A Metadata Resource to Promote Data Integration

Len Seligman, Arnon Rosenthal

The MITRE Corporation
{seligman, arnie}@mitre.org

Abstract. Data integration is expensive, largely due to the difficulty of gathering relevant metadata. We observe that a database may participate in multiple integration efforts, and that much of the same metadata is required, regardless of which kind of effort is being undertaken. This presents a tremendous opportunity to reduce the long-term development and maintenance cost of interoperable systems, by gathering relevant metadata only once and representing it in ways that are amenable to reuse. We present metadata management strategies that promote reuse and describe a repository-based vision for the development of interoperable systems.

1. Introduction

Metadata collection and management play a critical role in developing and maintaining *data integration* – the ability to meaningfully exchange or combine information across independently developed databases and applications. We focus here on metadata that describes the meaning and representation, rather than metadata that summarizes the contents of some object or document. Such metadata helps us ensure that when we communicate data between systems, the recipient interprets it as the sender intended.

In the emerging information-rich world, there will be many data providers and consumers, and a consumer may require information from multiple data providers. Two systems *interoperate* if they communicate information amongst themselves. Data is *fused* if consumers see logical objects that combine information from multiple sources. *Integration* will be used as a generic term, to include both modes.

Much published research in data integration concerns support for one-shot schema integration or integration via multidatabase query. Many methodologies ignore the need to coexist with other forms of integration. This paper examines a complex environment in which organizations carry out multiple integration efforts, with multiple partners, producing different kinds of integrated systems that involve overlapping subsets of its component databases. This environment, which we call “data integration in the large,” is typical of our clients and most large organizations.

There are surprisingly many varieties of data integration efforts including (1) providing a federated view of multiple databases [9], (2) reverse engineering existing systems, consolidating them, and migrating them toward a new target system [2], (3) supporting data flow among loosely coupled interoperable systems, (4) "scrubbing" and fusing information from multiple databases in order to build a data warehouse [11], and others. Figure 1 illustrates options 2 -4, which are less frequently discussed in the research literature.

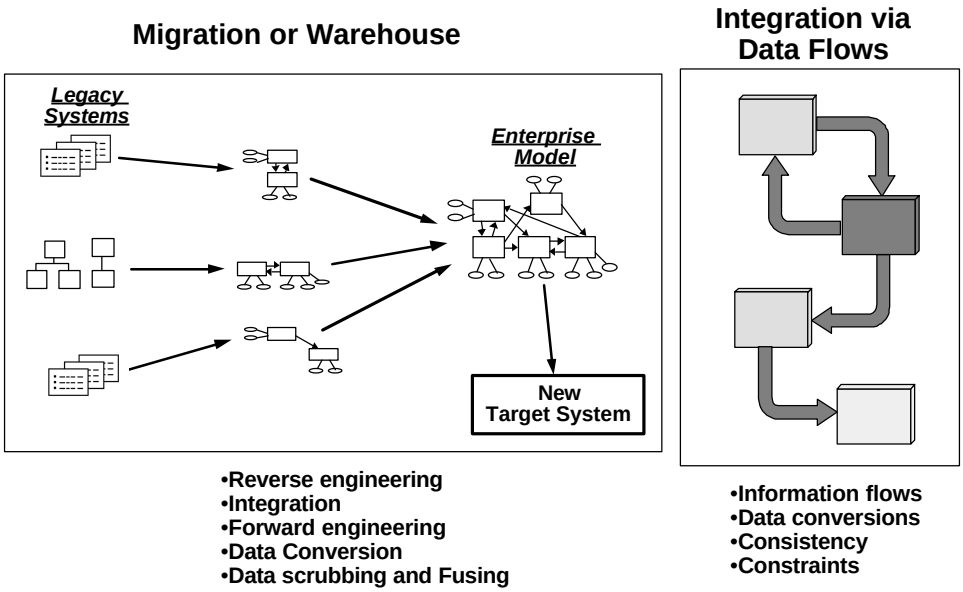

Figure 1. Other Ways to Provide "Integrated" Data

Data heterogeneity is the most serious obstacle to integration efforts. It will remain a major pain in the development budget, long after it becomes routine to use DBMS gateway products to give the illusion that a server users the client's query language. Collecting and validating the necessary metadata (e.g., determining whether concepts used in two organizations are really the same, based on their code, documentation, procedures manuals, and oral lore) is very costly. Tools will be able to assist, but the effort will remain time-consuming. It is therefore critical to devise ways to *reuse* integration metadata across integration efforts.[1]

In this paper, we describe the metadata requirements of several varieties of data integration and note their large overlap. Based on these requirements, we recommend methodologies for improving the management of intersystem metadata and for providing automation support for system integrators. Finally, we propose a vision for metadata-driven development and evolution of interoperable systems.

