Flexible Dataspace Management Through Model Management

Cornelia Hedeler, Khalid Belhajjame, Lu Mao, Norman W. Paton,
Alvaro A.A. Fernandes, Chenjuan Guo, and Suzanne M. Embury
School of Computer Science, The University of Manchester
Oxford Road, Manchester M13 9PL, UK
(chedeler,khalidb,maol,norm,alvaro,guoc,embury)@cs.manchester.ac.uk

ABSTRACT

The vision of dataspaces has been articulated as providing various of the benefits of classical data integration but with reduced up-front costs, which, combined with opportunities for incremental refinement, enables a "pay as you go" approach to the data integration problem. However, results that seek to realise the vision tend to make design commitments, often to meet quite specific application assumptions, that are likely to restrict their wider use. Instead of precommitting to a specific solution, we build on research in model management and present a generic framework consisting of a collection of types and operations for dataspace management systems that can be instantiated in various ways. The key extension for dataspaces is the integration of user feedback as annotations to model management constructs, and the development of operations that take account of these annotations. The flexibility of the framework is demonstrated through various case studies that meet differing requirements.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

dataspaces, model management

1. INTRODUCTION

Integrating multiple autonomous and heterogeneous data sources is a challenging task [11]. This stems in significant measure from the fact that the development and maintenance of mappings between schemas has proved to be labour intensive. Furthermore, it is often difficult to get the mappings right, due to the frequent occurrence of exceptions and special cases, as well as autonomous changes in the sources that require propagation to the mappings. As a result, classical data integration technology occupies a position at the high initialisation cost, high-quality end of the data access spectrum, and is less effective for numerous, or rapidly changing, resources, or for on-the-fly data integration.

In contrast, the vision of *dataspaces* [10] is that various of the benefits provided by planned, resource-intensive data integration could be obtained at much lower initialisation cost, thereby supporting integration on demand and in changing environments, albeit with a lower initial quality of integration. For this to happen, dataspaces would be expected to use techniques that infer relationships between resources, and that improve these relationships in the light of user or developer feedback. As such, a dataspace can be seen as a data integration system that exhibits the following distinguishing features: (i) low/no initialisation cost, and (ii) support for incremental improvement.

However, to date, no reference framework for the data types and operations that are required to support a generic dataspace management system (DSMS) has emerged. Indeed, the dataspace vision has so far principally given rise to various proposals addressing specific parts of various phases of the dataspace lifecyle [12], such as initialisation (i.e., the initial integration of data sources, e.g., [6]), usage (i.e., querying, e.g., [16]) or the improvement of the dataspace over time utilising user feedback (e.g., [13]). Where complete proposals have been made these either make simplifying assumptions or provide minimal support for at least one of the distinguishing features (e.g. [17, 20, 9]). This paper presents a generic framework, consisting of data types and operations for a DSMS, covering the complete lifecycle of a dataspace, including initialisation, usage, improvement and maintenance.

The integration of autonomous data sources utilising heterogeneous data models requires the manipulation of schemas and of the relationships between schemas, as well as the translation of schemas and data from one data model into a different one. These challenges have been addressed by the model management community in the form of high-level algebraic operations [3, 4]. It has been recognised previously [10] that a DSMS could utilise model management operations, e.g., for schema integration within the initialisation phase, and for schema evolution during the maintenance phase. However, the precise role of existing model management operations, and the additional requirements of dataspace management, have not previously been identified.

The framework presented in this paper builds on existing model management operations (e.g., [3, 4]), extending and supplementing them with additional operations to support more phases of the dataspace life cycle. This approach en-

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

ables us to defer commitment to specific implementations of the operations, and to provide instead a framework from which components can be used selectively and within which it is possible to experiment with a range of specific techniques.

The remainder of the paper is organised as follows: Section 2 introduces the types and operations of the framework, Section 3 instantiates the framework to create dataspaces with varying properties and Section 4 concludes.

2. TYPES AND OPERATIONS

In the following we define the types and introduce the operations that form the framework.

