# Getting Data to Applications: Why We Fail, How We Can Do Better

**Arnon Rosenthal, Frank Manola, Scott Renner**

The MITRE Corporation, 202 Burlington Road, Bedford MA 01730
(781) 271- {7577, 8147, 2000}
{arnie, fmanola, sar }@mitre.org

## ABSTRACT

*Many initiatives, governmental and commercial, have pursued the grand vision of "transparent access" – making all data available to all consumers (users and applications), in a way the consumer can interpret, anywhere and at any time. Among large-scale enterprises, success stories in achieving such visions seem rare or nonexistent. Instead of leading one more charge "over the top", we suggest that a better guiding metaphor would be "continual evolution of a partially satisfactory system". This resembles what we have today, and what we will have in 2010. The key question, then, is: "How can the builders of each system/data-resource prepare to work with partners, including future, unknown partners". We address this by identifying first steps that may offer some immediate benefits, give crucial stakeholders (metadata providers) incentives to participate, and yet fit a long term vision of greatly increased access.*

Key Words - data integration, metadata, enterprise systems, interoperability, architectures

## 1. Introduction[1]

The data resources in most large enterprises generally exist as many separate islands of data, each maintained by one community for their purposes, but largely unusable by others. We would like to see one single integrated data resource usable by all. This goal is often described as "all data available to anyone, any way they choose, anywhere, and at any time". Many large-scale enterprise initiatives, governmental and commercial, have pursued this "grand vision" of integration. Such visions typically include such things as:

- A collection of new and legacy systems and data
- A universal central vocabulary (or global schema, or ontology)
- Metadata describing the individual connected systems in terms of this ontology. This metadata must be kept current.
- Smart (actually, brilliant) brokering and mediation to connect requesting applications with exactly the data sources they need

Success stories in achieving such visions seem rare or nonexistent. The problem is that, while such visions are fine as goal statements for an ultimate future, they are too abstract to provide useful direction for those who have to build the system. In particular, the "vision" by itself suggests an "all or nothing" approach, and since "all" cannot be achieved in any finite timeframe, the result is often nothing more than a relatively small number of hard-coded connections.

Instead of leading one more charge "over the top", we suggest an approach to achieving integration with features aimed at an imperfect world. In particular, we relax the requirement of universal transparent connection, in favor of more modular and achievable goals. Our basic premise is that, since diversity, change, and "being in transition" are the only sure requirements for 2010, the data architecture should address them directly. We first identify more precisely what we want to achieve, then examine some

[1] This work was presented at the AFCEA Federal Database Conference, Aug. 2000. This version is somewhat improved from the conference version, but is undergoing further revisions.

fallacious assumptions that lead to failure, and describe steps toward a strategy for overcoming them. Our basic principles are:

- For each resource, make it easy to add and maintain connections to it. Ease of creating new connections makes the organization more adaptable. This formulation of the problem looks outward from individual systems, rather than globally. It enables progress to be driven by local efforts, which consist mainly of providing thorough, on-line self-descriptions.
- Metadata (e.g., self-description) providers must benefit soon from creating accurate, current metadata. Metadata created by management fiat will be too vague or obsolete to support transparent access.
- The initial architecture should support more than read-only access. Initial implementations may only provide read access, but this should not be a built-in assumption of the architecture. A consequence of this is that mappings should be declarative, and at a high-enough level so that tools can reason about them.

## 2 What we want to achieve

The first stage in a practical approach is to identify the specific aspects of the general goal that are most important to those who must implement it. Only then can we reasonably measure success or failure. Specifically, then, the system must take into account that:

- There will be many sources of data, and very many consumers
- Some sources and consumers are external to the enterprise – and outside its control
- Even internal sources and consumers may exercise high autonomy over their affairs
- *Presentation* to users is not enough; support for application-level integration (where one automated system acts on data supplied by another) is required
- Simple *access to data* is not enough; some *data integration* is required. For example, some fusion should be available to resolve inconsistent data values that may be returned from multiple sources.

There are, of course, many other requirements that might be included in such a list (e.g. security), but we limit ourselves here to those that are important for integrating data.