[1] Some of the material here has been adapted from [5], which focused entirely on reuse.

2. Varieties of Data Integration Problems

This section describes a variety of integration problems that need to be solved, sketches their metadata requirements, and notes the substantial overlap in these requirements. This overlap is significant, because it presents opportunities for metadata reuse and resulting cost-savings.

2.1. The Unary Case: Providing a New Interface to an Existing System

The simplest step is to provide an alternative interface through which a data source can be viewed and manipulated. Since our topic is interoperability, we focus on new interfaces that make the data source better able to interoperate with other schemas in the system, especially a federation's schema.[2]

We call this the "unary" case because one deals with a single system. A benefit of the unary focus is that the metadata, new interfaces, and perhaps view definitions are available for reuse when integrating with other different federations. As part of mapping to a preexisting federation schema, the unary case defines natural modules for metadata, for view definitions, and for a schema integration toolkit.

When several component systems are individually mapped to the same schema, users can then be given a *uniform interface* that can be used to access a number of sources. (This is not yet a *unified interface*, since it applies to one source at a time.) With a uniform interface, an application can run on any source that provides sufficient information. Above this uniform interface, additional modules of metadata, plus mediation that produces view-mappings, can provide fusion across a set of sources. One can also provide a source-selection mediator, which determines what set of sources should be used for a particular request. (Such an approach is more flexible than having a federated schema, which normally fuses data from a fixed set of sources).

The specifications and mediation code that support the new interface can be considered a *semantic gateway*, closely analogous to the (syntactic) DBMS gateway that can make DB2 look like Sybase. Note that just as a DBMS gateway does not include a distributed query capability, for modularization reasons a semantic gateway does not attempt fusion.

Metadata Requirements for Providing a New Interface

---

[2] Export schemas that determine what data shall be made available are not part of integration; only views that reduce schema heterogeneity are considered here.

In order to support this kind of "integration," one must capture metadata about the component system and the new interface. This includes not only the schemas, but for each schema constituent (e.g., table, attribute) one must describe the meaning and the representation details.
Meaning can be described internally to the constituent, or by relating the constituent to a constituent elsewhere, or by relating the constituent to a reference definition that is maintained outside the individual systems. The first case tends to promote unnecessary diversity. The second case does not increase diversity, but it results in pairwise connections that are difficult to reuse in subsequent integration efforts. As described in Section 3.4, the third approach of specifying semantic relationships with reference definitions seems to maximize reuse.

In order to support the new interface, one needs a mapping between the component and interface schemas in executable form (e.g., an SQL view definition). However, as described in Section 3.3, these executable mappings can often be generated from smaller, more modular chunks of information. Extensible libraries of conversion functions provide an important building block.

Additionally, the metadatabase should capture information on access controls and information sharing. To support change impact analysis, the metadata should record the obligations the administrator of each component system owes to others. For example, such metadata could record the obligation of a database administrator to support a particular interface and that a certain individual must be given six month's notice prior to changing it.

2.2. Transparent Unified Access to Heterogeneous Sources

This is the most familiar type of data integration. The user formulates queries in terms of an integrated view. This could be an enterprise schema or a federated view [9]. In either case, the user is insulated from source selection and data fusion, as well as the representation details of each component data source. This is illustrated in Figure 2, in which a user formulates his query without regard to the location of particular data or the (possibly heterogeneous) representations used by the various Fisheries, Water Board, and Land Use databases.

Creation of a federated view involves determining the view's interface, and mapping individual components to that interface. In the literature on schema integration, the two are tightly coupled, but that need not always be the case. In our experience, it seems rare that two large schemas are integrated completely (and successfully). In addition, while the literature stresses producing a federated interface by means of schema integration, often the federated interface is specified to meet the needs of a new application. One then must produce views that derive this information from existing components.