Building on generic data models that subsume those of specific systems (e.g., [1]), we use the term *construct* to refer to one of the following elements of a schema: (i) entities, e.g., tables in relational schemas, classes in object schemas, (ii) attributes, e.g., columns in relational schemas, attributes in object schemas, and (iii) relationships, e.g., foreign keys in relational schemas, associations in object schemas. A set of constructs is denoted as C, or C_{s_i} where the constructs are part of the schema s_i. To represent the relationships between constructs in various schemas at varying levels of abstraction, we introduce the following types:

- A match $mt_{s_i-s_j} = \langle C_{s_i}, C_{s_j} \rangle$ is an association between sets of constructs in two schemas s_i and s_j, as might be identified by a matching algorithm. A set of matches is denoted as $MT_{s_i-s_j}$.
- Associations returned by most matching algorithms, however, tend not to provide enough information for the automatic derivation of mappings. Therefore, we define semantically richer schematic correspondences as follows: $cr_{s_i-s_j} = \langle kind, C_{s_i}, C_{s_j} \rangle$ is a schematic correspondence of a given *kind* between constructs in two schemas s_i and s_j, e.g., describing a *many-to-many* equivalence relationship between constructs [18]. Examples of *kinds* include 'missing attribute', 'name conflict', 'horizontal' or 'vertical partitioning'. A set of schematic correspondences is denoted as $CR_{s_i-s_j}$.
- A mapping $mp_{S_s \to s_i} = \langle q_{s_i}, q_{S_s} \rangle$ is an executable program that specifies how data structured under a set of source schemas can be combined into data structured according to an integration schema, where q_{s_i} is a query over an integration schema s_i and q_{S_s} is a query of the same arity over a set of source schemas S_s. A set of mappings is denoted as $MP_{S_s \to s_i}$ and specifies that the concepts represented by the two queries q_{s_i} and q_{S_s} are semantically equivalent [15]. Note, for example, that in the case of so-called $global-as-view$ mappings [15], which relate one concept in the integration schema s_i to a query over the set of source schemas S_s, a mapping could be defined as $mp_{S_s \to s_i} = \langle c_{s_i}, q_{S_s} \rangle$, where c_{s_i} is a concept, e.g., a relation, in the integration schema s_i and q_{S_s} is a query over a set of source schemas S_s.
- A query result is a set of result tuples $R_{q_{s_i}}$ of a query q_{s_i} posed over a schema s_i, where a single result tuple $r_{q_{s_i}} = AttV$. $AttV$ is a set of attribute-value pairs $\langle att_l, v_l \rangle$ where att_l is an attribute and v_l is its respective value.

We call the union type over these types $MM-Type$. Instances of the four above types characterise the state of the integration effort at a point in time. As such, consistent with the idea of user-driven gradual improvement that is implicit in the 'pay-as-you-go'approach, we expect that instances of the above types will be annotated with assessment and feedback outcomes. Instances of all four types can have annotations of the form $\langle source, value, ontology \rangle$ where *source* is the provenance of the annotation, e.g., the user providing the feedback or the matching algorithm, and *value* is an instance of a concept in the *ontology*. The use of an *ontology* enables different implementations of the operators introduced below to identify the corresponding annotation from a potentially wide range of annotations associated with a $MM-Type$ that can be recognised and processed by them. This generic representation of annotation acts as an extension point and allows us to handle annotations from various sources (e.g., user feedback) of different forms on any of the model management types. Indeed, several proposals have been made for approaches that obtain feedback on and that annotate different model management types. For example, [5] obtain feedback on properties of mappings, and use the feedback to select which mappings participate in query evaluation. By contrast, [2] obtain feedback on query results, and use the feedback to select mappings that meet user-specified precision or recall targets, and to generate new mappings.

