

A Functional Model for Dataspaces Management Systems

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

Abstract Dataspaces management systems (DSMSs) hold the promise of pay-as-you-go data integration. We describe a comprehensive model of DSMS functionality using an algebraic style. We begin by characterizing a dataspaces life cycle and highlighting opportunities for both automation and user-driven improvement techniques. Building on the observation that many of the techniques developed in model management are of use in data integration contexts as well, we briefly introduce the model management area and explain how previous work on both data integration and model management needs extending if the full dataspaces life cycle is to be supported. We show that many model management operators already enable important functionality (e.g., the merging of schemas, the composition of mappings, etc.) and formulate these capabilities in an algebraic structure, thereby giving rise to the notion of the core functionality of a DSMS as a many-sorted algebra. Given this view, we show how core tasks in the dataspaces life cycle can be enacted by means of algebraic programs. An extended case study illustrates how such algebraic programs capture a challenging, practical scenario.

Alvaro A. A. Fernandes
University of Manchester, e-mail: a.fernandes@cs.man.ac.uk

Cornelia Hedeler
University of Manchester, e-mail: chedeler@cs.man.ac.uk

Khalid Belhajjame
University of Manchester, e-mail: khalidb@cs.man.ac.uk

Lu Mao
University of Manchester, e-mail: maol@cs.man.ac.uk

Chenjuan Guo
University of Manchester, e-mail: guoc@cs.man.ac.uk

Norman W. Paton
University of Manchester, e-mail: norm@cs.man.ac.uk

Suzanne M. Embury
University of Manchester, e-mail: sembury@cs.man.ac.uk

1 Introduction

Given the explosion in the number of data sources that are available for remote access by applications in all areas of activity, it is no surprise that the problem of reconciling the inherent semantic heterogeneity, which such independently designed and maintained sources cannot but exhibit, has grown in importance in the information management area. This problem, under the label of *data integration* [5, 20, 30, 42], has been the focus of attention for more than fifteen years. Much progress has been made and several enduring contributions can be discerned such as the idea of mediator-wrapper architectures and techniques for view-based query rewriting that allow a query that is posed against an integration schema to be rewritten as a set of separate queries against the many independent, remote sources it draws data from [28]. Most of these techniques are grounded on two basic capabilities: the first is the ability to perform a semantic matching operation between two sources (typically at both schema and instance levels) that ultimately yields semantic correspondences between them; the second is the ability to derive from these correspondences a semantic mapping, i.e., ultimately, an executable expression that can correctly populate concepts in one schema with instances drawn from concepts in the other.

This technical progress notwithstanding, *data integration systems* (DISs) have only been successful in circumstances where (a) the scale of the required integration is small, (b) the set of data sources (as well as their schemas) is broadly static, and (c) the integration schema has sufficient strategic importance to merit incurring the significant expenditure involved in the expert-intensive matching and mapping stages of the process. As such, and with the benefit of hindsight, one can characterize these achievements as giving rise to integrated resources that have high upfront costs and high quality from the start but whose quality may decay as a result of changes unless high maintenance costs are incurred in the expert-intensive process of propagating the changes. Unfortunately, the continuously growing need for on-the-fly, on-demand combination of data resources (as manifested in the rise of techniques such as web data mash-ups [45], for example) makes this approach (that we might call *traditional, or first-generation, data integration*) fail in terms of cost-effectiveness.

In response to this state of affairs, the idea of *pay-as-you-go data integration* has gained momentum. The realization has grown that users would rather have something that produces results that may be tentative at the start than to have nothing at all (or to have to wait long and pay a lot to have something close to perfect). The vision of *dataspaces* originally proposed by Halevy, Franklin and Maier [26, 29] is that such artefacts will enable agile data integration with much lower upfront and maintenance costs. This *second-generation* approach to data integration is founded, therefore, on the pervasive use of automation in the bootstrapping stages, including the identification of matches and their representation as semantic correspondences and the derivation of mappings, on the one hand, and on continuous improvement based on user feedback and automated change propagation, on the other hand. The underlying strategy is based on the idea that if the upfront costs are low, speculative

integrations can be attempted among which some will prove useful. Those that do prove useful in principle will thereby motivate users to provide the feedback that, over time, will compensate for the shortcomings associated with the pervasive use of automation in initializing and maintaining the integrated resource.

Among the different conceptions associated with *dataspace management systems* (DSMSs), we have pursued one [6, 32, 33, 47] founded on model management research [2, 8, 9]. Here, we build upon our previous work with a view to describing a comprehensive model of dataspace functionality using an algebraic style. Our main contribution, therefore, is a formalization of how a dataspace can be operated on, i.e., of a functional model for dataspace, building on a history of advances by data integration and model management researchers whilst giving crisp contours to the notion of a dataspace, contours that had been hitherto only ambiguously and vaguely drawn.

In our work, we adopt a conception of dataspace as building upon existing research areas. We will make the relationships clearer later but, broadly speaking, we see dataspace as being dependent on automating mapping generation, for which one can use model-management techniques, whose outcome is improved through user feedback. This means that we see model-management techniques primarily as a tool for data integration of existing data resources.

It is also useful to consider a dataspace life cycle. This is because a DSMS is a data integration system that relies heavily on automation for bootstrapping and on feedback for improvement, while still having to respond effectively and efficiently to changes to data sources by propagating them through.

We envisage dataspace to have a **life cycle** with the following main stages, or phases: initialization, use, maintenance and improvement [31]. As expected, the initialization stage is a one-off, whereas use, maintenance and improvement phases transition from one to the other until the dataspace is disposed of. We now discuss the stages in slightly more detail.

Initialization: A dataspace must be initialized, or bootstrapped, typically relying on automation whenever past DISs relied heavily on human expertise. In this phase, sources are identified (or discovered), and possibly ranked, before being chosen for participation in the dataspace. Matching techniques [59] can then be used to yield associations between sources that are scored in terms of similarity. Such associations can give rise to semantic correspondences (e.g., of the kind studied in [41], e.g., that a concept c in a source s is horizontally partitioned into two concepts c_1 and c_2 in another source s').

The semantic correspondences generated over a set of schemas can then be operated upon using model-management techniques [9]. For example, one can take pairs of semantic correspondences and *compose* them, so that if a construct c_1 corresponds to a construct c_2 , and c_2 corresponds to a construct c_3 , one can obtain from them the semantic correspondence that relates c_1 to c_3 . As another example, the model management technique known as *merge* takes two models (e.g., schemas) along with whatever semantic correspondences hold between them and produces from these a new model that reflects the input correspondences along with two new semantic correspondences that relate the new model to each of the

models passed as input. This is, of course, crucial to generate integrated schemas. Note that, unlike in traditional DISs, more than one integrated schema can co-exist in such a dataspace. Semantic correspondences, in turn, can give rise to mappings (e.g., a view that describes c in s as the union of c_1 and c_2 in s'). Once mappings are available, the dataspace has been initialized and is ready for use.

We note that the model-management literature originally focussed on semantic correspondences that are relatively inexpressive. Indeed, [9] seems to be the first paper in that literature to make a case for more expressive semantic correspondences. In our own work, as explained below, we have distinguished between associations, which, as the outcome of matching techniques, are interpretable as claims of similarity; correspondences, which are associations for which there exists evidence that they capture semantic heterogeneity; and mappings, which are correspondences to which one can attach (through algorithmic derivation or through human expertise) an executable expression that reconciles the semantic heterogeneity that the correspondence captures, thereby allowing their use in view-based query evaluation against a mediated schema over autonomous data sources.

Use: A dataspace can be queried using the traditional DIS wrapper-mediator architecture in which a query q against an integrated schema S is rewritten, using the mappings available, in terms of queries q_1, \dots, q_n against sources s_1, \dots, s_n . The results r_1, \dots, r_n from those queries are rewritten, again using mappings, into a result s for the query q against S .

In our own work, we have focussed on query evaluation, but there are many other ways in which a dataspace could be used, e.g., browsing [35], keyword searching [14, 43, 46, 62], or interaction based on the notion of trails [18, 64].

Improvement: This stage is characteristic of dataspace and aims to counteract the shortcomings ensuing from the reliance on automation for bootstrapping (the other characteristic feature of DSMSs in this context). Here, feedback is gathered to which the system response is, under the pay-as-you-go approach, to make the most of the feedback that is given in terms of whatever increase in perceived quality can be obtained from that feedback.

In our own work [6], we have used feedback on query results to annotate, select and refine the collection of mappings that can be used to answer a given query by taking into account the possibilities for trading off precision and recall. It is possible and useful to gather other kinds of feedback, e.g., on queries [15]; on mappings [1]; or for query specification [61, 62].

Maintenance: Changes to the sources must be propagated throughout. Thus, changes in source schemas need to be reflected in the mappings, changes in source extents may change the scores that can be assigned to associations, thereby potentially changing which semantic correspondences (and hence mappings) are backed by sufficient empirical evidence.

Fig. 1 depicts this life cycle in abstract form (using a loosely-notated state-transition diagram, in which ε denotes an automatic transition) while Fig. 2 zooms into the initialization phase (the other phases being, of course, more application/-scenario-specific and, hence, less amenable to being abstracted in the same way).

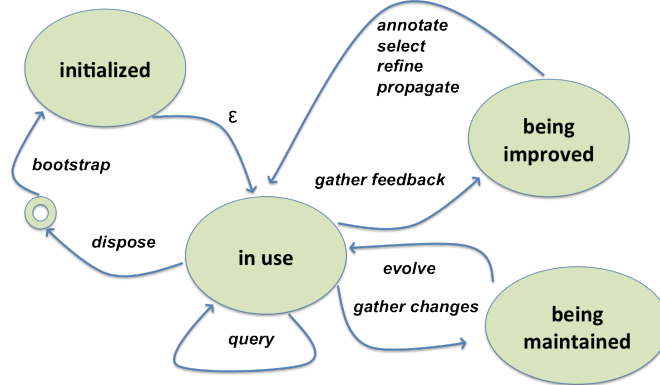


Fig. 1 Dataspace: Life cycle

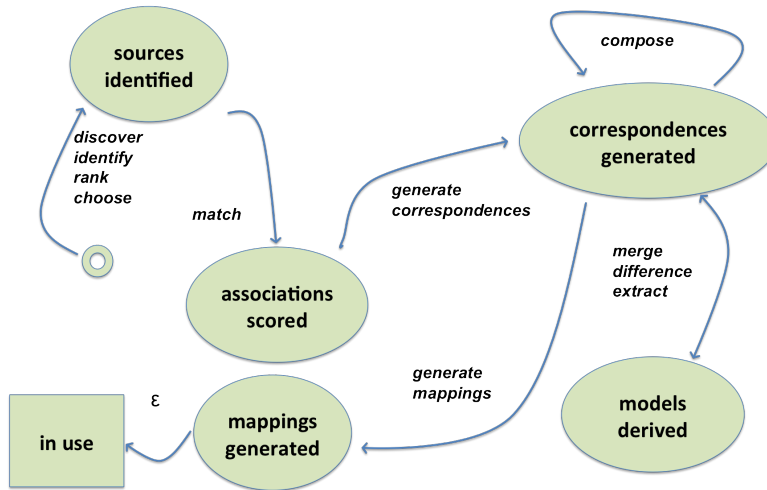


Fig. 2 Dataspace life cycle: Initialization stage

2 Background

This section briefly recalls prior work on DSMSs and on model management. In the case of DSMSs, the picture that emerges is of a field mostly characterized by attempts at defining components (which deliver partial functionality only) or point solutions (i.e., DSMSs that are very specialized either by targeting a specific application domain or by espousing assumptions that significantly narrow the problem scope). In the case of model management, most research has stopped short of tying operations on models to operations on the modelled constructs as supported by traditional *database management systems* (DBMSs). Indeed, proposals for *model management systems* (MMSs) have mainly focussed on the capabilities needed to

manipulate and evolve collections of models, as opposed to those that must be available for one to be able to use and improve them. For example, MMSs have often targeted metadata-intensive applications (those in which operations like translating models, manipulating mappings, etc., occur very often) without fully supporting the use of the managed models and mappings (e.g., in query answering, or in extract-transform-load application in data warehousing).

2.1 Dataspace Management Systems

We have recently surveyed the literature on dataspace and refer the reader to the detailed findings there [33]. In that work, we devised a classification framework (extending our previous work [31]) in order to characterize and contrast a number of dataspace proposals. Here, we simply delineate the major categories in that framework in order to draw an overall picture of how dataspace are seen by the researchers involved, and, in particular, with respect to what is perceived to be the functionality a DSMS should provide.

