterprise and web scale settings, acting over sources that provide computational services, structured data access and search. This chapter has sought to characterise the area, with a view to comparing the contributions to date, identifying topics for further investigation, and clarifying the space of issues of relevance to pay-as-you-go integration.

With respect to the interplay between search computing and dataspaces, we note that techniques from search computing can be borrowed to address issues that arise within dataspaces, and vice versa. In particular, search computing techniques can be used in dataspaces when queries need to be evaluated using search sources. On the other hand, the pay-as-you-go dataspace philosophy can be used in search computing for incrementally defining ranking functions based on feedback supplied by end users.

References

1. Amer-Yahia, S., Botev, C., Shanmugasundaram, J.: Texquery: a full-text search extension to xquery. In: WWW 2004: Proceedings of the 13th international conference on World Wide Web, pp. 583–594. ACM, New York (2004)
2. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. VLDB J. 17(6), 1347–1370 (2008)
3. Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A., Hedeler, C.: Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In: EDBT (2010)
4. Blunschi, L., Dittrich, J.-P., Girard, O.R., Karakashian, S.K., Salles, M.A.V.: A dataspace odyssey: The imemex personal dataspace management system (demo). In: CIDR, pp. 114–119 (2007)
5. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of multi-domain queries on the web. PVLDB 1(1), 562–573 (2008)
6. Cafarella, M.J., Etzioni, O.: A search engine for natural language applications. In: WWW 2005: Proceedings of the 14th international conference on World Wide Web, pp. 442–452. ACM, New York (2005)
7. Cafarella, M.J., Halevy, A.Y., Khoussainova, N.: Data integration for the relational web. PVLDB 2(1), 1090–1101 (2009)
8. Chakrabarti, S., Puniyani, K., Das, S.: Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In: WWW 2006: Proceedings of the 15th international conference on World Wide Web, pp. 717–726. ACM, New York (2006)
9. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic information retrieval approach for ranking of database query results. ACM Trans. Database Syst. 31(3), 1134–1168 (2006)
10. Dittrich, J.-P., Salles, M.A.V.: idm: A unified and versatile data model for personal dataspace management. In: VLDB 2006: 32nd International Conference on Very Large Data Bases, pp. 367–378. ACM, New York (2006)
11. Doan, A., Ramakrishnan, R., Chen, F., DeRose, P., Lee, Y., McCann, R., Sayyadian, M., Shen, W.: Community information management. IEEE Data Eng. Bull. 29(1), 64–72 (2006)
12. Dong, X., Halevy, A.Y.: A platform for personal information management and integration. In: CIDR 2005, pp. 119–130 (2005)
13. Dong, X., Halevy, A.Y., Yu, C.: Data integration with uncertainty. In: VLDB 2007: 33rd International Conference on Very Large Data Bases, pp. 687–698 (2007)
14. Dong, X.L., Halevy, A.Y., Yu, C.: Data integration with uncertainty. VLDB J. 18(2), 469–500 (2009)

15. Florescu, D., Kossmann, D., Manolescu, I.: Integrating keyword search into xml query processing. In: Proceedings of the 9th international World Wide Web conference on Computer networks: the international journal of computer and telecommunications netowrking, pp. 119–135. North-Holland Publishing Co., Amsterdam (2000)
16. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record 34(4), 27–33 (2005)
17. Haas, L., Lin, E., Roth, M.: Data integration through database federation. IBM Systems Journal 41(4), 578–596 (2002)
18. Halevy, A., Franklin, M., Maier, D.: Principles of dataspace systems. In: PODS 2006: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–9. ACM, New York (2006)
19. Hedeler, C., Belhajjame, K., Fernandes, A.A.A., Embury, S.M., Paton, N.W.: Dimensions of dataspaces. In: Sexton, A.P. (ed.) BNCOD 2009. LNCS, vol. 5588, pp. 55–66. Springer, Heidelberg (2009)
20. Howe, B., Maier, D., Rayner, N., Rucker, J.: Quarrying dataspaces: Schemaless profiling of unfamiliar information sources. In: ICDE Workshops, pp. 270–277. IEEE Computer Society, Los Alamitos (2008)
21. Ives, Z.G., Green, T.J., Karvounarakis, G., Taylor, N.E., Tannen, V., Talukdar, P.P., Jacob, M., Pereira, F.: The orchestra collaborative data sharing system. SIGMOD Record 37(3), 26–32 (2008)
22. Ives, Z.G., Knoblock, C.A., Minton, S., Jacob, M., Talukdar, P.P., Tuchinda, R., Ambite, J.L., Muslea, M., Gazen, C.: Interactive data integration through smart copy & paste. In: CIDR (2009)
23. Jeffery, S.R., Franklin, M.J., Halevy, A.Y.: Pay-as-you-go user feedback for dataspace systems. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 847–860. ACM, New York (2008)
24. Leser, U., Naumann, F.: (almost) hands-off information integration for the life sciences. In: Conf. on Innovative Database Research (CIDR), pp. 131–143 (2005)
25. Llu, J., Dong, X., Halevy, A.: Answering structured queries on unstructured data. In: WebDB 2006, pp. 25–30 (2006)
26. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.: Corpus-based schema matching. In: International Conference on Data Engineering (ICDE 2005), pp. 57–68 (2005)
27. 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 2007: Third Biennial Conference on Innovative Data Systems Research, pp. 342–350 (2007)
28. 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, pp. 342–350 (2007)
29. McCann, R., Shen, W., Doan, A.: Matching schemas in online communities: A web 2.0 approach. In: ICDE, pp. 110–119 (2008)
30. 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(1), 78–83 (2001)
31. Pottinger, R., Bernstein, P.A.: Schema merging and mapping creation for relational sources. In: EDBT, pp. 73–84 (2008)
32. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal: Very Large Data Bases 10(4), 334–350 (2001)
33. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. 10(4), 334–350 (2001)

34. Salles, M.A.V., Dittrich, J.-P., Karakashian, S.K., Girard, O.R., Blunschi, L.: itrails: Pay-as-you-go information integration in dataspaces. In: VLDB 2007: 33rd International Conference on Very Large Data Bases, pp. 663–674. ACM, New York (2007)
35. Sarma, A.D., Dong, X., Halevy, A.: Bootstrapping pay-as-you-go data integration systems. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 861–874. ACM, New York (2008)
36. Sarma, A.D., Dong, X.L., Halevy, A.Y.: Data modeling in dataspace support platforms. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 122–138. Springer, Heidelberg (2009)
37. Talukdar, P.P., Jacob, M., Mehmood, M.S., Crammer, K., Ives, Z.G., Pereira, F., Guha, S.: Learning to create data-integrating queries. PVLDB 1(1), 785–796 (2008)
38. Tatemura, J., Chen, S., Liao, F., Po, O., Candan, K.S., Agrawal, D.: Uqbe: uncertain query by example for web service mashup. In: SIGMOD Conference, pp. 1275–1280 (2008)