The operations introduced below can be combined in various ways to create dataspace management systems, as illustrated in Section 3. The operations extend those that have previously been proposed in model management systems [3, 4] to extend their applicability to the dataspaces life cycle. The majority of them are defined on schematic correspondences (as in [4, 14]) rather than matchings. The signatures of the operations that we will use in case studies are listed in Table 1 with optional parameters in square brackets, and are described below:

- MATCH: Given two schemas s_i and s_j, return a set of matches between them.
- MERGE: Given two schemas s_i, s_j, a set of schematic correspondences $CR_{s_i-s_j}$ between them, and a *kind* which indicates whether a union or a merged schema should be created, merge the two schemas into a third schema s_m, also returning the set of schematic correspondences between the two source schemas and the merged schema $(CR_{s_i-s_m}, CR_{s_j-s_m})$.
- VIEWGEN: Given an integration schema s_i, a set of source schemas S_s and a set of schematic correspondences $CR_{s_i-S_s}$ between s_i and the source schemas, produce a unified set of mappings $MP_{S_s \to s_i}$ that is consistent with the correspondences and that defines how to transform instances from constructs in the set of source schemas S_s into instances of the constructs in the integration schema s_i.

There are also aspects of the functionality to be provided by a DSMS that require the definition of new operations. The signatures of these operations can also be found in Table 1.

- INFERCORRESPONDENCE: Given a set of matches $MT_{s_i-s_j}$ between constructs in schemas s_i and s_j, infer the schematic correspondences $CR_{s_i-s_j}$ between them.

Table 1: Signatures of Operations

Operation	Signature	Types of parameters	Types of results
MATCH	MATCH(s_i, s_j, $[CP]$, $[MT^{an}_{s_i-s_j}]$) $\rightarrow MT_{s_i-s_j}$	Schemas s_i, s_j, optional set of control parameters CP, optional annotated set of previous matches $MT^{an}_{s_i-s_j}$	Set of matches $MT_{s_i-s_j}$
INFER-CORRESPONDENCE	INFERCORRESPONDENCE ($MTs_i - s_j$, $[CP]$, $[CR^{an}_{s_i-s_j}]$) $\rightarrow CR_{s_i-s_j}$	Set of matches $MTs_i - s_j$, optional set of control parameters CP, optional annotated set of previous correspondences $CR^{an}_{s_i-s_j}$	Set of correspondences $CR_{s_i-s_j}$
MERGE	MERGE $(s_i, s_j, CR_{s_i-s_j}, kind, [CP]) \rightarrow \langle S_m, CR_{s_i-s_m}, CR_{s_j-s_m}\rangle$	Schemas s_i, s_j, set of correspondences $CR_{s_i-s_j}$, *kind* indicating whether to create *union* or *merged* schema, optional set of control parameters CP	Resulting schema S_m, Sets of correspondences $CR_{s_i-s_m}, CR_{s_j-s_m}$ between S_m and each of input schemas S_i, S_j
VIEWGEN	VIEWGEN(s_i, S_s, $CR_{s_i-S_s}$, $[CP]$, $[MP^{an}_{S_s\rightarrow s_i}]$) $\rightarrow MP_{S_s\rightarrow s_i}$	Schema S_i, set of source schemas S_s, set of correspondences $CR_{s_i-S_s}$ between source schemas $S_s$and schema s_i, optional set of control parameters CP, optional annotated previous set of mappings $MP^{an}_{S_s\rightarrow s_i}$	Set of mappings $MP_{S_s\rightarrow s_i}$
ANSWERQUERY	ANSWERQUERY(q_{s_i}, $[CP]$, $[MP_{S_s\rightarrow s_i}]$, $[S_s]$) $\rightarrow R_{q_{s_i}}$	Query q_{s_i} over schema s_i, optional set of control parameters CP, optional set of mappings $MP_{S_s\rightarrow s_i}$ to be used for answering query over optional set of source schemas S_s	Set of result tuples $R_{q_{s_i}}$
ANNOTATE	ANNOTATE (M_i, A | M^{an}_j, $[CP]$) $\rightarrow M^{an}_i$	Set of instances of MM-Type M_i, annotation A or annotated set of instances of MM-Type M^{an}_j, optional set of control parameters CP	Annotated set of MM-Type M^{an}_i

- ANSWERQUERY: Given a query q_{s_i} posed over an integration schema s_i, translate the query into sub-queries over a set of source schemas S_s, execute the sub-queries, and combine and rank the results. Query answering can be parameterised further by providing the set of mappings ($MP_{S_s\rightarrow s_i}$) to be used for answering the query or a set of source schemas (S_s) over which the sub-queries are to be generated and executed.