DB2 [27] can be seen as a baseline system. It provides queries over heterogeneous sources using a mapping-based mediation approach where mappings are built with significant human intervention, but, significantly, some of the advances resulting from the Clio project [55, 56, 16, 34] were incorporated into DB2. ALADIN [43] supports semi-automatic data integration in the life sciences, with the aim of easing the addition of new data sources. ALADIN is, in this sense, an example of the importance of taking a life cycle view of dataspace. SEMEX [22, 44] integrates personal information. It requires a merged model to be provided upfront and uses it as a pivot to match with and map into the constituent sources. SEMEX responds to changes in sources and to the addition of new sources. iMeMeX [18, 10] also targets personal information, and supports incremental improvement through the manual provision of path-based queries known as iTrails [64]. PayGo [46] aims to integrate web resources. It relies on automation to produce a union schema, uses match to determine how similar constituent schemas are and clusters them. PayGo supports keyword searches but reformulates them into queries that attempt to identify relevant sources using the clusters previously built. UDI [17, 23, 24, 60] 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. In this respect, the conception of a dataspace adopted in UDI is close to the one adopted in our work. Roomba [38] 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. ORCHESTRA [62, 36], a collaborative data sharing system, covering the three phases initialization, 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. Cimple [21, 48] aims 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. CopyCat [37] 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. Similar to CopyCat, OCTOPUS [14] provides the means for integrating multiple sources on the web interactively by providing several operations that can be used 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. Neither CopyCat nor OCTOPUS distinguish between the various phases of the dataspace life-cycle, e.g., initialization, usage, and improvement. Instead, they promote a seamless combination of initialization, usage and improvement of the dataspace, albeit with a fair amount of user input required.

This brief overview shows that there is broad consensus that dataspace should be construed as second-generation integration platforms in which pervasive automation is deployed to push down costs and user feedback is gathered and responded to increase result quality.

2.2 *Model Management Systems*

The idea of endowing information management systems with the capability to manage models was first formulated in [8] and then revisited in [9]. By *model* is meant, in the information management context, an intensional description of a data resource. By *management* is meant, here, the reification of models as objects of a generic type over which an algebra (i.e., operations on models) is defined.

Model management can be seen as an attempt to simplify the support for a large set of information management tasks that involve operating on models and, in particular, morphisms between models. As listed in [9], such tasks include extract-transform-load in data warehousing; message or data translation; designing portals, forms processors, query interfaces, report writers; and, of particular interest here, wrapper and mediator generation. The most important model-management operations are *match* (which takes models and yields associations or correspondences, i.e., morphisms, between model elements), *merge* (which takes models and correspondences between them and yields a merged model obtained from the input models and correspondences between the former and each of the latter), *extract* (which takes models and correspondences between them and yields the submodel of one of the inputs that can be populated using the input correspondences), *diff* (which takes models and correspondences between them and yields the submodel of one of the inputs that cannot be populated using the input correspondences), and *compose* (which takes two sets of correspondences and yields the correspondences that comprise the composition morphism of the two inputs, seen as morphisms).

The simplification sought by the model management vision is seen to arise from the generic nature of internal representations and the high-level, expressive nature of the generic operations defined on them. The algebraic approach implies compositionality and, for subsets of the algebra, closure. The overall outcome is envisaged as an expressive algebraic language that can formalize important usage scenarios that would otherwise be quite complex to specify precisely in an error-free manner due to the ad-hoc nature of the representations and the transformations used, as induced by the multiplicity of concrete data modelling formalisms adopted at the user/application level.

The model management area has been surveyed in [8] with respect to previous research that inspired the vision, and in [9] with respect to the proposals that emerged in the wake of [8]. The literature is quite vast and encompasses topics, such as conceptual similarity matching, that, in themselves, have seen voluminous research activity [59].

Broadly construed to encompass systems that are primarily concerned with matching management or with mapping management, as well as the full gamut of model management capabilities, the area has produced impressive research systems such as COMA [19, 4], Clio [55, 56, 16, 34] (some of whose contributions have been incorporated into DB2), AutoMed [12, 11, 58], Rondo [54, 53, 51], GeRoMe [40] and MISM/MIDST [2, 3], among others.

3 Functional Model

The purpose of this section is the formulation of a functional model of dataspace as an algebra that modifies, extends, and complements the core model management techniques in the literature. We make no claim either that different conceptions of dataspace than emerge from the formulation below are in any way less preferable

nor that different formulations of the same conception than the one described below are not possible or as desirable.

We start with the assumption that dataspace are derived from a collection of existing data resources. We conclude with an algebraic account of the functionalities that our conception of dataspace makes available. Later, we show how these functionalities can capture interesting, challenging data management scenarios. Firstly, we provide a broad overview of how we construe DSMSs in relation to DBMSs, DISs and MMSs.

3.1 An Overview

As already mentioned, we construe DSMSs as combining, adapting and extending the functionality of DBMSs and MMSs. In this respect, we draw the contrast that DISs do not resort to automated matching and mapping generation techniques from MMSs and hence are not as dependent on gathering and responding to user feedback. As broadly suggested by Figs. 1 and 2, it is possible to provide a clearer picture of the functional relationships between these four classes of systems by building upon and extending previous work in the areas surveyed in Sec. 2. The goal, in this case, is to elucidate, with respect to existing functionality from DBMSs, DISs, and MMSs, what parts of it can be built upon, what changes and extensions to it are needed, and what additions to it are required.

Fig. 3 offers a broad overview of how we construe the functional relationships between the four classes of systems. The notation is, loosely, that of data flow diagrams in which arrows denote data (whose types are made clear below), darker boxes denote operations and lighter ovals denote stores. The irregular, dashed-line shapes are used to informally denote the boundaries between the four classes of system involved thereby roughly indicating what *distinctive* functionality they contribute. We do not mean to imply a strictly component-based architecture by this notational device, e.g., a DIS often comes with its own internal, specialized query evaluation component, and so may a DSMS.

With Fig. 3 we only aim to note that DSMSs build upon techniques and mechanisms that have been explored in the literature on DBMSs, DISs, and MMSs. Thus, broadly speaking, query evaluation is a core concern in DBMS research, the provision of explicit mappings that give rise to integrated models over existing sources and allow querying over such integrated models to take place lies at the heart of DIS research, and MMS research has focussed from the outset on providing correspondence-driven operations on models by means of which new mappings and models can be derived. In this respect, DSMSs benefit from the research results in those areas and bring a specific concern with improving query results by using feedback to compensate for the shortcomings of the pervasive use of automation instead of intensive reliance on human experts.

The remainder of this section contains our main contribution, viz., an algebraic formulation of the broad functionality depicted in Fig. 3. Inspiration for

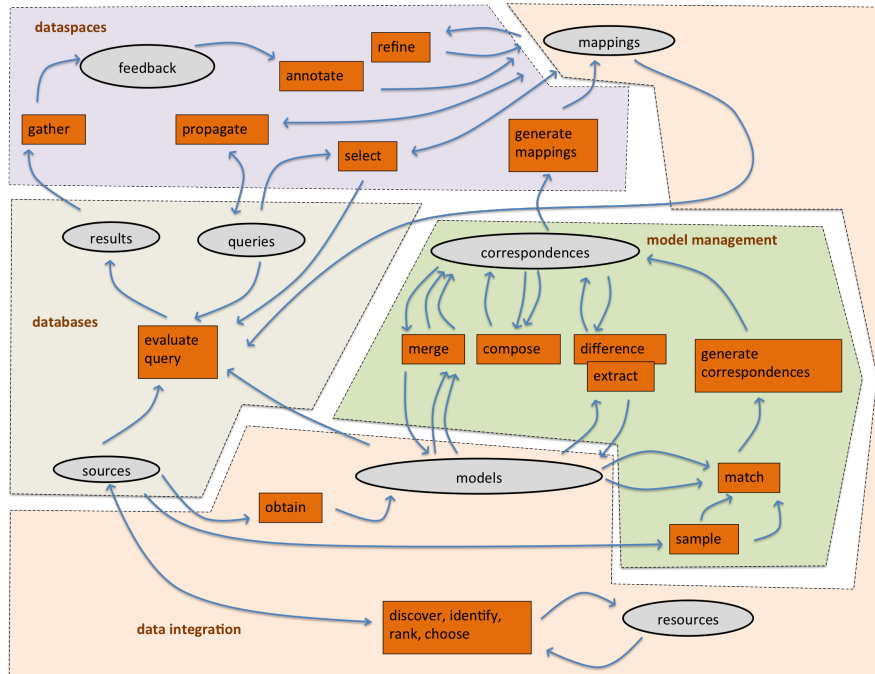


Fig. 3 Relating databases, data integration, model management and dataspace

this work was drawn primarily from the work on model management, particularly [2, 3, 54, 53, 51]. As we have pointed out, different formulations of the same conception than the one described below (e.g., drawing inspiration from [40] or [12, 11, 58]) are possible and, possibly, as desirable. Our formulation is, to the best of our knowledge, to increase the visibility of implicit dependencies, intersections and complementarities between the characteristic functionality of DBMSs, DISs, MMSs and DSMS, and, with respect to the latter, to characterize more clearly that the pervasive use of automation and feedback-based improvement is what make dataspace distinctive from other classes of data management artefacts.

3.2 Preliminary Assumptions

We assume the existence of a collection \mathbb{D} of data resources. We further assume that a data resource $d \in \mathbb{D}$ can be associated with an intensional description (e.g., a set of statements in some data definition language), as made more precise later, but we abstract over the precise semantics that underpins such intensional descriptions (e.g., the semantics of such modelling notions as inheritance, referential integrity, and

similar ones) insofar as our formulation aims to be agnostic as to the data modelling theory or paradigm used for individual data resources.

One of the main purposes of an intensional description is, of course, to characterize the valid states of the corresponding data resource. In other words, from the intensional description of a data resource, one must be able to formally decide whether the data resource is in a valid, or conformant, state at any point in time. Here, too, we do not go into the detail of this validity relation, and, instead, assume that the concrete implementations of the many operations on intensional descriptions we characterize later are validity-preserving.

3.3 Intensional Descriptions

Let Σ be an infinite set of **symbols** that can act as identifiers of the concepts and values that are modelled in any collection \mathbb{D} of **data resources**.

The intensional description of a data resource $d \in \mathbb{D}$ draws a finite set $C(d) \subset \Sigma$ comprising three pairwise disjoint subsets $SL(d)$, $SA(d)$ and $SR(d)$ containing the symbols used as identifiers of, respectively (and in the terminology of [3]) **superlexicals** (e.g., attributes, in the relational data model), **superabstracts** (e.g., relations) and **superrelationships** (e.g., primary-foreign key pairings), respectively.

The intensional description of a data resource $d \in \mathbb{D}$ is, as alluded to above, associated with the set \mathbb{IC} of well-formedness conditions (often referred to as **integrity constraints**) that characterize any state of d as conformant or not to the particular modelling theory or paradigm used to design and deploy it (e.g., in the case of the relational data model, \mathbb{IC} comprises the formalization of notions such as domain integrity, entity integrity, referential integrity, etc.). Since \mathbb{IC} is common to the intensional descriptions of data resources that conform to the data modelling theory or paradigm from which \mathbb{IC} stems, we omit any reference to it henceforth, thereby assuming that every intensional description is known to be further associated with the set of integrity constraints that is appropriate for it, which, in turn, we assume to be known from the context of the occurrence of the intensional description in the text. In this sense, for the purposes of this chapter, we consider intensional descriptions as syntactic structures over which we define an algebra, as described later.

We use the term **constructs** to refer to superlexicals, superabstracts and superrelationships indistinctly. Thus, given a data resource $d \in \mathbb{D}$, the set of construct identifiers in its intensional description is $C(d) = SL(d) \cup SA(d) \cup SR(d)$.

Let $L(d)$ be a set of **literals** that are possibly specific to a resource d (and hence to the paradigm used to model d). These are used to describe constructs (e.g., in the case of a SQL-based relational data resource, one might find a schema named `personnel`, containing, among many others, a column of type `varchar` named `address` in a table named `employee`, where the terms in `sans-serif` are literals used in that data resource). We assume L to be the union of a set CT of **construct type names** (e.g., `schema`, `table`, `column`), a set CD of **domain names** (e.g., `int`, `varchar`), and a set CN of **construct names** (e.g., `personnel`, `employee`, `name`, `address`,

salary). Note that construct names and construct identifiers are distinct notions with non-overlapping extensions. Construct identifiers are a notion at the super-(or meta-)model level. They are, therefore, resource-, paradigm- and implementation-independent. Construct names are resource-specific notions (just as construct type names and domain names are paradigm- and implementation-specific ones).