## 3. Why we fail

While many organizations, governmental and commercial, have tried to integrate their data, we do not know any success stories that satisfy all of the above requirements. The lack of success is often blamed on a lack of commitment and resources, but we suspect that doubling the budget would probably lead to a failure twice as costly. The deeper reasons lie in fallacious assumptions (often unconscious) on the part of the builders and their management. (Post mortems are too rarely published, so our conclusion rests on some experience plus widespread folklore and audience responses when the issues are raised.) In this section we examine some of these faulty assumptions and how they can lead to failure.

<u>The work will eventually be "finished"</u>

Completing a task is a good thing, especially in the eyes of managers. But data integration for a large enterprise is much more like a process than a destination – continuous change is certain. Believing in an end-state where the work will be "done" tends to produce one of two problems. On one hand, it can encourage a *static* system architecture that meets the milestones of the current objective but which cannot be updated to meet new objectives. Systems built in this fashion are the legacy systems of tomorrow. On the other hand, it can encourage a *future-objective* architecture that cannot accommodate the systems we have today, and therefore will not help with incremental transitions from the systems we have today. What we need is an architecture that helps us during the present transition, and one that keeps helping us with the transition to come. In short, since transition is a permanent feature, we should provide support for incremental improvement, as a permanent feature.

A single data standard is possible

Data integration requires that producers and consumers interpret the same data in compatible ways. One way to approach this problem is to insist that all participants adopt the same standard data models and data element definitions – either internally, rebuilding entire systems, or at the external interfaces. It is certainly good to avoid unnecessary diversity by defining community standards. But it is a mistake to believe that a single data standard will serve a large enterprise with many autonomous participants in a world of continuous change. The consequences are inflexible, tightly-coupled systems, incorrect data integration, and missed opportunities for correct integration. Multiple standards are inevitable, so we should look for ways to both minimize the number of standards, and help system builders deal with more than one.

Data standardization is sufficient

Data standardization allows a consumer to *understand* data from any producers who conform to the standard(s). However, it does not help the consumer to discover which producers have relevant data or to understand what portion of the desired data is actually available. Nor does it help the consumer to recognize and reconcile multiple reports about the same real-world entity. (Does "John Public" in one source refer to the same person as "John Q. Public" in another? If so, what happens if the sources disagree about his birth date?) The problem lies in the instance data collected by systems, not in the data definitions built into those systems. The solution must involve the people who operate the systems, and not just the builders.

Mandates will accomplish what we intend

Data integration for a large enterprise necessarily involves collecting high-quality metadata from many participants (e.g., good descriptions of what their systems provide). It is a mistake to assume that these people will provide good metadata simply because they are ordered to do so. Mandates for metadata are typically all pain and no gain for the people who bear the costs – and while the collecting authority can check the *form* of the responses, they cannot check the *contents*. The result is predictable: metadata that satisfies the format rules but does not provide useful information about the real system it purports to describe.

People respond to real incentives, not hopeful expectations, and so any means of progress will either be a carrot or a stick. A better stick is not worth much here – high-level stick-wielders have too many other tasks, the people for whom the task is low priority are too numerous, and only the latter really know whether or not they've made a serious effort to provide good metadata. We will take up the question of incentives in a later section.

Infrastructure funding is wasted funding

Infrastructure funding often appears to be wasted funding because it does not promise to deliver new functionality to the end user. Individual systems are funded because they do. This works to the extent that all the desired functionality can be completely divided up among the separate systems. But some sorts of functionality – including the data integration that is our goal – necessarily spans the boundaries of fielded systems and the management structures used to build them. This work falls into the twilight zone between systems; it is nobody's work because it is everybody's work. Organizations that only pay for separate systems will only *get* separate systems. Some tasks require cooperation across system boundaries; these must be funded across system boundaries if they are to receive the management attention they need to succeed.

## 4. How to quit failing

Inability to provide widespread access is often blamed on inadequate resources. We assign equal blame to fundamental design flaws based on the unacknowledged "assumptions" of section 3. The old

version obscured two important truths: (1) the enterprise is built in autonomous system-sized chunks, and (2) it takes effort to make these chunks fit together.

In this section, we present an approach based on more realistic assertions about the context in which the system must be constructed and operated. First, we give an overview of our general solution approach, and briefly describe the characteristics of the target system. Then we provide more details of key individual parts of the approach.