Additional operations not used here include COMPOSE, EXTRACT, and DIFF, which have been shown to be useful for schema evolution and thus are applicable during the maintenance phase [4]. An additional operator that is not considered in the scenarios presented here is IDENTIFYINSTANCESFORFEEDBACK which returns the subset of the instances of an $MM-Type$ provided as input for which user feedback is requested. For example, the candidate set can be chosen using a utility function that identifies instances of an $MM-Type$, which annotated, would provide the most benefit to the dataspace system [13].

Several of the model management operations listed can be extended further to act on annotated instances of $MM-Type$. MATCH, INFERCORRESPONDENCE and VIEWGEN have an optional parameter which allows the passing in of previous results of the operation that may be annotated. If annotation is provided, it can be taken into account when generating updated instances of the $MM-Type$, e.g., matchings with a score below a certain threshold could be excluded. Annotation can be used for incremental improvement of the overall integration, as illustrated in Scenario 3 in the next section using the annotation obtained by gathering user feedback. To associate annotations with their corresponding $MM-Type$ and to propagate the annotation, the following operation is introduced:

- ANNOTATE: Given instances M_i of an $MM-Type$ and a set of corresponding annotations A, annotate the instances with A. This can be used, for example, to annotate the query results with the feedback provided by the user. As an alternative, annotated instances M^{an}_j of a (different) $MM-Type$ M_j can be provided and ANNOTATE used to propagate the annotation.

Similar to the way we extend the $MM-Types$ with annotations of the form $\langle source, value, ontology\rangle$, we extend the operations with optional control parameters CP of the form $\langle value, ontology\rangle$ where $value$ is an instance of a concept in the $ontology$. Examples of control parameters include a threshold on the scores associated with matches or the precision- or recall-target that should be met by result of a query. Using these generic representations of control parameters and annotations enables us to provide various different implementations of the operators that recognise and are able to process various different control parameters and annotations of the $MM-Types$ provided as input.

3. CASE STUDIES

To demonstrate the flexibility of the approach, we indicate how the operations presented could be useful in the dataspace life cycle through a number of scenarios. Previous work [12] has shown that DSMS can vary to a great extent along a number of dimensions. For illustrative purposes, we have chosen to instantiate the framework by varying the following dimensions as summarised in Table 2:

- *Initialisation phase: Correspondences:* these can either be provided manually or inferred automatically,

Table 2: Characteristics of Case Studies

Dimension	Scenario 1 Initial limited integration	Scenario 2 Well known domain, some improvement	Scenario 3 Incremental improvement
Correspondences	manual	automatic	automatic
Integration schema	automatic, union	manual	automatic, merged
User feedback	none	on results	on results

1: {Intitialisation – Create union schema s_i of schemas s_1 and s_2 utilising manually defined matchings and correspondences and generate mappings between s_1, s_2 and s_i:}
2: $\langle s_i, CR_{s_1-s_i}, CR_{s_2-s_i}\rangle$=MERGE$(s_1, s_2, CR_{s_1-s_2}, union)$
3: $MP_{S_s\rightarrow s_i}$=VIEWGEN $(s_i, \{s_1, s_2\}, \{CR_{s_1-s_i}, CR_{s_2-s_i}\})$
4: {Usage – Query:}
5: $R_{q_{s_i}}$=ANSWERQUERY$(q_{s_i}, MP_{S_s\rightarrow s_i})$

Figure 1: Scenario 1

e.g., from matchings, which we assume are derived automatically using schema matching tools (e.g., COMA++ [8]). *Integration schema:* the integration schema is either provided manually, or derived automatically. It can either be a union schema or a merged schema.

- *Improvement phase: User feedback* is one of many kinds of annotation; optional user feedback can be obtained at various stages of the integration process by asking users to specify which matchings, correspondences or mappings they agree or disagree with or which query results they expected to see. User feedback can be utilised to improve the integration incrementally. Here, we will focus on user feedback on query results.

Scenario 1: The first scenario could be used for an initial quick set-up of a dataspace with a limited degree of integration (simple union schema) and with manually provided correspondences. This scenario can be compared to iMeMex [7], [20], in which a union schema is created and the integration is improved gradually by manually providing so called iTrails, which could be loosely compared to schematic correspondences. A union schema is also created in PayGo [17]. The matchings in PayGo, however, are inferred automatically, which is part of our Scenarios 2 and 3.