As others have done (most notably [51]) we characterize the intensional descriptions of a data resource d as a graph in which non-leaf nodes are (or are labelled by) construct identifiers in C , leaf nodes are (or are labelled by) literals in $L = CN \cup CT \cup CD$, and edges are subsets of the following relations¹:

$$\begin{aligned} \text{isA} &: C \times CT \\ \text{withDomain} &: C \times CD \\ \text{isNamed} &: C \times CN \\ \text{has} &: C \times C \end{aligned}$$

We use the relation names (i.e., `isA`, `withDomain`, `isNamed`, and `has`) to label the edges of the graph and define, for a data resource $d \in \mathbb{D}$, its **intensional description** (or **model**) to be a graph $G(d)$ with node set $C(d) \cup L(d)$ and edge set $E_1 \subseteq \text{isA} \cup E_2 \subseteq \text{withDomain} \cup E_3 \subseteq \text{isNamed} \cup E_3 \subseteq \text{has}$. Note that, in most practical cases, `withDomain` is more informative for superlexicals. Note also, that one can bind a construct with its extensional description when the data resource is in a given state (indeed this is one way of construing query evaluation, where the query defines a construct whose extension is sought) and represent that with a relation `inState`, though we do not delve into this possibility here.

3.4 Sorts

Our algebraic specification makes use of several sorts, all of which are alluded to in Fig. 3 and can be construed as structuring the data from which the operations hinted at in Fig. 3 and described more formally in Sec. 3.5 draw inputs and outputs. For all the sorts described below, we assume that their elements may be described by a **feature set**. For example, an association between ‘worker’ on the one hand and ‘partTimeWorker’ and ‘fullTimeWorker’ typically has a *similarity score* $\in [0, 1]$, which helps distinguish it, when there is an occasion to use that association, from a possibly lower-scored association between ‘worker’, on the one hand, and ‘partTimer’ and ‘fullTimer’, on the other. The former might be reflected in a feature-value pair such as `similarityScore = 0.75` while the latter might have instead `similarityScore = 0.70`. Likewise, a seman-

¹ We abuse notation and omit the reference to a data resource if what we write is valid for all data resources or if the intended reference to specific data resources is clear from context. Thus, we sometimes write C and L rather than the more precise $C(d)$ and $L(d)$, and so on.

tic correspondence that postulated that ‘worker’ in schema s_1 is horizontally partitioned into ‘part-time worker’ and ‘full-time worker’ in schema s_2 might have as one of its features the list of selection predicates that decide which tuple in ‘worker’ in s_1 belongs to which partition in the corresponding relations in s_2 (e.g., $\text{selectionPredicateList} = [(s_2.\text{partTimeWorker}, s_1.\text{numberOfHours} < 8), (s_2.\text{fullTimeWorker}, s_1.\text{numberOfHours} \geq 8)]$). However, we leave the precise, comprehensive and exhaustive definition of such feature sets undiscussed here except where it is crucial for our exposition to consider one or more features.

These are the sorts used in our formulation:

resources We assume the existence of a collection \mathbb{D} of data resources to each of which an intensional description can be associated. We assume that a data resource is interacted with by means of (G)UI/API mechanisms, i.e., we take it to be an independent software artefact that exposes an interface by means of which people and systems can interact with it.

sources The set of sources in a dataspace is that subset of the available data resources \mathbb{D} that one has discovered, identified, ranked or chosen to operate on. We assume that the intensional description associated with a data source, as well as a sample of its state, can be obtained. We denote a set of sources with D .

models In line with most of the background literature, we refer to intensional descriptions (i.e., an instance of a supermodel theory or paradigm) as **models**, as defined above. We denote a set of models with M .

queries A query is an expression against a model that, upon evaluation, returns a set of results that, as mentioned above, characterizes the extension, at evaluation time, of the construct in the model defined by the query expression. We denote a set of queries with Q .

results The results returned by query evaluation can be construed as a set of superabstract instances (e.g., tuples in relation ‘employee’), each of which construed, in turn, as a set of pairs, of which the first element is a construct (e.g., a superlexical instance such as the attribute name ‘dept’) and the second denotes the state of that construct (e.g., it is a value, in this case of an attribute, in the domain of the superlexical instance such as ‘Sales’). We denote a set of query results with R .

associations With or without human intervention, data sources are matched (typically taking account of their models and, sometimes, of samples of their state) in order to produce a similarity score between constructs (e.g., that ‘worker’ in a schema is similar to both ‘partTimeWorker’ and ‘fullTimeWorker’ in another), which sets a feature in the association. We construe an association as a pair of sets of constructs. Thus, we foresee associations as being not just $1 : 1$ but also $1 : n$ and $n : m$ both at superabstract and superlexical levels. It is useful to think of a set of associations as a bidirectional morphism between sets of constructs. We denote a set of associations with A .

correspondences The associations found through matching can suggest to a human expert or to an algorithm that some semantic relationship (typically equivalence, but also subsumption, and others) is likely to hold. For example, given

strong evidence that ‘worker’ in a schema is similar to both ‘partTimeWorker’ and ‘fullTimeWorker’ in another, one could postulate that ‘worker’ in the former is horizontally partitioned (i.e., is the union of) ‘partTimeWorker’ and ‘fullTimeWorker’ in the latter, on the basis of a predicate on the attribute ‘numberOfHours’ of ‘worker’. We construe a semantic correspondence as a pair of sets of constructs. Again, it is useful to think of a set of correspondences as a bidirectional² morphism between sets of constructs and we note that, as such, they bear close structural similarity with associations. However, as hinted above, the feature set that describes an association is presumed to be different from the feature set that describes a correspondence. This reflects the distinct interpretations we place upon each and, hence, the roles they play in the algebra. An association is presumed to possess very little semantic import (essentially, it postulates a degree of syntactic and structural similarity on the basis of the evidence available to the matching algorithms that were used to derive them). In contrast, a semantic correspondence is presumed to carry significantly more semantic import (e.g., it postulates the existence of a relationship that, more or less directly, points the way to the semantic reconciliation between the related sets of constructs). The additional information that stems from the interpretation we place upon correspondences and that distinguish them from associations and mappings is assumed to be captured in a feature set. We denote a set of semantic correspondences with F .

mappings The semantic correspondences that are found to be supported by sufficient evidence can be selected by a human expert or by an algorithm for use in mapping a query against a derived, integrated model onto a set of queries against the primary models (i.e., those of the data sources included in the dataspace and involved in the query) and in translating the results emitted by the data source into results structured in terms of the derived, integrated model. We construe a mapping element as a pair of sets of constructs. This means, once more, that mappings (i.e., sets of mapping elements) are bidirectional³ morphisms between sets of constructs and, as such, are structurally similar to sets of associations and to sets of semantic correspondences, with the difference lying, again, on the feature set that describes a mapping element (e.g., as described below, mappings can be annotated for precision and recall on the basis of feedback on the results they produce when used in a query). In this respect, there is one particular characteristic of mappings, viz., that we can, and often do, construe them as views, i.e., executable expressions in a query language. Such an expression can be understood as an intensional description of the mapping (understood, extensionally, as a set of pairs of constructs). As we have shown in [47], given correspondences whose semantic import leads fairly directly to the derivation of view expressions that reconcile some semantic heterogeneities [41], we can algorithmically derive from them the view expression that allows us to populate the source construct(s) with data obtained from the target construct(s) (and *vice versa*), and make it a

² Note that if the relationship is inherently unidirectional, e.g., one of subsumption, then the construal postulated here would have to be refined.

³ Note, once more, for inherently unidirectional relationships this construal is too coarse.

bound feature in the feature set of a mapping. For example, if there is a semantic correspondence relating ‘worker’ in s_1 to ‘partTimeWorker’ and ‘fullTimeWorker’ in s_2 and if its feature set contains

```
selectionPredicateList = [
    (s2.partTimeWorker, s1.numberOfWorks < 8),
    (s2.fullTimeWorker, s1.numberOfWorks ≥ 8)]
```

then we have shown in [47] how it is possible to derive the view

$$s_1.worker \leftarrow s_2.partTimeWorker \cup s_2.fullTimeWorker$$

that populates the construct in the s_1 side of the mapping in terms of the constructs in the s_2 side, as well as the views

```
s2.partTimeWorker ← σs1.numberOfWorks < 8 s1.worker
s2.fullTimeWorker ← σs1.numberOfWorks ≥ 8 s1.worker
```

that populates the constructs in the s_2 side of the mapping in terms of the construct in the s_1 side. In summary, in the case of mappings, we sometimes construe them intensionally as views and sometimes extensionally, as a set of mapping elements (i.e., pairs of sets of constructs). In what follows, we make use of both construals (as exemplified above) without remarking on it unless the context does not suffice to make it clear which one we are using. We denote a set of mapping elements with V .

feedback The central tenet of a pay-as-you-go approach to dataspace is the recourse to feedback to compensate for the shortcoming of pervasively automating the generation of associations, correspondences and mappings. Feedback can take many forms and an open model of what constitutes feedback is desirable while dataspace research has not yet reached maturity. In our own work [6], we have explored a particular kind of user feedback. More specifically, given the result of evaluating a query that relied on mappings from an integrated schema onto existing sources, if a user provides feedback as to which tuples are true positives (i.e., are in the result and should have been), false positives (i.e., are in the result but should not have been) and false negatives (i.e., are not in the result but should have been), we have shown how such feedback can be used to annotate mappings with estimates of quality (in the form of estimates of precision and recall) that are then used to select which mappings to use in answering queries, as well as to refine mappings (i.e., to derive new mappings from existing ones that are estimated to have better quality than the latter). We denote a set of feedback instances with U .

3.5 Operations

This section introduces two groups of operations. The first group acts on elements of the sorts above, or collections formed with such elements. This first group is not

characteristic of the functionality exhibited by DBMSs, DISs, MMSs or DSMSs. The second group is, in contrast, characteristic of such systems and constitutes the main focus of interest.

3.5.1 Structural Operations

The first group of operations are structural in intention, i.e., they access, transform and derive new elements of the three underlying collection types, viz., sets, graphs, and morphisms. In this sense, they are paradigm- and domain-independent and, mainly, supportive. They were first proposed in [52] and revisited in [51]. Our account here is closer to the former work in using set comprehension notation for the definition of the operations.

In defining this first group of operations, we use unconventional notation, as follows. Whenever in a signature the type \mathbb{F} occurs, we mean that the signature is valid if every occurrence of \mathbb{F} is simultaneously bound by one of A , F or V (i.e., associations, correspondences or mappings, resp.). Whenever the type \mathbb{S} occurs, the signature is valid if every occurrence of \mathbb{S} is simultaneously bound to either C or L (i.e., construct names or literals, resp.); and, finally, whenever the type \mathbb{X} occurs, the signature is valid if every occurrence of \mathbb{X} is simultaneously bound to the same sort. The signatures of the structural operations are given in Fig. 4, and their definitions, using set comprehension notation, are given in Fig. 5. We note that it is possible to formally define the semantics of these operations more precisely because of their being paradigm- and domain-independent. In contrast, for the second group of operations, their precise definition is more dependent on a modelling paradigm or theory being fixed and made more precise than we can do in the confines of this specific document.

We note that, using the set comprehension notation employed in Fig. 5, we can express the transformation of a set of sets X into the iterative union of its elements as $\text{iUnion}(X) \equiv \{x' | x \leftarrow X, x' \in x\}$. Correspondingly, assuming a set of membership predicates $P = \{p_1, p_2, \dots, p_n\}$, we can express the transformation of a set X into a set of n subsets $\{X_1, X_2, \dots, X_n\}$, where X_i contains those elements for which $p_i \in P$ is true, as $\text{setOfSubsetsOf}(X, P) \equiv \{\{x | x \leftarrow X, p(x)\} | p \in P\}$.

In what follows, the operations in Figs. 4 and 5 are not resorted to intensively (though a few definitions in Sec. 3.5.5 depend on some of them). This is because the scenarios in Sec. 4 can by and large be handled by the operations in Fig. 6. However, those operations are necessary for manipulation of structures (e.g., to convert objects of one sort into objects of another sort). As such, along with syntax for transforming sets into sets of sets and *vice versa*, they are important in making the algebra more effective in modelling less simplified use cases and scenarios than those in Sec. 4.