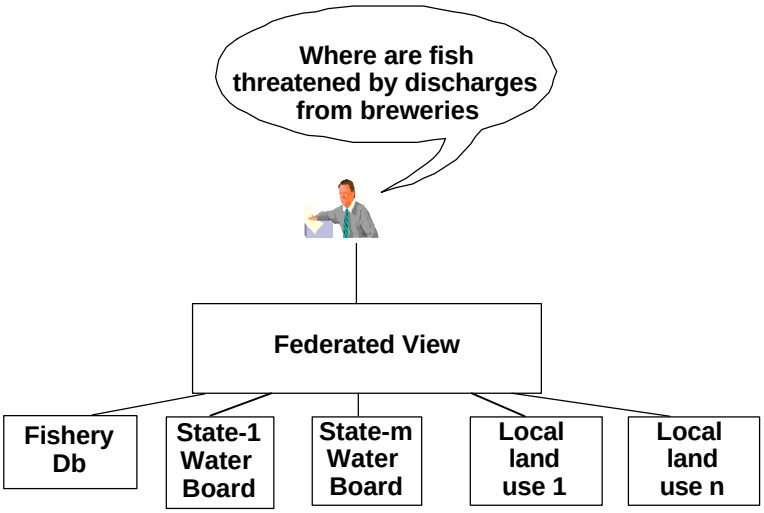

Figure 2. Transparent Access via a Federated View

Once the federated schema is known, creation of an integrated view can be decomposed into a series of efforts to support new unary interfaces. For example, to provide a federated view F which is derived from components A, B, and C, one must create a new interface conforming to F for each component A, B, and C.

The only additional metadata required beyond that collected for the unary case is the following:

- Descriptions of which component systems are to be included in a given integrated view
- Rules for *object fusion.* Object fusion provides virtual objects that gather together all the information about a single real world entity (e.g., news reports about both Elvis and E. Presley).
- Rules for *value fusion* . Value fusion provides a single virtual object when the inputs have different assertions about the same property value. (For example, if three components list Elvis' weight as 170, 230, and -999, what value or set of values should be returned?)

2.3. Data Flows Among Loosely-coupled Interoperable Systems

Many data-intensive systems exchange information through Electronic Data Interchange (EDI) messages, extract programs, files conforming to an exchange standard, and other mechanisms that cause data to flow from a source to a recipient. (Recursing, one can often consider there to be two separate dataflows, from the source to the intermediate form and from the intermediate form to the recipient). Data flows are sometimes used instead of a federated approach, because performance and reliability needs may rule out pulling data

from widely-scattered sites at run time. Instead, data flows are scheduled so that each site has an adequately recent snapshot of information.

Knowledge of many kinds goes into the programs that create and consume these data flows, but that knowledge is rarely documented well. This knowledge concerns both the contents of the data flows, and their scheduling. For example, over several weeks we partially unscrambled a 6000 line C program with embedded SQL that extracts data from one Oracle database, converts it, and loads into another. The code performs data extraction from the source, attribute and class transformations, constraint checking, and information fusion at the recipient.

The knowledge needed to create such programs would be invaluable for other integration efforts (e.g., to create a federated view), but it has been quite effectively encrypted. The threads need to be separated, and each one documented as declaratively as possible. Current tools can barely determine which C variables are referenced in each SQL statement; they certainly cannot follow a value from an SQL read, through various constraints and conditionals and other C variables, and into an update.

At a minimum, it would be useful simply to document the data interrelationships that are now only implicit in the extract and load programs. Ideally, this knowledge would be captured in reusable modules from which extract and load programs could be automatically generated.

Interdatabase consistency is often required [6], but the required and achieved consistency are rarely documented. For example, one might want to enforce a constraint that a data receiver's value for FuelOnHand should not deviate by more than 20% from its current value in the source database. In most current systems, these constraints are poorly documented and are managed in an ad hoc manner, both in the semantic constraints and the required timeliness of repair. Typically one documents only the operational procedures (e.g., daily download) and some of the mechanisms (e.g., triggers) used; other consistency enforcement may be hidden, e.g., in programs that update several databases simultaneously. A metadata-based approach that captures the constraints declaratively offers clear advantages, and may even make it possible to automatically generate constraint handlers [8] or replication server subscriptions.