Here, we assume that two sources with schemas s_1 and s_2 are to be integrated. The integration schema s_i is inferred automatically from the source schemas by generating the corresponding union schema. Schematic correspondences $CR_{s_1-s_2}$ between the sources are provided manually.

The steps required for this scenario are listed in Figure 1. As the schematic correspondences are provided, the integration consists of creating the union schema of s_1 and s_2 by calling MERGE with $kind = union$ (Line 2) and the generation of the mappings between each of the source schemas and the integration schema (Line 3). ANSWERQUERY uses the generated mappings to translate the query into subqueries over the sources (Line 5).

1: {Initialisation – Integrate sources with schemas s_1 and s_2 by matching them with s_i, infer schematic correspondences and generate the corresponding mapping:}
2: $MT_{s_1-s_i}$=MATCH(s_1, s_i)
3: $MT_{s_2-s_i}$=MATCH(s_2, s_i)
4: $CR_{s_1-s_i}$=INFERCORRESPONDENCE$(MT_{s_1-s_i})$
5: $CR_{s_2-s_i}$=INFERCORRESPONDENCE$(MT_{s_2-s_i})$
6: $MP_{S_s\rightarrow s_i}$=VIEWGEN $(s_i, \{s_1, s_2\}, \{CR_{s_1-s_i}, CR_{s_2-s_i}\})$
7: {Usage – Query:}
8: $R_{q_{s_i}}$=ANSWERQUERY$(q_{s_i}, MP_{S_s\rightarrow s_i})$
9: **loop**
10: {Improvement – User provides feedback on results in form of annotation A, which is used to annotate the query results and the mappings:}
11: $R^{an}_{q_{s_i}}$=ANNOTATE$(R_{q_{s_i}}, A)$
12: $MP^{an}_{S_s\rightarrow s_i}$=ANNOTATE$(MP_{S_s\rightarrow s_i}, R^{an}_{q_{s_i}})$
13: {Usage – Query using annotated mappings:}
14: $CP = \langle 0.8, precisionTarget\rangle$
15: $R_{q_{s_i}}$=ANSWERQUERY$(q_{s_i}, CP, MP^{an}_{S_s\rightarrow s_i})$
16: **end loop**

Figure 2: Scenario 2

Scenario 2: The second scenario fits situations where the domain or the required information is well understood, i.e., the integration schema can be designed manually; examples include personal information management (e.g., SEMEX [9]), or disaster management. For this case study, we assume that the integration schema s_i is provided and that the sources with schemas s_1 and s_2 are to be integrated. In addition, we gather user feedback on the query results, which is used to annotate the mappings with quality information, which, in turn, may be used to choose only the mappings with sufficient quality to answer subsequent queries.

The steps required to integrate the sources and query them are listed in Figure 2. The integration process consists of the operations MATCH (Lines 2 and 3), which matches source schemas s_1 and s_2 to the given integration schema s_i, and INFERCORRESPONDENCE (Lines 4 and 5), which infers the schematic correspondences between the source schemas and the integration schema. This is followed by the generation of the mappings between each of the source schemas and the integration schema (Line 6) and the answering of queries posed over the integration schema using the generated mappings to translate the query into subqueries over the sources (Line 8) (same as Lines 3 and 5 in Figure 1). A user who issues a query can provide feedback on the results, in the form of annotations indicating, e.g., which results were expected (true positives), which were returned in error (false positives) and which were expected but not returned (false negatives). The feedback is used to annotate the query results accordingly (Line 11), which in turn are used to annotate the mappings that were used to produce them (Line 12). Examples of mappings annotations include estimates of the precision and recall of the mappings based on the user feedback provided. The annotated mappings are then used to answer subsequent queries (Line 15), e.g., by choosing only mappings with precision and recall above a certain threshold (a possible implementation for this mapping selection process can be found in [2]). As indicated in Figure 2 by the *Loop ... EndLoop* block around Lines 10-15, the process of annotating query results, propagating that annotation to the