Primitive

$\text{genId} : \mathbb{S} \rightarrow \mathbb{S}$
 $\text{identity} : \mathbb{S} \rightarrow \mathbb{F}$
 $\text{domain} : \mathbb{F} \rightarrow \mathbb{S}$
 $\text{invert} : \mathbb{F} \rightarrow \mathbb{F}$
 $\text{restrictD} : \mathbb{F} \times \mathbb{S} \rightarrow \mathbb{F}$
 $\text{transitiveClosure} : \mathbb{F} \rightarrow \mathbb{F}$
 $\text{constructsIn} : M \rightarrow \mathbb{S}$
 $\text{copyUpdate} : M \times \mathbb{S} \rightarrow M$
 $\text{submodelOf} : M \times \mathbb{S} \rightarrow M$

Derived

$\text{range} : \mathbb{F} \rightarrow \mathbb{S}$
 $\text{restrictR} : \mathbb{F} \times \mathbb{S} \rightarrow \mathbb{F}$
 $\text{restrict} : \mathbb{F} \times M \times M \rightarrow \mathbb{F}$
 $\text{traverse} : \mathbb{S} \times \mathbb{F} \rightarrow \mathbb{S}$

Generic

$\text{union} : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{X}$
 $\text{minus} : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{X}$
 $\text{intersection} : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{X}$

Fig. 4 Structural operations: Signatures

3.5.2 Dataspace Operations

The second group of operations is characteristic of the functionality exhibited by DBMSs, DISs, MMSs or DSMSs. Their signatures are given, grouped by system type, in Fig. 6. We construe DISs to subsume the functionality of DBMSs and that of DSMSs to subsume the functionalities of DBMSs, DISs and MMSs. MMSs are seen as providing useful functionality for both DBMSs and DISs and have, therefore, typically been construed as standalone systems with respect to them. As far as we are aware, ours is the first detailed proposal that construes DSMSs as building upon MMS. We note that to assign these operations a precise formal definition is somewhat dependent on a modelling paradigm or theory being fixed (e.g., this is the case for `evalQ`). A semantics for the model-management operations is proposed (and studied for its formal properties) in some detail in [52], which addresses the challenges identified in [8].

Primitive

$$\begin{aligned}
\text{genId}(X) &\equiv x \in \Sigma \wedge x \notin X \\
\text{identity}(X) &\equiv \{(x,x) \mid x \leftarrow X\} \\
\text{domain}(X) &\equiv \{x \mid (x,y) \leftarrow X\} \\
\text{invert}(X) &\equiv \{(y,x) \mid (x,y) \leftarrow X\} \\
\text{restrictD}(X,X') &\equiv \{(x,y) \mid (x,y) \leftarrow X, x \in X'\} \\
\text{transitiveClosure}(X) &\equiv \{(x,y) \mid (x,y) \leftarrow X\} \cup \{(x,z) \mid (x,y) \leftarrow X, (y,z) \leftarrow X\} \\
\text{constructsIn}(X) &\equiv \{x \mid (x,y) \leftarrow X, x \in C(X)\} \cup \{y \mid (x,y) \leftarrow X, y \in C(X)\} \\
\text{copyUpdate}(X,X') &\equiv \{(z,y) \mid (x,y) \leftarrow X, x \in X', z \equiv \text{genId}(X)\} \\
&\quad \cup \{(x,z') \mid (x,y) \leftarrow X, y \in X', z' \equiv \text{genId}(X \cup \{z\})\} \\
\text{submodelOf}(X,X') &\equiv \{(x,y) \mid (x,y) \leftarrow X, x \in X' \vee y \in X'\}
\end{aligned}$$

Derived

$$\begin{aligned}
\text{range}(X) &\equiv \text{domain}(\text{invert}(X)) \\
\text{restrictR}(X,X') &\equiv \text{invert}(\text{restrictD}(\text{invert}(X),X')) \\
\text{restrict}(X,X',X'') &\equiv \text{restrictR}(\text{restrictD}(X, \text{constructsIn}(X')), \text{constructsIn}(X'')) \\
\text{traverse}(X,X') &\equiv \text{range}(\text{restrictD}(X,X'))
\end{aligned}$$

Generic

$$\begin{aligned}
\text{union}(X,X') &\equiv \{x \mid x \leftarrow X\} \cup \{x \mid x \leftarrow X'\} \\
\text{minus}(X,X') &\equiv \{x \mid x \leftarrow X, x \notin X'\} \\
\text{intersection}(X,X') &\equiv \{x \mid x \leftarrow X, x \in X'\}
\end{aligned}$$

Fig. 5 Structural operations: Set comprehension semantics

We now comment on the operations whose signatures are given in Fig. 6. Again, we group them by system type.

3.5.3 Data Source Operations

$r := \text{evalQ}(d, q)$ A query q can be evaluated against a data source d to produce a result r . The modelling theory or paradigm that constrains the representations of superabstracts, superrelationships and superlexicals in d and the syntax and semantics of q define the representation and semantics of the result r . In the classical case of the relational model, d can be construed as a set of tables, q as relational algebraic expression over d and r is characterized by the semantics of the relational algebraic language of which q is an element.

DBMS

$$\text{evalQ} : D \times Q \rightarrow R$$

DIS

$$\text{discover} : \mathbb{D} \rightarrow \mathbb{D}$$

$$\text{identify} : \mathbb{D} \rightarrow D$$

$$\text{rank} : D \rightarrow D$$

$$\text{choose} : D \rightarrow D$$

$$\text{obtain} : D \rightarrow M$$

$$\text{evalIQ} : Q \times M \times V \rightarrow R$$

MMS

$$\text{sample} : D \rightarrow D$$

$$\text{match} : D \times D \times M \times M \rightarrow A$$

$$\text{inferCorrespondences} : A \rightarrow F$$

$$\text{compose} : F \times F \rightarrow F$$

$$\text{merge} : M \times M \times F \rightarrow M \times F \times F$$

$$\text{extract} : M \times F \rightarrow M \times F$$

$$\text{difference} : M \times F \rightarrow M \times F$$

DSMS

$$\text{viewGen} : F \rightarrow V$$

$$\text{gather} : R \rightarrow U$$

$$\text{annotate} : U \rightarrow V$$

$$\text{select} : Q \times V \rightarrow V$$

$$\text{refine} : V \rightarrow V$$

$$\text{propagateToV} : V \rightarrow V$$

$$\text{propagateToQ} : V \times Q \rightarrow Q$$

Fig. 6 Dataspace operations: Signatures

3.5.4 Data Integration Operations

$D1 := \text{discover}(D); D2 := \text{identify}(D1); D3 := \text{rank}(D2); D' := \text{choose}(D3)$

It seems important to distinguish within the set D of available data resources that subset D' of it whose elements are believed to be useful, usable and used. There are obviously many ways in which the distinction can be drawn on the basis of different set of operations. Here, for illustration only, we assume that resources are discovered (by which we mean ascertained to be useful through mechanisms like search engines, directory services, etc.), identified (by which we mean ascertained to be usable, e.g., open to being accessed, in possession of appropriate APIs, sufficiently described, etc.), ranked (by which we mean scored with metrics such as authoritativeness, freshness, completeness, etc.) and chosen (by which we mean made members of the dataspace) for actual use. We note that while we seem, above, to suggest that these four operations thread their outputs into inputs, sequentially, this need not be the case.

$m := \text{obtain}(d)$ A model m of a data source d must be obtained so that, as a first class object, it can be scrutinized and manipulated. Under the assumption that the DSMS uses a supermodel theory or paradigm, then m is the outcome of a translation step from the theory or paradigm that constrains d into the DSMS supermodel.

$r := \text{evalIQ}(q, m, \nu)$ This operation stands in contrast to its single-source form in Sec. 3.5.3 above. Here, a query q can be evaluated against an integrated set of data D sources to produce a result r , where the integration arises from a set of mappings ν over the mediated model m obtained from D . In a global-as-view approach [28], the query q posed against the mediated schema arising from the integration of D is translated using the mapping elements in ν over m into single-source queries (see [28] for a survey of the techniques involved). Thus, if $D = \{d_1, d_2\}$, then, broadly speaking, two operations are issued, viz., $r_1 := \text{evalQ}(d_1, q_1)$ and $r_2 := \text{evalQ}(d_2, q_2)$ using ν over m to derive both q_1 and q_2 from q , and, on the way back, r from r_1 and r_2 .

3.5.5 Model Management Operations

$d' := \text{sample}(d)$ Given a data source d , a sample d' of it can be constructed. This operation, among other potential uses, enables a principled reduction (by which we mean one that is representative of the information content of d) in the size of the inputs to the `match` operation. While many MMSs constrain themselves to matching at schema-level only, additionally matching at instance-level is often a more effective policy if it can be done efficiently, and we construe `sample` as contributing to this purpose. In implementing this operation (and many others below), one expects further parameters (e.g., in the case of `sample`, the desired size of the sample, whether it should be drawn from some given distribution, etc.) but, here, we focus on the abstract operation.

$a := \text{match}(d, d', m, m')$ Given two data sources (or samples thereof) d and d' with their obtained models m and m' respectively, a matching algorithm is used to obtain a set a of associations between them, where each element in a is a pair in which the first and the second element are sets of constructs. Each such pair has in its feature set an assigned `similarityScore` produced by the matching algorithm. There are many ways in which this operation can be implemented [59], the most important dimensions of variation in the present context perhaps being (a) whether the operation uses, internally, multiple matchers making a the result of an aggregation of similarity scores, possibly independently, produced by distinct matching algorithms; (b) whether both instance- and schema-level information is taken as evidence of similarity or only one of them (typically, schema-level in that case); (c) whether, as suggested above, matching is done pairwise or, alternatively, whether matching is done over a set of data sources taken together, with the former approach being based on an assumption that the associations a_1, a_2, \dots, a_n resulting from a sequence of pairwise matching operations can be aggregated (e.g., by simple union); and (d) whether the associations are, as suggested above, between sets of constructs or between single constructs. In implementing this operation, one might use one or many algorithms, and in the latter case, one might use different strategies for aggregating the scores returned by individual algorithms (see, e.g., [19, 4]).

$f := \text{inferCorrespondences}(a)$ Given a set a of associations, a set f of semantic correspondences can be generated from it. As pointed out in Sec. 3.4, semantic correspondences only differ from associations in their information content, i.e., how they are interpretable. It was also pointed out that this distinction can be construed as being captured in a different feature set which annotates associations and correspondences differently. Thus, the former is annotated with, essentially, a similarity score, while the latter are annotated with sufficient information for mappings to be derived that mediate between the sets of constructs involved. Now, invoking $f := \text{inferCorrespondences}(a)$ derives from a subset of the associations in a a set f for each of whose elements, on the basis of matching evidence (i.e., the similarity scores associated with the elements of a), it can be postulated that it can be interpreted as a semantic correspondence. In our own work [47], we have focussed on the semantic correspondences identified in [41] (essentially, same name for distinct constructs, distinct names for the same construct, missing constructs, horizontally- or vertically-partitioned constructs).

$f := \text{compose}(f1, f2)$ Given two sets $f1$ and $f2$ of semantic correspondences, $\equiv \text{compose}(f1, f2)\{(x, z) \mid (x, y) \in f1 \wedge (y, z) \in f2\}$. In other words, when $f1$ and $f2$ are construed as morphisms, $f := \text{compose}(f1, f2)$ is the composite morphism from $f1$ and $f2$.

$(m, f1, f2) := \text{merge}(m1, m2, f)$ Informally, this operation aims to retain, in the merged model, the information content of the input models. More precisely, given two models $m1$ and $m2$ and the set f of semantic correspondences that are postulated to hold between them, this operation derives a new model m such that $f1$ is a set of correspondences between m and $m1$ and $f2$ is a set of correspondences between m and $m2$, the assumption being (in [52]) that

m is minimal, $\text{constructsIn}(m) = \text{domain}(f1) \cup \text{domain}(f2)$, $\text{range}(f1) = \text{constructsIn}(m1)$, $\text{range}(f2) = \text{constructsIn}(m2)$, and $f = \text{compose}(\text{invert}(f1), f2)$.

$(m', f') := \text{extract}(m, f)$ Informally, this operation aims to return from the input model m , given its correspondences with another model (that may remain unnamed), that portion m' of m that *participates* in f while making the ensuing adjustments in the input correspondences. More precisely, given a model m and a set f of semantic correspondences, this operation derives a model m' and set of semantic correspondences f' such that [52] m' is minimal, $f = \text{compose}(f', \text{compose}(\text{invert}(f'), f))$, and $\text{constructsIn}(m') = \text{range}(f')$.

$(m', f') := \text{difference}(m, f)$ Conceptually, this operation acts as the complement of `extract`. Informally, this operation aims to return from the input model m , given its correspondences with another model (that may remain unnamed), that portion m' of m that does *not participate* in f while making the ensuing adjustments in the input correspondences. More precisely, given a model m and a set f of semantic correspondences, this operation derives a model m' and set of semantic correspondences f' such that [52] m' is minimal, $(m, ft) = \text{extract}(m, f)$ and $(m, ft, f') = \text{merge}(m, m', \text{compose}(\text{invert}(ft), f'))$.

An extended discussion of the properties stemming from the above semantics of `compose`, `merge`, `extract`, and `difference` can be found in [52].

3.5.6 Dataspace-Specific Operations

$v := \text{viewGen}(f)$ Given a set f of semantic correspondences, it is possible to derive, algorithmically, the mappings v that the correspondences, conceptually speaking, encode. As mentioned above, in [47] we have shown how `viewGen` can, in fact, generate executable expressions (in our case, view expressions against the supermodel we use). In the more general case, the operation is expected to select from the set of semantic correspondences those that are available for use by `evalIQ` in generating queries for `evalQ`, as described above.