### 4.1 Solution Overview

Our solution aims to make sense of current resources for interoperability, to provide an environment that exploits whatever interoperability resources are available, and to encourage development of new ones.

**4.1.1 Reformulate the Objective.** The first step in how to quit failing is to adopt a more realistic objective. Instead of the "grand vision" where everyone can get everything from everybody, we suggest an adaptable system that allows the enterprise to work with known partners, and that also has the flexibility to work with unknown future partners. Furthermore, this flexibility means that instead of meeting all possible requirements (providing universal access), we can position ourselves to meet new ones as they arise. A good analogy is that DoD doesn't have troops everywhere, but instead has an infrastructure so that it can change its deployments as new needs arise.

This approach aims at an *adaptable, locally-improvable* system, rather than a *perfect, universal* system. It anticipates that such a system is based on *high quality* metadata that describes sources, services, and data requirements. This metadata needs to be *active,* i.e., used to run the system. Otherwise it can start erroneous and soon become obsolete. Finally, we want to *obtain short-term benefits from each investment.* Managers are rightly skeptical of calls for large investments with payoff only in the far future. We accept the discipline of giving at least some short term benefits, in order to get feedback if we go off course, and to keep stakeholders interested.

Defining this objective more precisely involves identifying known adaptability requirements, such as:

- *The system will have to evolve through multiple versions*. The system model should include the current state, and allow upgrades to key software components (such as database management systems and infrastructure services)
- *The system will need to incorporate additional clients (customers) and additional sources of information.* Some will be similar to existing constituents, and others (e.g., in Kosovo, the Russian army and Oxfam) will see the world quite differently.
- *New interactions will be desired among existing communication partners.*

In fact, the obligation to provide some additional communication each year (while pursuing the goal architecture) should provide feedback on what works and what doesn't.

A change of metaphor may also help. In many discussions, we use the metaphor that we want systems to meet their data needs as if plugging into a "power grid" with a "wall plug". There is an important element of truth here; namely that each individual system and consumer should be designed to plug in to the overall system, without necessarily knowing who else will be connected. This is the only way to scale up. However, a wall plug is too simple an analogy when considering system interface requirements. A better analogy would be some of the interfaces on the back of a typical computer. They have numerous pins, and require agreements about what flows through each pin, as seen in Figure 1. If conversion between one type of connector and another is required for a given interface, what flows through each pin must be described, and a transformation worked out for each flow. In other words, *interfaces connecting systems must be defined in terms of multiple aspects, each one of which is important.*

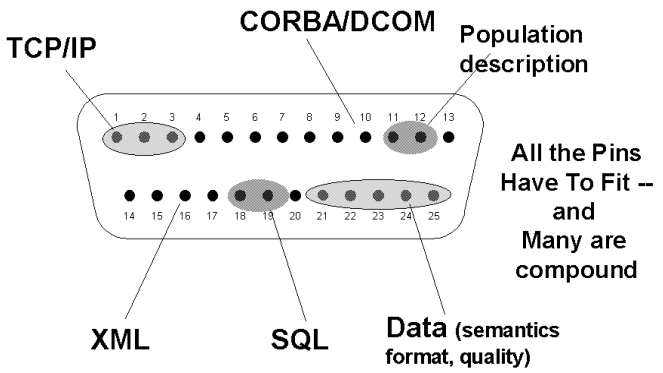


Fig. 1 A better interoperability metaphor – a multi-pin connector.
Solutions must deal with *all* pins

**4.1.2. Partition the Problem (Divide and Conquer).** Creation of a large system is a daunting task that requires decentralization. This means that the building process should be partitioned, ideally so that each task is done by the most appropriate group. We aim to fit an acquisition process that has stakeholders with parochial interests, and funding in various pots, notably for:

- Individual systems
- Funded efforts to connect systems in order to meet an identified need
- Families of systems (e.g., PEO)
- Coalitions drawn from the other categories, to meet perceived common needs

At the 100000 foot level, this resembles DOD.