1: {Initialisation – Integrate sources with schemas s_1 to s_k by matching them, inferring schematic correspondences, creating a merged integration schema and generating the corresponding mappings:}
2: $MT_{s_1-s_2}$=MATCH(s_1, s_2)
3: $CR_{s_1-s_2}$=INFERCORRESPONDENCE($MT_{s_1-s_2}$)
4: $\langle s_m, CR_{s_1-s_m}, CR(s_2 - s_i)\rangle$ = MERGE($s_1, s_2, CR_{s_1-s_2}, merge$)
5: $MP_{S_s \to s_m}$ = VIEWGEN ($s_m, \{s_1, s_2\}, \{CR_{s_1-s_m}, CR_{s_2-s_m}\}$)
6: **loop**
7: $MT_{s_i-s_m}$=MATCH(s_i, s_m)
8: $CR_{s_i-s_m}$ = INFERCORRESPONDENCE($MT_{s_i-s_m}$)
9: $\langle s_{m'}, CR_{s_i-s_{m'}}, CR(s_m - s_{m'})\rangle$ = MERGE($s_i, s_m, CR_{s_i-s_m}, merge$)
10: $MP_{S_s \to s_{m'}}$ = VIEWGEN ($s_{m'}, \{s_i, s_{m'}\}, \{CR_{s_i-s_{m'}}, CR_{s_i-s_{m'}}\}$)
11: **end loop**
12: {Query:}
13: $R_{q_{s_{m'}}}$=ANSWERQUERY($q_{s_{m'}}, MP_{S_s \to s_{m'}}$)
14: {Improvement – User provides feedback on results in form of annotation A, which is used to annotate the query results and the mappings:}
15: $R^{an}_{q_{s_{m'}}}$=ANNOTATE($R_{q_{s_{m'}}}, A$)
16: $MP^{an}_{S_s \to s_{m'}}$=ANNOTATE($MP_{S_s \to s_{m'}}, R^{an}_{q_{s_{m'}}}$)

Figure 3: Scenario 3

mappings used to answer the query and then re-running the query utilising those annotated mappings can be executed multiple times for incremental improvement of the dataspace by applying feedback (e.g., [2]). When applying the ANSWERQUERY operator the second and subsequent times, different implementation strategies can be chosen that take account of control parameters CP (e.g., specifying which precision target the query results should meet) and annotated mappings $MP^{an}_{S_s \to s_i}$ as input. For example, utilising the user feedback provided on the query results, mappings can be annotated with precision and recall values, which in turn can then be utilised to select the mappings to be used for answering the query [2].

Scenario 3: The third scenario is similar to UDI [6], in that the source schemas are matched and a merged integration schema is created automatically. This initial integration is improved further by gathering user feedback on the results of queries posed over the integration schema and propagating the annotation to the mappings, correspondences and matchings.

Here, we assume that sources with schemas s_1 to s_k are to be integrated by automatically deriving the matchings and correspondences between two source schemas and generating the merged integration schema s_m. Once an integration schema is generated, the remaining sources are integrated incrementally by automatically deriving the matchings and correspondences between the next source schema s_i and the current integration schema s_m. Using those generated correspondences, a new integration schema is generated by merging the current integration schema with the newly integrated source. The integration is improved by gathering user feedback on the query results, which is then propagated.

The steps required for this scenario are listed in Figure 3. The integration process starts by matching S_1 and S_2 (Line 2), inferring the schematic correspondences from the matches (Line 3), merging the two schemas (Line 4) and generating the corresponding mappings (Line 5). Additional sources are integrated incrementally by repeating the steps listed in Lines 2 - 5, using the data source to be added with schema s_i and the current merged schema s_m as input and creating a new integration schema $s_{m'}$ (Lines 7 - 10 in Figure 3). For an example of an implementation of MERGE that is commutative and associative, see [19]. The authors show under which circumstances the order in which the various schemas are merged has no effect on the resulting schema. Over the integration schema, a query is posed, which is translated into sub-queries and executed over the sources, utilising the mappings generated (Line 13) (same as Line 5 in Figure 1). The user can then annotate the query results by providing feedback (Line 15 - same as Line 11 in Figure 2). The annotation of the query results is then propagated accordingly to the corresponding mappings that produced the query results (Line 16 - same as Line 12 in Figure 2). This can be followed, for example, by using ANNOTATE to propagate the annotation from the mappings to the correspondences and the matchings, and using these annotated $MM - Types$ as input to the extended model management operators for incremental improvement of the integration.