$u := \text{gather}(r)$ This operation is construed as providing the means by which a set u of feedback instances can be gathered. Incremental improvement based on user feedback can take a variety of forms: through the manual provision of mappings (e.g., [64]); through the annotation of query results as to which items are spurious or which should be ranked higher (e.g., [62]); through a more intensively interactive approach requiring a fair amount of user input during the integration process (e.g., [37]), or through a process by which mappings are debugged (e.g., [50]). We observe that all these approaches require, to different degrees, an understanding of the syntax and semantics of mapping and schema languages on the part of the person providing the feedback. This has the drawback that only experts can provide feedback, shutting out casual, non-expert users from the process. This is one reason why, in this formulation, feedback is gathered on query results r (which is the case we have explored in our own work [6]), as this seems to require less expertise and more closely taps into the conceptions and

expectations that users have of the dataspace content. In our case, we ask users to indicate which tuples are true positives (i.e., are in the result and should have been, denoted by *TP*), false positives (i.e., are in the result but should not have been, denoted by *FP*) and false negatives (i.e., are not in the result but should have been, denoted by *FN*). The effort in doing this is, so to speak, what the user pays. The operations below illustrate the use that can be made of this kind of feedback and hence the payback for the user in providing it. It is possible, as we have discussed in [32], to have different kinds of annotation for which annotation, selection, refinement and propagation would require different algorithms, techniques and strategies which are, for the most part, yet to be pursued in the literature.

$v := \text{annotate}(u)$ Given a set u of feedback instances on results r computed using a mapping v , if the instances characterize subsets *TP*, *FP* and *FN* of r , then we can annotate v with estimates of its `precision` and `recall` as shown in [6]. The idea in this case is to learn from the feedback given which, among alternative mappings that could be used to answer a query, have the best precision-recall trade-off. Over time, these annotations enable the system to discriminate between mappings that produce good results from those that do not. This is crucial in the dataspace context because of the reliance on automation to derive associations, correspondences and mappings (i.e., the morphisms that give rise to mediated models over independent data sources). The next operations seek to make use of these annotations on mappings.

$v' := \text{select}(q, v)$ Given a query q and set of (presumably annotated) alternative mappings v which could be used for evaluating q , we can select the set v' of mappings that are estimated to produce better quality results (e.g., results to which users assign a better precision-recall trade-off). Our work [6] has provided evidence that, with relatively small amounts of effort in the provision of *TP*, *FP*, and *FN* feedback, it is possible to select mappings with a good precision-recall trade-off for a query. However, this assumes that such mappings are already available in the first place, i.e., that the automated generation of mappings succeeded in generating good initial ones. The next operation seeks to cater for the possibility that this may not happen.

$v' := \text{refine}(v)$ Given a set of (presumably annotated) mappings v , we can use their estimated precision and recall to guide a search process over the space of refined mappings, i.e., mappings that are derivable from v , in order to yield a set of mappings v' with better estimates than v . Our work [6] describes the transformations on mappings that generate the space of mappings and an evolutionary algorithm that searches that space. The experimental evidence indicates that this approach is effective in making the most of the feedback (in line with the pay-as-you-go philosophy) to compensate for the potential shortcomings of the pervasive use of automation to bootstrap dataspace.

$v' := \text{propagateToV}(v); q' := \text{propagateToQ}(v, q)$ Given a set of (presumably annotated) mappings v , we can use their estimated precision and recall to annotate a set of mappings v' with estimates of precision and recall, provided that v and v' stand in some relationship (e.g., v' uses v). As a variant operation with

a similar intent, given a set of (presumably annotated) mappings ν , we can use their estimated precision and recall to annotate a query with an estimate of the precision-recall trade-off that its results would exhibit. These two operations also seek to make the most of feedback. They enable, in the cases described in [6], the propagation of estimates from mappings to mappings and from mappings to queries. This was shown, once again, to compensate, to a cost-effective extent for shortcoming in the bootstrapping phase.

This section has formulated a functional model of dataspace using an algebraic approach that builds on, complements, and extends previous work, particularly in the model management literature. The next section shows how the resulting many-sorted algebra can express frequently occurring use cases and scenarios of great practical interest.

4 Examples and Use Case

In this section we illustrate how the functional model in Sec. 3 can support the dataspace life cycle. Thus, we show in Sec. 4.1 how, in our functional model, one can initialize a dataspace (in our illustration, we use three data sources) and bring it to the point in which it can be used for querying. Then, in Sec. 4.2, we show how the functional model supports dataspace maintenance and, in Sec. 4.3, dataspace improvement. We illustrate this, firstly, by capturing the consequences of a change in the intensional description of a data source after which the dataspace is, again, ready for querying, and, secondly, by capturing the consequences of the inclusion of an additional data source in the dataspace. Finally, in Sec. 4.4, we show that these tasks (viz., initialization, maintenance and improvement) are typical of important practical scenarios for dataspace use. We illustrate this with a use case from bioinformatics.

4.1 Example: Dataspace Initialization

In this section we present, in Fig. 7, an example of how a dataspace may come into being by bootstrapping, i.e., by recourse to automation of the matching and merging of data sources and of mapping generation to mediate the interactions between the merged model and that of each data source.

The example can be seen as illustrating how the core functionality of a traditional DIS can be captured in our model of DSMSs. Note that while a traditional approach to integration would be more effective (i.e., deliver high quality results) from start, it would also incur higher costs due to human involvement throughout the set up of the DIS, making this route the slower and more expensive in terms of reaching the stage in which queries can be posed against the mediated model.

We refrain from commenting on the script in Fig. 7 (and all other similar figures in this section) in the text and, instead, prefer to intersperse them between statements in the script itself.

```

1: /* Discover, identify, rank and choose three data sources out of a set of data sources  $D$ . */
2:  $\{d_1, \dots, d_n\} := \text{discover}(D)$ 
3:  $\{d_1, \dots, d_m\} := \text{identify}(\{d_1, \dots, d_n\})$  /* with  $3 \leq m \leq n$ . */
4:  $\{d_1, \dots, d_m\} := \text{rank}(\{d_1, \dots, d_m\})$  /* with  $\{d_1, \dots, d_m\}$  potentially being in a different order. */
5:  $\{d_1, d_2, d_3\} := \text{choose}(\{d_1, \dots, d_m\})$ 
6: /* Obtain the models from the data sources and sample them. */
7: for  $i \in [1..3]$  do
8:    $m_i := \text{obtain}(d_i)$ 
9:    $d_{si} := \text{sample}(d_i)$ 
10: end for
11: /* Match the obtained models and generate the correspondences needed to derive merged models. */
12:  $a_{1-2} := \text{match}(d_{s1}, d_{s2}, m_1, m_2)$ 
13:  $a_{1-3} := \text{match}(d_{s1}, d_{s3}, m_1, m_3)$ 
14:  $f_{1-2} := \text{inferCorrespondences}(a_{1-2})$ 
15:  $f_{1-3} := \text{inferCorrespondences}(a_{1-3})$ 
16: /* Derive the merged models and generate the correspondences between them and the existing models,
    composing the correspondences as needed to map the merged model over all data sources onto the
    latter. */
17:  $(m_{12}, f_{m_{12}-1}, f_{m_{12}-2}) := \text{merge}(m_1, m_2, f_{1-2})$ 
18:  $f_{m_{12}-3} := \text{compose}(f_{m_{12}-1}, f_{1-3})$ 
19:  $(m_{123}, f_{m_{123}-m_{12}}, f_{m_{123}-3}) := \text{merge}(m_{12}, m_3, f_{m_{12}-3})$ 
20:  $f_{m_{123}-1} := \text{compose}(f_{m_{123}-m_{12}}, f_{m_{12}-1})$ 
21:  $f_{m_{123}-2} := \text{compose}(f_{m_{123}-m_{12}}, f_{m_{12}-2})$ 
22: /* Generate the mappings needed to evaluate queries against the merged model over the data sources
    in the dataspace. */
23: for  $i \in [1..3]$  do
24:    $v_{m_{123}-i} := \text{viewGen}(f_{m_{123}-i})$ 
25: end for
26: /* The bootstrapping is complete. */
27: /* We can now evaluate a query  $q$  against the merged model using the generated mappings. */
28:  $r := \text{evalIQ}(q, m_{123}, \{v_{m_{123}-1}, v_{m_{123}-2}, v_{m_{123}-3}\})$ 

```

Fig. 7 Bootstrapping example based on traditional data integration using views to query against a mediated schema over multiple data sources

Fig. 7 illustrates how dataspace can support the automated bootstrapping of an artefact which in a traditional data integration would be carefully crafted by human experts.

4.2 Example: Dataspace Maintenance

Automated bootstrapping of the kind illustrated in Sec. 4.1, can significantly reduce upfront costs. These are not the only high costs incurred in traditional data integration. Another cause for human intervention in traditional data integration scenarios

is the need to respond to changes in the data sources, which is particularly problematic when the latter are autonomous (i.e., outside the managerial control of the stakeholder of the integrated resource). There are, among others, the issues of timeliness (i.e., how quickly can changes in the sources be reflected in the models that integrate them) and of cost (insofar as there is a need to resort to human expertise to propagate the changes).

The example in Fig. 8 illustrates how dataspace aim to tackle this issue by automating the propagation of changes. It assumes that the three data sources $\{d_1, d_2, d_3\}$ have been integrated as described in Fig. 7.

4.3 Example: Dataspace Improvement

We note that, largely relying on the model-management operations, this conception of dataspace provides the basis for supporting more than mediated schema, e.g., alternative ones, and, in the scope of each mediated schema, the coexistence of alternative mappings into the data sources. This pervasive use of automation and this support for alternative models and for alternative mappings for a given model is likely to result on results of lesser quality in comparison to traditional data integration.

The example in Fig. 9 illustrates how dataspace aim to tackle this issue by procuring and responding to feedback. It uses the dataspace created in the example in Fig. 8.

4.4 A Bioinformatics Use Case

Assume that a group of immunologists is studying malaria. They have decided to use the mouse as a model organism and have carried out a variety of transcriptomics and proteomics experiments whose results have been deposited in local copies of, resp., Array Express [57], a database of gene expression and other microarray data, and Pride [65], a standards-compliant repository for proteomics data. In order to support the analysis of the data they obtained, various databases need to be integrated. The immunologists choose to bootstrap a dataspace by including in it the following data sources: Genbank [7], which stores the genetic sequences of a large number of organisms; MGD [13], the Mouse Genome Database; Gene Ontology [63], which is a knowledge base on the function of genes; and KEGG [39], a pathway database that is conjectured to contain relevant information on pathways additional to those in MGD.