Most of the metadata required to drive loosely-coupled interoperable systems is shared with the unary case. However, at least two new kinds of metadata are required:

- Assertions about the frequency and regularity of data flows. For example, is the data shipped daily, weekly, as soon as relevant updates take place? Or perhaps they happen erratically, but never more than a month apart?
- Descriptions of the mechanisms used to send data from the source system to the receiver. For example, are EDI messages used, and if so, what is the format? Is a

combination of 3GL code and SQL view definitions used? Or perhaps a commercial data replication product (e.g., Sybase Replication Server)?

Additional metadata is required to support interoperable systems which perform consistency enforcement between data sources and receivers:

- The constraints themselves, in a declarative form.
- The handlers used to monitor constraint conditions and to initiate data flow upon violation of those constraints.

2.4. Data Warehousing

Data warehousing brings together selected data from component data sources, cleans it (e.g., by removing inconsistencies), and transforms it so that it is suitable for end-user query and analysis (e.g., by creating summary tables along different "dimensions," such as sales by product, by region, and by supplier).

Administering a data warehouse requires a great deal of metadata, but it does not appear to require any new kinds. Essentially, warehouse development requires the union of the metadata required to support the previous two integration scenarios: Federated Views and Loosely-coupled Interoperable Systems. Below we elaborate on this observation.

Data warehouses are collections of views, each of which is derived from selected data in the component databases. As a result, one needs all the metadata used to construct federated views. As with federated views, the integrator (1) constructs a common target interface for the relevant subsets of all component databases that feed the warehouse, and (2) specifies rules for doing object and value fusion. Admittedly, the kinds of transformations and reconciliation supported in a warehouse are likely to be more complex than in a federated database, which performs these operations at query time. However, the metadata constructs should be the same.

Additional metadata is needed to describe the required flow of data from the component databases to the warehouse—descriptions of the frequency and regularity of these data flows and the mechanism to be used. Also, if consistency constraints are to be enforced between the component data sources and the warehouse, then these should be captured in the metadatabase, along with a description of the mechanisms that check the constraints.

2.5. Data Migration

When systems are reengineered, it is often necessary to migrate a database and its applications to a new environment, e.g., from a legacy file system or DBMS such as VSAM or IMS to a relational or object database. An even more challenging task arises when several organizations with functionally similar databases (e.g., Inventory) are to be merged.

Like the unary case, migrations require creating a new schema and mapping the old schema to the new one. Data must then be shipped from the old system(s) to the new, typically via bulk load statements. This process may require difficult reverse engineering and integration efforts. One must define a schema that contains the essential information from all the components, but omits data elements that were implementation artifacts (e.g., status fields that help control multistep processes). In addition, the old and new systems may need to be run in parallel for a period of time. As in the "Loosely-coupled Interoperable Systems" scenario, the two systems may need to be kept consistent. These consistency constraints can sometimes be handled by propagating database changes, possibly in both directions [2].

The Data Migration scenario may involve bidirectional dataflows, but requires essentially the same metadata as the construction of a data warehouse. Careful configuration management will also be essential.

2.6. Summary

Figure 3 illustrates the substantial overlaps in metadata requirements among integration scenarios. In that figure, rectangular boxes represent different data integration scenarios. Boxes with rounded edges represent different kinds of metadata. An arrow from a metadata box to a scenario means that that metadata is required to support the integration scenario to which is connected. For example, Federated Views require all the metadata used by the New Interface scenario plus metadata on the rules for doing object and value fusion. Some scenarios have no unique metadata requirements; for example Warehousing and Migration require only the metadata used by Federated Views and Loosely-coupled Interoperable Systems.