4. CONCLUSION

Model management define a range of operations, that have been shown to be useful in a range of data integration scenarios [4]. Model management is important in significant measure because it identifies operations that have been shown to be reusable in different contexts. This paper has explored the extent to which existing model management operations can be used to support the development of dataspaces. In so doing, the paper has made the following contributions:

- A small number of additional operators have been identified (INFERCORRESPONDENCE, ANSWERQUERY, IDENTIFYINSTANCEFORFEEDBACK, ANNOTATE) that reflect new requirements from dataspaces.
- Conservative extensions to the existing model management operators have been proposed that accommodate incremental refinement, building on annotations that can be associated with query results, matchings, schematic correspondencies or mappings.
- Three scenarios have been described that demonstrate how the extended model management proposal can be used to support the functionalities supported by existing proposals for dataspace management systems.

These results also demonstrate that an approach to dataspace management that builds on model management defers commitment to certain decisions that have previously tended to be hard wired in dataspace platforms. However, application requirements have been shown to be diverse in practice (e.g., requirements and constraints are very different in personal data management and in web scale data integration), so it seems unlikely that a one-size-fits-all approach will be broadly applicable. Thus, the increased flexibility that is offered by building on model management opens the door to the development of dataspaces tailored to meet application requirements, while building on a manageable collection of generic components.

5. ACKNOWLEDGEMENT

This work is funded by the EPSRC under Grant EP/ F031092/1. We are grateful for this support.

6. REFERENCES

[1] P. Atzeni, L. Bellomarini, F. Bugiotti, and G. Gianforme. Mism: A platform for model-independent solutions to model management problems. *J. Data Semantics*, 14:133–161, 2009.

[2] K. Belhajjame, N. W. Paton, S. M. Embury, A. A. A. Fernandes, and C. Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *EDBT*, pages 573–584, 2010.

[3] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A vision for management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[4] P. A. Bernstein and S. Melnik. Model management 2.0: Manipulating richer mappings. In *SIGMOD*, pages 1–12, 2007.

[5] H. Cao, Y. Qi, K. S. Candan, and M. L. Sapino. Feedback-driven result ranking and query refinement for exploring semi-structured data collections. In *EDBT*, pages 3–14, 2010.

[6] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD*, pages 861–874, 2008.

[7] J.-P. Dittrich and M. A. Vaz Salles. idm: A unified and versatile data model for personal dataspace management. In *VLDB*, pages 367–378, 2006.

[8] H.-H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Inf. Syst.*, 32(6):857–885, 2007.

[9] X. Dong and A. Y. Halevy. A platform for personal information management and integration. In *CIDR*, pages 119–130, 2005.

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

[11] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage years. In *VLDB*, pages 9–16, 2006.

[12] C. Hedeler, K. Belhajjame, N. W. Paton, A. A. A. Fernandes, and S. M. Embury. Dimensions of dataspaces. In *BNCOD*, pages 55–66, 2009.

[13] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860, 2008.

[14] D. Kensche, C. Quix, X. Li, and Y. Li. Geromesuite: a system for holistic generic model management. In *VLDB*, pages 1322–1325, 2007.

[15] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[16] J. Liu, X. Dong, and A. Halevy. Answering structured queries on unstructured data. In *WebDB*, pages 25–30, 2006.

[17] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[18] L. Mao, K. Belhajjame, N. W. Paton, and A. A. A. Fernandes. Defining and using schematic correspondences for automatically generating schema mappings. In *CAiSE*, pages 79–93, 2009.

[19] R. Pottinger and P. A. Bernstein. Associativity and commutativity in generic merge. In *Conceptual Modeling: Foundations and Applications*, pages 254–272, 2009.

[20] M. A. Vaz Salles, J.-P. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi. itrails: Pay-as-you-go information integration in dataspaces. In *VLDB*, pages 663–674, 2007.