Figure 10 shows the bootstrapping of this dataspace and is analogous to the example in Fig. 7 in that it shows how our functional model captures such tasks as discovering, identifying, ranking and choosing the data sources, then creating the dataspace over the chosen sources by generating a merged integration schema to the

```

1: /* Assume that  $d_1$  has changed but we have its previous model  $m_1$  and a previous sample  $s_1$  from it. */
2: /* Obtain the new, changed model of  $d_1$  and sample the latter again. */
3:  $m_{1'}$  := obtain( $d_1$ )
4:  $d_{s_{1'}}$  := sample( $d_1$ )
5: /* Match the new model and the old, generate correspondences over the resulting associations, identify the differences between the new model and the old, and, through composition, generate the new correspondences needed to derive the new merged model. */
6:  $a_{1-1'}$  := match( $d_{s_1}$ ,  $d_{s_{1'}}$ ,  $m_1$ ,  $m_{1'}$ )
7:  $f_{1-1'}$  := inferCorrespondences( $a_{1-1'}$ )
8: ( $m'_{1'}$ ,  $f'_{1-1'}$ ) := difference( $m_1$ ,  $f_{1-1'}$ )
9:  $f_{m_{123}-1'}$  := compose( $f_{m_{123}-1}$ ,  $f'_{1-1'}$ )
10: /* Derive the new merged model from the previous merged model and the changed model  $m'_{1'}$ , then, through composition, generate the correspondences between the new merged model and the unchanged data sources. */
11: ( $m_{1'23}$ ,  $f_{m_{1'23}-m_{123}}$ ,  $f_{m_{1'23}-1'}$ ) := merge( $m_{123}$ ,  $m'_{1'}$ ,  $f_{m_{123}-1'}$ )
12:  $f_{m_{1'23}-2}$  := compose( $f_{m_{1'23}-m_{123}}$ ,  $f_{m_{123}-2}$ )
13:  $f_{m_{1'23}-3}$  := compose( $f_{m_{1'23}-m_{123}}$ ,  $f_{m_{123}-3}$ )
14: /* Generate the mappings needed to evaluate queries against the new merged model over the changed source(s) in the dataspace. */
15: for  $i \in [1', 2, 3]$  do
16:    $v_{m_{1'23}-i}$  := viewGen( $f_{m_{1'23}-i}$ )
17: end for
18: /* This maintenance action is complete. */
19: /* We can now evaluate a query  $q$  against the new merged model using the generated mappings. */
20:  $r$  := evalIQ( $q$ ,  $m_{1'23}$ , { $v_{m_{1'23}-1'}$ ,  $v_{m_{1'23}-2}$ ,  $v_{m_{1'23}-3}$ })
21: /* Assume now that a new data source  $d_4$  has been added to the dataspace. */
22: /* Obtain its model and sample it. */
23:  $m_4$  := obtain( $d_4$ )
24:  $d_{s_4}$  := sample( $d_4$ )
25: /* Match the new source and its model with a similar existing data source if one exists, instead of with the existing merged model, in order to benefit from instance-level matching, then, through composition, generate the correspondences required to merge the incoming model with the existing merged model. */
26:  $a_{2-4}$  := match( $d_{s_2}$ ,  $d_{s_4}$ ,  $m_2$ ,  $m_4$ ) /* Assuming  $d_2 \sim d_4$ . */
27:  $f_{2-4}$  := inferCorrespondences( $a_{2-4}$ )
28:  $f_{m_{1'23}-4}$  := compose( $f_{m_{1'23}-2}$ ,  $f_{2-4}$ )
29: /* Derive the new merged model to include the incoming model, then, through composition, generate the correspondences between the new merged model and the preexisting data sources. */
30: ( $m_{1'234}$ ,  $f_{m_{1'234}-m_{1'23}}$ ,  $f_{m_{1'234}-4}$ ) := merge( $m_{1'23}$ ,  $m_4$ ,  $f_{m_{1'23}-4}$ )
31: for  $i \in [1', 2, 3]$  do
32:    $f_{m_{1'234}-i}$  := compose( $f_{m_{1'234}-m_{1'23}}$ ,  $f_{m_{1'23}-i}$ )
33: end for
34: /* Generate the mappings needed to evaluate queries against the new merged model over the enlarged collection of sources in the dataspace. */
35: for  $i \in [1', 2..4]$  do
36:    $v_{m_{1'234}-i}$  := viewGen( $f_{m_{1'234}-i}$ )
37: end for
38: /* This maintenance action is complete. */
39: /* We can now evaluate a query  $q$  against the new merged model using the generated mappings. */
40:  $r'$  := evalIQ( $q$ ,  $m_{1'234}$ , { $v_{m_{1'234}-1'}$ ,  $v_{m_{1'234}-2}$ ,  $v_{m_{1'234}-3}$ ,  $v_{m_{1'234}-4}$ })

```

Fig. 8 Maintenance example based on evolving a mediated schema in response to changes to the schema m_1 of data source d_1 and, then, on the addition of a new data source d_4

```

1: /* Assume, following on from Fig. 8 that alternative mappings to those initially generated have been
   made available. For the purposes of this example, we assume that alternative mappings use  $\mu$  rather
   than  $\nu$  in the naming scheme we are using. We now wish to pursue opportunities for improvement. */
2: /* We start by gathering user feedback on the result  $r'$  in Fig. 8, then using it to annotate the available
   mappings and to select those to be used in the next evaluation of queries (and possibly future ones). */
3:  $u := \text{gather}(r')$ 
4:  $\{v_{m_1'234-1}^a, v_{m_1'234-2}^a, v_{m_1'234-3}^a, v_{m_1'234-4}^a, \mu_{m_1'234-2}^a, \mu_{m_1'234-3}^a\} := \text{annotate}(u)$ 
5:  $\{v_{m_1'234-1}^a, \mu_{m_1'234-2}^a, v_{m_1'234-3}^a, v_{m_1'234-4}^a\} :=$ 
6:    $\text{select}(q, \{v_{m_1'234-1}^a, v_{m_1'234-2}^a, v_{m_1'234-3}^a, v_{m_1'234-4}^a, \mu_{m_1'234-2}^a, \mu_{m_1'234-3}^a\})$ 
7: /* Note that the alternative mapping  $\mu_{m_1'234-2}^a$  was selected over  $v_{m_1'234-2}^a$ , the initially generated one, on
   the grounds that it has been annotated with better quality estimates. */
8: /* For the purposes of informing users (for one example), we can also propagate the quality estimates
   for the selected, annotated mappings to any query that uses them. */
9:  $q^a := \text{propagateToQ}(\{v_{m_1'234-1}^a, \mu_{m_1'234-2}^a, v_{m_1'234-3}^a, v_{m_1'234-4}^a\}, q)$ 
10: /* Now, when we reevaluate the query  $q$ , we can use the selected mappings. */
11:  $r'' := \text{evalIQ}(q, m_1'234,$ 
12:    $\{v_{m_1'234-1}^a, \mu_{m_1'234-2}^a, v_{m_1'234-3}^a, v_{m_1'234-4}^a\})$ 
13: /* These gather-annotate-select steps can be repeated as many times as necessary in order to select
   the best set of mappings. */
14: /* Assume we still want to pursue further opportunities for improvement. */
15: /* We can gather more user feedback but now on the latest query result  $r''$ . */
16:  $u := \text{gather}(r'')$ 
17: /* We can then try to refine the mappings in light of the latest quality estimates. */
18:  $\{v_{m_1'234-1}^r, \mu_{m_1'234-2}^r, v_{m_1'234-3}^r, v_{m_1'234-4}^r\} := \text{refine}(\{v_{m_1'234-1}^a, \mu_{m_1'234-2}^a, v_{m_1'234-3}^a, v_{m_1'234-4}^a\})$ 
19: /* Now, when we reevaluate the query  $q$ , we can use the refined mappings. */
20:  $r''' := \text{evalIQ}(q, m_1'234,$ 
21:    $\{v_{m_1'234-1}^r, \mu_{m_1'234-2}^r, v_{m_1'234-3}^r, v_{m_1'234-4}^r\})$ 
22: /* Of course, selection and refinement steps can be interleaved as needed. */

```

Fig. 9 Improvement by gathering feedback on query results then using it for mapping selection and refinement

point in which it is possible to evaluate a query against this schema that resolves into queries over the sources in the dataspace.

Running a set of initial queries shows that the results contain tuples that the immunologists working on mouse are not interested in. In particular, since Genbank and KEGG contain information not just on mouse but on a variety of other organisms, the queries tend to return tuples containing information on those other organisms too.

The scientists decide to spend some effort on improving the initial integration by re-running a set of queries and providing feedback on the results, indicating which result tuples they expect to see and which they do not expect to see in the result. The feedback provided is then used to annotate the mappings that were used to evaluate the queries. This underpins the selection of the mappings to be used in future query evaluations and has effects of increasing the likelihood that mappings that have lower quality (e.g., worse precision-recall trade-offs) are excluded.

However, another run of the queries with the new set of mappings shows that more improvements can possibly be made. The scientists try refining the mappings

```

1: /* The data sources chosen are the local copy of ArrayExpress, AEloc; the local copy of Pride, PRloc;
   Genbank, denoted by GB; MGD, and KEGG. */
2:  $D := \{d_{AEloc}, d_{PRloc}, d_{GB}, d_{MGD}, d_{KEGG}\}$ 
3: /* Obtain the models from the data sources and sample them. */
4:  $\{m_{AEloc}, m_{PRloc}, m_{GB}, m_{MGD}, m_{KEGG}\} := \text{obtain}(d_{AEloc}) \cup$ 
5:    $\text{obtain}(d_{PRloc}) \cup \text{obtain}(d_{GB}) \cup \text{obtain}(d_{MGD}) \cup \text{obtain}(d_{KEGG})$ 
6:  $\{d_{s_{AEloc}}, d_{s_{PRloc}}, d_{s_{GB}}, d_{s_{MGD}}, d_{s_{KEGG}}\} := \text{sample}(d_{AEloc}) \cup$ 
7:    $\text{sample}(d_{PRloc}) \cup \text{sample}(d_{GB}) \cup \text{sample}(d_{MGD}) \cup \text{sample}(d_{KEGG})$ 
8: /* Match the obtained models and generate the correspondences needed to derive merged models. */
9:  $a_{AEloc-PRloc} := \text{match}(d_{s_{AEloc}}, d_{s_{PRloc}}, m_{AEloc}, m_{PRloc})$ 
10:  $a_{AEloc-MGD} := \text{match}(d_{s_{AEloc}}, d_{s_{MGD}}, m_{AEloc}, m_{MGD})$ 
11:  $a_{GB-MGD} := \text{match}(d_{s_{GB}}, d_{s_{MGD}}, m_{GB}, m_{MGD})$ 
12:  $a_{MGD-KEGG} := \text{match}(d_{s_{MGD}}, d_{s_{KEGG}}, m_{MGD}, m_{KEGG})$ 
13:  $f_{AEloc-PRloc} := \text{inferCorrespondences}(a_{AEloc-PRloc})$ 
14:  $f_{AEloc-MGD} := \text{inferCorrespondences}(a_{AEloc-MGD})$ 
15:  $f_{GB-MGD} := \text{inferCorrespondences}(a_{GB-MGD})$ 
16:  $f_{MGD-KEGG} := \text{inferCorrespondences}(a_{MGD-KEGG})$ 
17: /* Derive the comprehensive merged model by firstly creating a merged model of the data sources with
   the experimental data sources, then a merged model of Genbank, MGD and KEGG, then merging the
   two resulting models. Then, generate the correspondences between models, composing the correspon-
   dences as needed to map the merged model over all data sources onto the latter. */
18:  $(m_{Exp}, f_{Exp-AEloc}, f_{Exp-PRloc}) := \text{merge}(m_{AEloc}, m_{PRloc}, f_{AEloc-PRloc})$ 
19:  $(m_{Genetic}, f_{Genetic-GB}, f_{Genetic-MGD}) := \text{merge}(m_{GB}, m_{MGD}, f_{GB-MGD})$ 
20:  $f_{Genetic-KEGG} := \text{compose}(f_{Genetic-MGD}, f_{MGD-KEGG})$ 
21:  $(m_{GenKEGG}, f_{GenKEGG-Genetic}, f_{GenKEGG-KEGG}) := \text{merge}(m_{Genetic}, m_{KEGG}, f_{Genetic-KEGG})$ 
22:  $f_{MGD-GenKEGG} := \text{compose}(\text{invert}(f_{Genetic-MGD}), \text{invert}(f_{GenKEGG-Genetic}))$ 
23:  $f_{Exp-GenKEGG} := \text{compose}(\text{compose}(f_{Exp-AEloc}, f_{AEloc-MGD}), f_{MGD-GenKEGG})$ 
24:  $(m_{Mouse1}, f_{Mouse1-Exp}, f_{Mouse1-GenKEGG}) := \text{merge}(m_{Exp}, m_{GenKEGG}, f_{Exp-GenKEGG})$ 
25:  $f_{Mouse1-AEloc} := \text{compose}(f_{Mouse1-Exp}, f_{Exp-AEloc})$ 
26:  $f_{Mouse1-PRloc} := \text{compose}(f_{Mouse1-Exp}, f_{Exp-PRloc})$ 
27:  $f_{Mouse1-GB} := \text{compose}(\text{compose}(f_{Mouse1-GenKEGG}, f_{GenKEGG-Genetic}), f_{Genetic-GB})$ 
28:  $f_{Mouse1-MGD} := \text{compose}(\text{compose}(f_{Mouse1-GenKEGG}, f_{GenKEGG-Genetic}), f_{Genetic-MGD})$ 
29:  $f_{Mouse1-KEGG} := \text{compose}(f_{Mouse1-GenKEGG}, f_{GenKEGG-KEGG})$ 
30: /* Generate the mappings needed to evaluate queries against the merged model over the data sources
   in the dataspace. */
31:  $v_{Mouse1} := \text{viewGen}(f_{Mouse1-AEloc}) \cup \text{viewGen}(f_{Mouse1-PRloc}) \cup$ 
32:    $\text{viewGen}(f_{Mouse1-GB}) \cup \text{viewGen}(f_{Mouse1-MGD}) \cup \text{viewGen}(f_{Mouse1-KEGG})$ 
33: /* The bootstrapping is complete. */
34: /* We can now evaluate a query  $q$  against the merged model using the generated mappings. */
35:  $r1 := \text{evalIQ}(q, m_{Mouse1}, v_{Mouse1})$ 

```

Fig. 10 Bioinformatics use case: Bootstrapping a dataspace with Genbank, MGD, KEGG and local copies of ArrayExpress and Pride

with the aim of obtaining new mappings that better reflect the expectations of the immunologists. These steps are shown in Figure 11, where, for simplicity, the improvement is applied to one query only. This example is analogous to the one in Fig. 9.