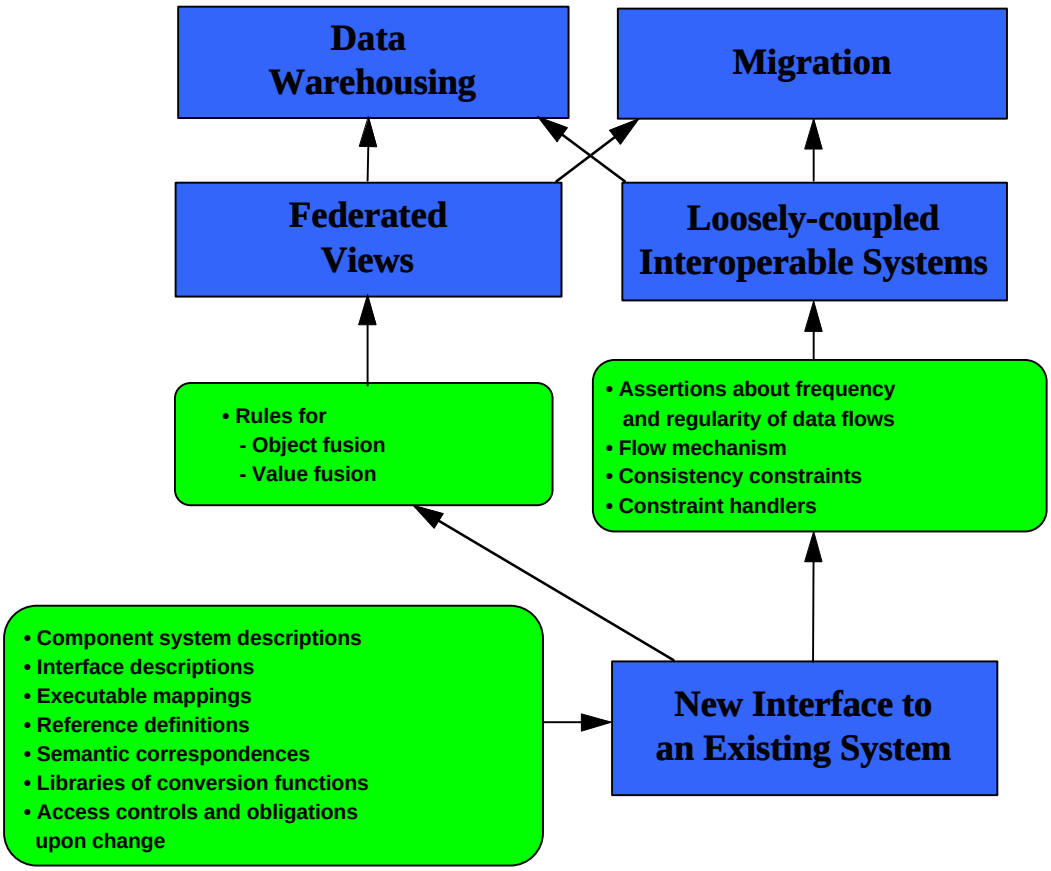

Figure 3. Overlapping Metadata Requirements Among Integration Scenarios

3. Guidelines for Capturing Reusable Metadata

This section proposes some guidelines to help methodology and tool developers maximize the reusability of collected metadata. This reuse is critical if we are to reduce the cost of data integration in the large. Some of this material is described in greater detail in [5].

3.1. Go Beyond the Data Dictionary: Capture Information on Representation and Semantics

Data dictionaries capture useful information about component databases. However, much more metadata is required to facilitate data integration.

Data dictionaries are very limited in the amount of representation information they capture. At the table level, they capture the names of the attributes, while at the attribute level, they typically capture the datatype and possibly domain constraints. Information on the interpretation of the attributes (e.g., “Aircraft.Altitude is in kilometers”) may be expressed in

textual form, but is usually not represented in a form that can drive automated tools such as mediators [12].

Techniques for capturing more of this information in a modular way are proposed in [7]. Meta-attributes are <property, value> pairs that describe the representation and interpretation of attributes. Given two attributes that are described in terms of common meta-attributes, it is often possible to determine if meaningful data exchange can occur and, if so, to automatically generate the required transformations using a library of conversion functions. Some data dictionaries (e.g., the U.S. Government's Defense Data Repository System) capture certain predefined meta-attributes (e.g., UnitOfMeasure), however, users cannot easily extend them to capture additional descriptive properties for new kinds of data (e.g., format and version level for attributes of type *document*).