After a period in which they analyse the experimental data using the refined dataspace, the scientists would like to know how the information gathered and the knowledge obtained transfers across to humans. Therefore, they decide to add

```

1: /* Gather user feedback on the query result r1, use it to annotate the available mappings and to select
   those to be used from now on. */
2: u1 := gather(r1)
3: vaMouse1 := annotate(u1)
4: vaMouse1 := select(q, vaMouse1)
5: /* Propagate the quality estimates for the selected, annotated mapping to any query that uses them. */
6: qa := propagateToQ(vaMouse1, q)
7: /* Evaluate q again, now with the selected set of mappings. */
8: r2 := evalIQ(q, mMouse1, vaMouse1)
9: /* These gather-annotate-select steps can be repeated as many times as necessary in order to select
   the best set of mappings. */
10: /* Gather user feedback on the latest query result, say, r2. */
11: u2 := gather(r2)
12: /* Try refining the mappings in light of the latest quality estimates. */
13: vrMouse1 := refine(vaMouse1)
14: /* Evaluate q again, now with the refined mappings. */
15: r3 := evalIQ(q, mMouse1, vrMouse1)
16: /* Of course, selection and refinement steps can be interleaved as needed. */

```

Fig. 11 Bioinformatics use case: Improvement by gathering feedback on query results then using it for mapping selection and refinement

two more sources to their dataspace, viz., OMIM (Online Mendelian Inheritance in Man) [49], a database containing information on human genes and phenotypes, and Ensembl [25], a software system and data resource that produces, provides and maintains automatic annotation on selected eukaryotic genomes, including both mouse and human.

The steps involved in adding these two new sources are shown in Figure 12. Note that, for evaluating queries against the extended merged schema, the scientists derive new mappings by composing them with the original mappings rather than the refined ones, as the refinement aimed at including only information on the mouse, but no information on human.

The scientists then decide that they would like to study some additional experimental data made public by other researchers working on malaria in mouse as well as in human. This requires the integration of the public repositories of ArrayExpress and Pride. As the schemas of the local copies of ArrayExpress and Pride used for bootstrapping in Fig. 10 may have evolved since the scientists installed their own local copy, the schemas need to be compared first and differences identified.

The steps involved in integrating the public versions of ArrayExpress and Pride in the dataspace and generating merged schemas for both mouse and the study across mouse and human are shown in Figure 13.

5 Conclusions and Future Work

In this work, we have built on our recent research on dataspaces [6, 32, 33, 47] and have aimed to present in some detail:

```

1: /* Two additional data sources, viz., OMIM and Ensembl, denoted by ES are to be added to the dataspac. */
2: /* Obtain the models from the new data sources and sample them. */
3: {mOMIM, mES} := obtain(dOMIM) ∪ obtain(dES)
4: {dsOMIM, dsES} := sample(dOMIM) ∪ sample(dES)
5: /* Match the obtained models with each other and with MGD, which was part of the previous integration and contains overlapping information with Ensembl. Then, generate the correspondences needed to derive new merged models. */
6: aOMIM-ES := match(dsOMIM, dsES, mOMIM, mES)
7: aMGD-ES := match(dsMGD, dsES, mMGD, mES)
8: fOMIM-ES := inferCorrespondences(aOMIM-ES)
9: fMGD-ES := inferCorrespondences(aMGD-ES)
10: /* Derive the new merged model to include the incoming models, then, through composition, generate the correspondences between the new merged model and the preexisting data sources. */
11: (mHuman1, fHuman1-OMIM, fHuman1-ES) := merge(mOMIM, mES, fOMIM-ES)
12: fMouse1-Human1 := compose(compose(fMouse1-MGD, fMGD-ES), invert(fHuman1-ES))
13: (mMH1, fMH1-Human1, fMH1-Mouse1) := merge(mMouse1, mHuman1, fMouse1-Human1)
14: fMH1-AEloc := compose(fMH1-Mouse1, fMouse1-AEloc)
15: fMH1-PRloc := compose(fMH1-Mouse1, fMouse1-PRloc)
16: fMH1-GB := compose(fMH1-Mouse1, fMouse1-GB)
17: fMH1-MGD := compose(fMH1-Mouse1, fMouse1-MGD)
18: fMH1-KEGG := compose(fMH1-Mouse1, fMouse1-KEGG)
19: fMH1-OMIM := compose(fMH1-Human1, fHuman1-OMIM)
20: fMH1-ES := compose(fMH1-Human1, fHuman1-ES)
21: /* Generate the mappings needed to evaluate queries against the new merged model over the enlarged collection of sources in the dataspace. */
22: vMH1 := viewGen(fMH1-AEloc) ∪ viewGen(fMH1-PRloc) ∪ viewGen(fMH1-GB) ∪
23:           viewGen(fMH1-MGD) ∪ viewGen(fMH1-KEGG) ∪ viewGen(fMH1-OMIM) ∪
24:           viewGen(fMH1-ES)
25: /* This maintenance action is complete. */
26: /* We can now evaluate a query q against the new merged model using the generated mappings. */
27: r1 := evalIQ(q, MH1, vMH1)

```

Fig. 12 Bioinformatics use case: Maintenance by evolving the mediated schema in response to changes to the addition of OMIM and Ensembl to the dataspace

1. a view of dataspace management systems as second-generation data integration platforms in which pervasive automation is deployed to push down costs and user feedback is gathered and used to increase result quality;
2. a view of the life cycle of a dataspace that captures, albeit coarsely, the functionality of a growing body of literature (which we have most recently surveyed in [33]), is consistent with (1), and comprising initialization, use, maintenance and improvement stages;
3. a conceptualization of the functional architecture of dataspace management systems that is consistent with (1) and explains more precisely than has been done so far in what ways they are related to classical database, data integration and model management systems;
4. a formalization of the functional model for dataspace management systems as a many-sorted algebra that is consistent with (3), and, while building on a history of advances by data integration and model management researchers, gives

```

1: /* Two additional data sources, viz., the public versions of ArrayExpress, denoted by AEpub, and of Pride,
   denoted by PRpub are added to the dataspace. */
2: /* Obtain the models from the new data sources and sample them. */
3: {mAEpub, mPRpub} := obtain(dAEpub) ∪ obtain(dPRpub)
4: {dsAEpub, dsPRpub} := sample(dAEpub) ∪ sample(dPRpub)
5: /* Match the models of the public versions of ArrayExpress and Pride with the local versions, generate the
   correspondences and identify the differences, and, through composition, generate the correspondences
   needed to create a new merged model for mouse and another one for mouse and human. */
6: aAEloc-AEpub := match(dsAEloc, dsAEpub, mAEloc, mAEpub)
7: aPRloc-PRpub := match(dsPRloc, dsPRpub, mPRloc, mPRpub)
8: fAEloc-AEpub := inferCorrespondences(aAEloc-AEpub)
9: fPRloc-PRpub := inferCorrespondences(aPRloc-PRpub)
10: (mAEpubLocDiff, fAEpub-AEloc-Diff) := difference(mAEpub, invert(fAEloc-AEpub))
11: (mPRpubLocDiff, fPRpub-PRloc-Diff) := difference(mPRpub, invert(fPRloc-PRpub))
12: fMouse1-AEpubLocDiff := compose(fMouse1-AEloc, invert(fAEpub-AEloc-Diff))
13: fMouse1-PRpubLocDiff := compose(fMouse1-PRloc, invert(fPRpub-PRloc-Diff))
14: fMH1-AEpubLocDiff := compose(fMH1-AEloc, invert(fAEpub-AEloc-Diff))
15: fMH1-PRpubLocDiff := compose(fMH1-PRloc, invert(fPRpub-PRloc-Diff))
16: /* Derive the new merged models for both mouse and the study across mouse and human from the
   previously merged models and the parts of the public versions of ArrayExpress and Pride that are
   different from the local versions and generate the correspondences between the new merged models
   and the integrated data sources. */
17: (mMouse1.5, fMouse1.5-Mouse1, fMouse1.5-AEpubLocDiff) := merge(mMouse1, mAEpubLocDiff,
   fMouse1-AEpubLocDiff)
18: fMouse1.5-PRpubLocDiff := compose(fMouse1.5-Mouse1, fMouse1-PRpubLocDiff)
19: (mMouse2, fMouse2-Mouse1.5, fMouse2-PRpubLocDiff) := merge(mMouse1.5, mPRpubLocDiff,
   fMouse1.5-PRpubLocDiff)
20: fMouse2-Mouse1 := compose(fMouse2-Mouse1.5, fMouse1.5-Mouse1)
21: fMouse2-AEloc := compose(fMouse2-Mouse1, fMouse1-AEloc)
22: fMouse2-PRloc := compose(fMouse2-Mouse1, fMouse1-PRloc)
23: fMouse2-GB := compose(fMouse2-Mouse1, fMouse1-GB)
24: fMouse2-MGD := compose(fMouse2-Mouse1, fMouse1-MGD)
25: fMouse2-KEGG := compose(fMouse2-Mouse1, fMouse1-KEGG)
26: fMouse2-AEpubLocDiff := compose(fMouse2-Mouse1.5, fMouse1.5-AEpubLocDiff)
27: (mMH1.5, fMH1.5-MH1, fMH1.5-AEpubLocDiff) :=
28:   merge(mMH1, mAEpubLocDiff, fMH1-AEpubLocDiff)
29: fMH1.5-PRpubLocDiff := compose(fMH1.5-MH1, fMH1-PRpubLocDiff)
30: (mMH2, fMH2-MH1.5, fMH2-PRpubLocDiff) :=
31:   merge(mMH1.5, mPRpubLocDiff, fMH1.5-PRpubLocDiff)
32: fMH2-MH1 := compose(fMH2-MH1.5, fMH1.5-MH1)
33: fMH2-AEloc := compose(fMH2-MH1, fMH1-AEloc)
34: fMH2-PRloc := compose(fMH2-MH1, fMH1-PRloc)
35: fMH2-GB := compose(fMH2-MH1, fMH1-GB)
36: fMH2-MGD := compose(fMH2-MH1, fMH1-MGD)
37: fMH2-KEGG := compose(fMH2-MH1, fMH1-KEGG)
38: fMH2-OMIM := compose(fMH2-MH1, fMH1-OMIM)
39: fMH2-ES := compose(fMH2-MH1, fMH1-ES)
40: fMH2-AEpubLocDiff := compose(fMH2-MH1.5, fMH1.5-AEpubLocDiff)
41: /* Generate the mappings needed to evaluate queries against the new merged models over the enlarged
   collection of sources in the dataspace. */
42: vMouse2 := viewGen(fMouse2-AEloc) ∪ viewGen(fMouse2-PRloc) ∪
43:   viewGen(fMouse2-GB) ∪ viewGen(fMouse2-MGD) ∪
44:   viewGen(fMouse2-KEGG) ∪ viewGen(fMouse2-AEloc ∪ fMouse2-AEpubLocDiff) ∪
45:   viewGen(fMouse2-PRloc ∪ fMouse2-PRpubLocDiff)
46: vMH2 := viewGen(fMH2-AEloc) ∪ viewGen(fMH2-PRloc) ∪ viewGen(fMH2-GB) ∪
47:   viewGen(fMH2-MGD) ∪ viewGen(fMH2-KEGG) ∪ viewGen(fMH2-OMIM) ∪
48:   viewGen(fMH2-ES) ∪ viewGen(fMH2-AEloc ∪ fMH2-AEpubLocDiff) ∪
49:   viewGen(fMH2-PRloc ∪ fMH2-PRpubLocDiff)
50: /* This maintenance action is complete. */

```

Fig. 13 Bioinformatics use case: Adding the public versions of ArrayExpress and Pride.

crisp contours to the notion of a dataspace, contours that had been hitherto only ambiguously and vaguely drawn;

5. examples of algebraic programs that illustrate how the functional model in (4) can capture and support the various stages of the dataspace life cycle in (2) as well as a more extended use case in bioinformatics that shows how important practical scenarios can be supported.

There is much still to accomplish in dataspace research, even though the literature, as shown here, has grown in breadth and depth and is very much thriving at the time of writing. Among the areas for further work in which the contribution we have made in this work may play a helpful role, we identify (without in any way aiming to be exhaustive):

1. refining and completing the formalization;
2. studying the mathematical and computational properties of the algebra;
3. proposing concrete algorithms for the algebraic operations;
4. implementing a dataspace management system that can be construed as an engine for the programs our algebra gives rise to;
5. considering rewriting strategies for heuristic optimization based on the mathematical properties of the algebra and cost-based optimization techniques based on the concrete algorithms available;
6. encompassing more forms of use;
7. studying the cost-benefit properties of more forms of feedback;
8. providing a principled treatment of how uncertainty can be quantified in relation to the evidence used and then propagated through operator applications.

Dataspaces are, in many respects, still but a promise, and it is too early to tell whether the belief that they could be effective and efficient in complementing classical database and data integration systems will be vindicated. The work we presented here aims to make the issues and challenges clearer and to constitute a step towards the exploration of this exciting hypothesis.

Acknowledgements This work has been funded by the UK EPSRC under Grant EP/F031092/1. We are grateful for this support.

References

1. Alexe, B., Chiticariu, L., Miller, R.J., Tan, W.C.: Muse: Mapping understanding and design by example. In: ICDE, pp. 10–19. IEEE (2008)
2. Atzeni, P., Bellomarini, L., Bugiotti, F., Gianforme, G.: Mism: A platform for model-independent solutions to model management problems. *J. Data Semantics* **14**, 133–161 (2009)
3. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. *VLDB J.* **17**(6), 1347–1370 (2008)
4. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with coma++. In: F. Özcan (ed.) SIGMOD Conference, pp. 906–908. ACM (2005)
5. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.* **18**(4), 323–364 (1986)

6. Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A.A., Hedeler, C.: Feedback-based annotation, selection and refinement of schema mappings for dataspace. In: I. Manolescu, S. Spaccapietra, J. Teubner, M. Kitsuregawa, A. Léger, F. Naumann, A. Ailamaki, F. Özcan (eds.) *EDBT, ACM International Conference Proceeding Series*, vol. 426, pp. 573–584. ACM (2010)
7. Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Wheeler, D.L.: Genbank. *Nucleic Acids Research* **31**(1), 23–27 (2003). Databases in biology: Genbank
8. Bernstein, P.A., Halevy, A.Y., Pottinger, R.: A vision of management of complex models. *SIGMOD Record* **29**(4), 55–63 (2000)
9. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: C.Y. Chan, B.C. Ooi, A. Zhou (eds.) *SIGMOD Conference*, pp. 1–12. ACM (2007)
10. Blunski, 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)
11. Boyd, M., Kittivoravitkul, S., Lazanitis, C., McBrien, P., Rizopoulos, N.: Automed: A bay data integration system for heterogeneous data sources. In: *CAiSE*, pp. 82–97. Springer (2004)
12. Boyd, M., McBrien, P.: Comparing and transforming between data models via an intermediate hypergraph data model. *J. Data Semantics IV* **3730**, 69–109 (2005)
13. Bult, C., Eppig, J., Kadin, J., Richardson, J., Blake, J., the members of the Mouse Genome Database Group: The mouse genome database (mgd): mouse biology and model systems. *Nucleic Acids Research* **36**(Database issue), D724–8 (2008)
14. Cafarella, M.J., Halevy, A.Y., Khousainova, N.: Data integration for the relational web. *PVLDB* **2**(1), 1090–1101 (2009)
15. Cao, H., Qi, Y., Candan, K.S., Sapino, M.L.: Feedback-driven result ranking and query refinement for exploring semi-structured data collections. In: *EDBT*, pp. 3–14 (2010)
16. Chiticariu, L., Hernández, M.A., Kolaitis, P.G., Popa, L.: Semi-automatic schema integration in clio. In: *VLDB*, pp. 1326–1329 (2007)
17. Das Sarma, A., Dong, X., Halevy, A.: Bootstrapping pay-as-you-go data integration systems. In: *SIGMOD*, pp. 861–874 (2008)
18. Dittrich, J.P., Vaz Salles, M.A.: idm: A unified and versatile data model for personal dataspace management. In: *VLDB*, pp. 367–378 (2006)
19. Do, H.H., Rahm, E.: Coma: a system for flexible combination of schema matching approaches. In: *VLDB*, pp. 610–621 (2002)
20. Doan, A., Halevy, A.Y.: Semantic integration research in the database community: A brief survey. *AI Magazine* **26**(1), 83–94 (2005)
21. 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)
22. Dong, X., Halevy, A.Y.: A platform for personal information management and integration. In: *CIDR*, pp. 119–130 (2005)
23. Dong, X., Halevy, A.Y., Yu, C.: Data integration with uncertainty. In: *VLDB*, pp. 687–698 (2007)
24. Dong, X.L., Halevy, A.Y., Yu, C.: Data integration with uncertainty. *VLDB J.* **18**(2), 469–500 (2009)
25. Flicek, P., Aken, B.L., Ballester, B., Beal, K., Bragin, E., Brent, S., Chen, Y., Clapham, P., Coates, G., Fairley, S., Fitzgerald, S., Fernandez-Banet, J., Gordon, L., Gräf, S., Haider, S., Hammond, M., Howe, K., Jenkinson, A., Johnson, N., Keefe, A.K.D., Keenan, S., Kinsella, R., Kokocinski, F., Koscielny, G., Kulesha, E., Lawson, D., Longden, I., Massingham, T., McLaren, W., Megy, K., Overduin, B., Pritchard, B., Rios, D., Ruffier, M., Schuster, M., Slater, G., Smedley, D., Spudich, G., Tang, Y.A., Trevanion, S., Vilella, A., Vogel, J., White, S., Wilder, S.P., Zadissa, A., Birney, E., Cunningham, F., Dunham, I., Durbin, R., Fernández-Suarez, X.M., Herrero, J., Hubbard, T.J.P., Parker, A., Proctor, G., Smith, J., Searle, S.M.J.: Ensembl’s 10th year. *Nucleic Acids Research* **38**(Database issue), D557–D562 (2010)
26. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspace: a new abstraction for information management. *SIGMOD Record* **34**(4), 27–33 (2005)

27. Haas, L., Lin, E., Roth, M.: Data integration through database federation. *IBM SYSTEMS JOURNAL* **41**(4), 578–596 (2002)
28. Halevy, A.Y.: Answering queries using views: A survey. *The VLDB Journal* **10**(4), 270–294 (2001)
29. Halevy, A.Y., Franklin, M.J., Maier, D.: Principles of dataspace systems. In: S. Vansummeren (ed.) *PODS*, pp. 1–9. ACM (2006)
30. Halevy, A.Y., Rajaraman, A., Ordille, J.J.: Data integration: The teenage years. In: U. Dayal, K.Y. Whang, D.B. Lomet, G. Alonso, G.M. Lohman, M.L. Kersten, S.K. Cha, Y.K. Kim (eds.) *VLDB*, pp. 9–16. ACM (2006)
31. Hedeler, C., Belhajjame, K., Fernandes, A.A.A., Embury, S.M., Paton, N.W.: Dimensions of dataspace. In: A.P. Sexton (ed.) *BNCOD, Lecture Notes in Computer Science*, vol. 5588, pp. 55–66. Springer (2009)
32. Hedeler, C., Belhajjame, K., Mao, L., Paton, N.W., Fernandes, A.A.A., Guo, C., Embury, S.M.: Flexible dataspace management through model management. In: F. Daniel, L.M.L. Delcambre, F. Fotouhi, I. Garrigós, G. Guerrini, J.N. Mazón, M. Mesiti, S. Müller-Feuerstein, J. Trujillo, T.M. Truta, B. Volz, E. Waller, L. Xiong, E. Zimányi (eds.) *EDBT/ICDT Workshops, ACM International Conference Proceeding Series*. ACM (2010)
33. Hedeler, C., Belhajjame, K., Paton, N.W., Campi, A., Fernandes, A.A.A., Embury, S.M.: Dataspace. In: S. Ceri, M. Brambilla (eds.) *SeCO Workshop, Lecture Notes in Computer Science*, vol. 5950, pp. 114–134. Springer (2009)
34. Hernández, M.A., Ho, H., Popa, L., Fuxman, A., Miller, R.J., Fukuda, T., Papotti, P.: Creating nested mappings with clio. In: *ICDE*, pp. 1487–1488 (2007)
35. Howe, B., Maier, D., Rayner, N., Rucker, J.: Quarrying dataspace: Schemaless profiling of unfamiliar information sources. In: *ICDE Workshops*, pp. 270–277 (2008)
36. 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)
37. 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*. www.crdrrdb.org (2009)
38. Jeffery, S.R., Franklin, M.J., Halevy, A.Y.: Pay-as-you-go user feedback for dataspace systems. In: *SIGMOD*, pp. 847–860 (2008)
39. Kanehisa, M., Goto, S., Furumichi, M., Tanabe, M., Hiraoka, M.: Kegg for representation and analysis of molecular networks involving diseases and drugs. *Nucleic Acids Research* **38**(Database issue), D355–D360 (2010)
40. Kensche, D., Quix, C., 0002, X.L., Li, Y., Jarke, M.: Generic schema mappings for composition and query answering. *Data Knowl. Eng.* **68**(7), 599–621 (2009)
41. Kim, W., Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer* **24**(12), 12–18 (1991)
42. Lenzerini, M.: Data integration: A theoretical perspective. In: L. Popa (ed.) *PODS*, pp. 233–246. ACM (2002)
43. Leser, U., Naumann, F.: (almost) hands-off information integration for the life sciences. In: *CIDR*, pp. 131–143 (2005)
44. Liu, J., Dong, X., Halevy, A.: Answering structured queries on unstructured data. In: *WebDB*, pp. 25–30 (2006)
45. Lorenzo, G.D., Hacid, H., Paik, H.Y., Benatallah, B.: Data integration in mashups. *SIGMOD Record* **38**(1), 59–66 (2009)
46. 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)
47. Mao, L., Belhajjame, K., Paton, N.W., Fernandes, A.A.A.: Defining and using schematic correspondences for automatically generating schema mappings. In: P. van Eck, J. Gordijn, R. Wieringa (eds.) *CAiSE, Lecture Notes in Computer Science*, vol. 5565, pp. 79–93. Springer (2009)
48. McCann, R., Shen, W., Doan, A.: Matching schemas in online communities: A web 2.0 approach. In: *ICDE*, pp. 110–119 (2008)

49. McKusick, V.A.: Mendelian inheritance in man and its online version, omim. *Am J Hum Genet.* **80**(4), 588–604 (2007). URL <http://www.ncbi.nlm.nih.gov/omim/>
50. Mecca, G., Papotti, P., Raunich, S., Buoncristiano, M.: Concise and expressive mappings with +spicy. *PVLDB* **2**(2), 1582–1585 (2009)
51. Melnik, S.: Generic Model Management: Concepts and Algorithms, *Lecture Notes in Computer Science*, vol. 2967. Springer (2004)
52. Melnik, S., Bernstein, P.A., Halevy, A., Rahm, E.: A semantics for model management operators. Technical Report MSR-TR-2004-59, Microsoft Research (2004)
53. Melnik, S., Bernstein, P.A., Halevy, A., Rahm, E.: Supporting executable mappings in model management. In: SIGMOD, pp. 167–178 (2005)
54. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: a programming platform for generic model management. In: SIGMOD, pp. 193–204 (2003)
55. Miller, R.J., Haas, L.M., Hernández, M.A.: Schema mapping as query discovery. In: VLDB, pp. 77–88 (2000)
56. 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)
57. Parkinson, H., Sarkans, U., Kolesnikov, N., Abeygunawardena, N., Burdett, T., Dylag, M., Emam, I., Farne, A., Hastings, E., Holloway, E., Kurbatova, N., Lukk, M., Malone, J., Mani, R., Pilicheva, E., Rustici, G., Sharma, A., Williams, E., Adamusiak, T., Brandizi, M., Sklyar, N., Brazma, A.: Arrayexpress update—an archive of microarray and high-throughput sequencing-based functional genomics experiments. *Nucleic Acids research* (2010)
58. Poulouvassilis, A., McBrien, P.: A general formal framework for schema transformation. *Data Knowl. Eng.* **28**(1), 47–71 (1998)
59. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. *VLDB J.* **10**(4), 334–350 (2001)
60. Sarma, A.D., Dong, X.L., Halevy, A.Y.: Data modeling in dataspace support platforms. In: A. Borgida, V.K. Chaudhri, P. Giorgini, E.S.K. Yu (eds.) *Conceptual Modeling: Foundations and Applications*, *Lecture Notes in Computer Science*, vol. 5600, pp. 122–138. Springer (2009)
61. Talukdar, P.P., Ives, Z.G., Pereira, F.: Automatically incorporating new sources in keyword search-based data integration. In: A.K. Elmagarmid, D. Agrawal (eds.) *SIGMOD Conference*, pp. 387–398. ACM (2010)
62. 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)
63. The Gene Ontology Consortium: Gene ontology: tool for the unification of biology. *Nature Genetics* **25**(1), 25–29 (2000). *Databases in biology: Gene Ontology*
64. Vaz Salles, M.A., Dittrich, J.P., Karakashian, S.K., Girard, O.R., Blunski, L.: itrails: Pay-as-you-go information integration in dataspace. In: VLDB, pp. 663–674 (2007)
65. Vizcaíno, J., Côté, R., Reisinger, F., Foster, J., Mueller, M., Rameseder, J., Hermjakob, H., Martens, L.: A guide to the proteomics identifications database proteomics data repository. *Proteomics* **9**(18), 4276–83 (2009)