3.2. Capture Information on all Data Containers, Not Just Those Managed by DBMSs

It is important to capture this metadata not only on databases and files, but also on Electronic Data Interchange (EDI) messages. Often, these messages are the mechanism by which data flows from one system to another. For example, the U.S. military has several families of messages by which information (e.g., mission plans) flows from one system to another. Data carried in such messages presents most of the same administration requirements as fields in a database. Another example of non-DBMS data containers is file exchange formats. These have been defined for logistics, geospatial data, and other domains; their metadata requirement seems essentially identical to the treatment for messages. Both messages and files may require constructs beyond flat schemas, e.g., set- and entity-valued attributes. Use of the same metadata constructs as those used to describe database fields will greatly facilitate interoperability.

In addition, it is important to recognize that not all views of data are managed by database management systems. For example, many organizations now use commercial data replication products to populate data warehouses. These products employ "replication subscriptions" to determine what data should be cached and how often the data should be refreshed. Subscriptions have many of the same properties as database views, however, the subscriptions are expressed in a variety of proprietary languages. To facilitate metadata reuse, it is important that metadata about the subscriptions be captured in a standard form, from which replication subscription can be generated. As another example, applications often restructure data, and when they do, the transformations should be described in a metadatabase using the same operators used to describe database views.

3.3. Capture Small Modules of Information

To maximize reusability, one needs metadata granules of varying size. A small granule is more likely to match the needs of many systems; on the other hand, if many small granules are all held in common (e.g., for systems that share a data feed), then it is easier to administer if they are gathered into larger granules.

We illustrate the granularity issue by considering large SQL view definitions. If a view contains all the data needed for a new requirement, in the proper form, then it is a convenient granule. Otherwise, one wishes to reuse smaller granules (and to generate the view definition automatically). We give two examples.

First, a view that defines a new interface may involve many attributes, and the query that derives it may be a large expression in a textual query language. These are awkward units for reuse. The next integration effort may require different transformations on some of the attributes, and may require attributes not mentioned in the view. It is undesirable to administer multiple layers of view, where some layers undo the work of lower ones..

Reuse will be easier if the information in the view definition is modularized. Frequently, the derivations of attributes in a view's target list are independent. The fact that Name is truncated to 20 characters is independent of the fact that Aircraft_Range is converted from miles to kilometers. Each of these can be stored in the repository as a single assertion, reusable as needed. An analyst who wishes to reuse or modify the conversion on Aircraft_Range can now avoid poring through a view definition that derives thirty other attributes. Security is also improved because one can release definitions selectively.

Second, a similar tactic can be used with views that span components. As in [10], one can capture a collection of intercomponent assertions, and use them to generate the view. For example, suppose in separate integration efforts, a view that spans components 1 and 2 combines EMPLOYEE and WORKER, while a view that spans components 2 and 3 combines WORKER and PERSON. If a later integration effort on components 1 and 3 needs to combine information about EMPLOYEE and PERSON, it will be difficult to do so.

If one has intercomponent assertions instead of just views, the situation is easier. For example, one might have assertions on the actual populations, that every instance of EMPLOYEE in component 1 appears as an instance of WORKER in component 2, and that a similar inclusion holds between WORKER in component 2 and PERSON in component 3. Now one can infer an inclusion between EMPLOYEE and WORKER.

Negative and uncertain information should also be considered part of the metadata resource. Suppose an administrator has answered "No" to the question "Does Aircraft.Readiness in the maintenance database mean the same as Aircraft.Readiness in the air defense system?". This question should not be asked again.

3.4. Use Reference Definitions

One powerful technique is to support a library of reusable reference definitions, capturing commonality wherever it is discovered. As discussed in section 3.3, these can be for either small or large chunks of information. One infers correspondence between component system metadata objects from the fact that they are asserted to be the same as some reference object X. (Assertions of lesser degrees of similarity are useful for information exploration but seem of limited use in generating operational applications).

Reference definitions can come in many forms. First, one can define the name and meaning of a small number of common domains, and tell what representation properties need to be understood for each. For example, one can predefine Distance (in prose) and note that distances are numeric and have LengthUnits. (The reference definition may subsume multiple database attributes (e.g., for Location as <Latitude, Longitude>, or for Price as <Amount, Currency>). Other approaches connect small reference definitions into a larger structure that provides a conceptual schema for an application requirement or for a problem domain. These schemas can be local to an organization, or can be provided by a standards committee. As an extreme case, MCC's Carnot project [3] employs the huge Cyc knowledge base (which has millions of definitions) as a reference model for objects in "common sense reality."

The reuse advantage of reference definitions is that to relate n systems one needs only n sets of intercomponent assertions, rather than $O(n^2)$. The information that one component system's class "Car" means the same as another component's "Voiture" is hard to reuse; the information that it matches the definition of Automobile in a particular organizational or domain-specific ontology is more helpful. Also, the enterprise schema may be well considered and documented, as it receives substantial attention.

As a modest extension, one can provide multiple reference schemas instead of one. This enables integrators to refer to any "well known" object, even when there is no universally accepted conceptual schema. Further research is required in how to manage the evolution of reference schemas, and (even tougher) of multiple, coexisting reference schemas.

4. Conclusions

This paper has described the challenges of data integration in the large, in which component databases participate in multiple integration efforts. There is significant overlap in the metadata requirements of several important integration scenarios, which presents an opportunity to reduce costs through metadata reuse.

Figure 3 presented an overview of the relationship among integration problems. What is most striking is that all the metadata captured to support the unary case is useful for every other integration scenario. We anticipate that similar reuse will be possible with mediators, e.g., a mediator that uses the metadata to generate view definitions in a query language. In addition, because the unary metadata provides improved or alternate documentation of an individual system, it can be used in other efforts in which that system participates. The previous section presents more detailed guidelines for reuse.

In order to achieve the desired reuse, however, organizations must effectively manage integration-relevant metadata. We advocate the use of a repository-centered development approach [1]. A repository is a database that stores and manages metadata about an organization's information assets and which sits at the center of an Integrated Computer-aided Software Engineering (I-CASE) environment. In I-CASE, there are multiple specialized CASE tools, each of which focuses on a particular aspect of the design and implementation process. Each tool stores its outputs in the repository, thereby making it possible to use multiple niche tools as a part of the same design effort. To date, the I-CASE approach has been employed mainly for the development of single systems, however, we believe it has great promise for supporting interoperable systems.

We close by presenting a vision for data integration in the large, illustrated in Figure 4. We envision a loose collection of tools for collecting, maintaining, and exploiting the metadata which enables data integration. This metadata is stored and managed in a repository, which includes information on component systems, reference definitions (including reference schemas, ontologies, organizational data standards, etc.), assertions about correspondences among components and between components and the reference definitions, libraries of conversion functions, and schemas for any integrated views (which might be used for a distributed query tool or as the schema of a data warehouse). Once collected, this metadata can be used to drive several different kinds of integration efforts. For example, in Figure 4, descriptive information on the Land DB and the River DB need only be collected once, even though they both participate in multiple integration efforts.

We are currently prototyping the architecture shown in Figure 4, including parts of the integration toolkit (to identify corresponding data elements across systems), the metadata repository, and a generator of semantic gateways which exploits metadata stored in the repository. Preliminary indications are that we can use the metadata-driven approach to support semantic interoperability among heterogeneous databases [7], argument lists for service invocations [4], and file exchange formats.

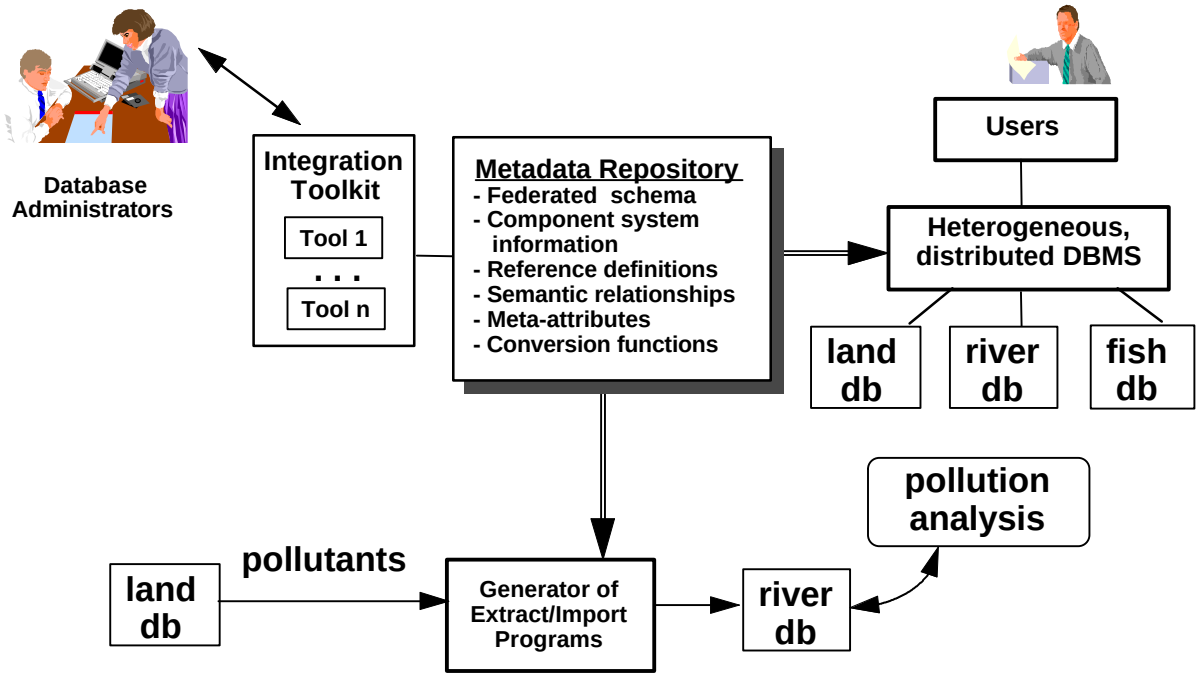

Figure 4. Interoperable Information Systems of the Future

Acknowledgments

The authors wish to thank Jay Scarano, Scott Renner, and Chris Bosch, with whom we have shared many illuminating conversations on data integration, repositories, and mediation.

References

[1] P. Bernstein, U. Dayal, "An Overview of Repository Technology", *Intl. Conf. on Very Large Data Bases*, Santiago, Chile, 1994

[2] M. Brodie, M. Stonebraker, "DARWIN: On the Incremental Migration of Legacy Information Systems," TR-0222-10-92-165, GTE Laboratories, Waltham, MA

[3] C. Collet, M Huhns, W Shen, "Resource Integration Using a Large Knowledge Base in Carnot," *IEEE Computer*, 24(12), December 1991

[4] A. Rosenthal and E. Sciore, "Description, Conversion, and Planning for Semantic Interoperability", *IFIP WG6.2 Conference on Data Semantics*, Atlanta GA, 1995, Kluwer, New York, 1996.

[5] A. Rosenthal and L. Seligman, "Data Integration in the Large: The Challenge of Reuse," *Int. Conf. on Very Large Databases*, Santiago, Chile, 1994

[6] M. Rusinkewicz, A. Sheth, G. Karabatis, "Specifying Interdatabase Dependencies in a Multidatabase Environment," *IEEE Computer*, Vol. 24, No. 12, December 1991

[7] E. Sciore, M. Siegel, A. Rosenthal, “Using Semantic Values to Facilitate Interoperability among Heterogeneous Information Systems," *ACM Transactions on Database Systems*, 19(2), June 1994

[8] L. Seligman and L. Kerschberg, “An Active Database Approach to Consistency Management in Data- and Knowledge-based Systems,” *Int. Journal of Intelligent and Cooperative Information Systems*, 2(2), 1993

[9] A. Sheth and J. Larson, “Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases,” *ACM Computing Surveys*, 22(3), September 1990

[10] A. Sheth, S. Gala, and S. Navathe, “On Automatic Reasoning for Schema Integration,” *International Journal on Intelligent and Cooperative Information Systems*, 2(1), March 1993

[11] J. Widom, ed., “Special Issue on Materialized Views and Data Warehousing,” *Data Engineering Bulletin*, IEEE Computer Society, 18(2), June 1995

[12] G. Wiederhold, “The Roles of Artificial Intelligence in Information Systems,” *Journal of Intelligent Information Systems*, August 1992