Specifically, we suggest that the systems building process should identify:

- *What each system needs to do individually.* The main responsibility will be to provide self-description (e.g., what terms does this system use, what services or data does it provide or use). Where possible, these descriptions should use existing vocabularies, from one or another community. (XML namespaces potentially provide a suitable way to reference these vocabularies).
- *What each cross-system development project needs to do.* These are projects that are targeted at creating specific inter-system connections. Such projects are in a particularly good position to contribute partial system descriptions (developed as a necessary part of establishing the connections) that can be reused later in developing other connections.
- *What domain coalitions must do.* These must develop or select (and control evolution of) widely used vocabularies, e.g., common terms with agreed meanings. (This is something that eBusiness, health care (HL7), and other consortia are increasingly engaged in developing).
- *What technical management of the overall combination of systems must do*. They must provide a technical architecture that identifies, among other things, various facets or aspects in terms of which interoperability can be defined and assessed. They must also identify what must be funded as technical infrastructure, e.g., tools, brokers, repositories, etc. Coordination efforts (e.g., to reach interoperability agreements) must also be funded and managed.

Breaking up the work in this way insures that work is performed by those best able to perform it, and also identifies those costs which will have to be assumed by the overall system rather than by the constituent programs.

**4.1.3 Description of the Target System.** The target system constructed in accordance with these suggestions would have the general architecture shown in Figure 2, which shows the necessary processes. The architecture resembles that found in many research papers on large-scale federated systems. The difference is in the idea of facilitating diversity and change by means of pervasive self-description.

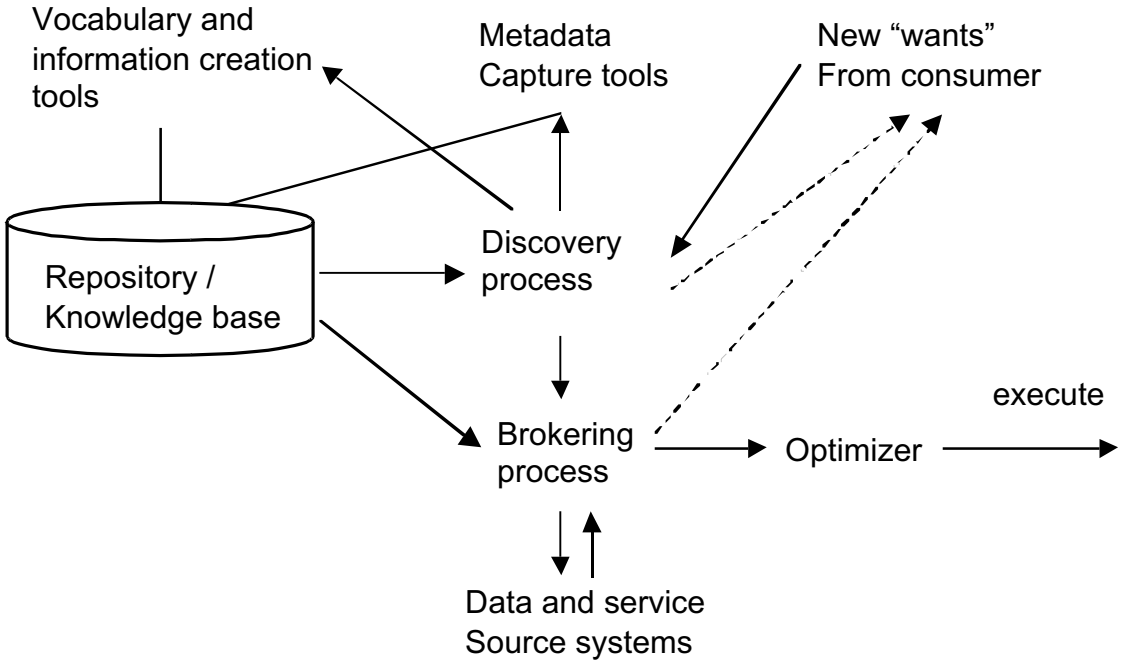


Fig. 2: Processes for meeting a new consumer requirement for data

Today, the technology for many of the processes consists of human labor – interviewing users to find data semantics and other metadata that helps describe systems, and configuring an efficient executable system. Ideally, these processes would be largely automated, but that is a long time off. Hence the architecture is intended to support a mix of human and automated work. The architecture has the following constituents:

- Systems that are data and service providers and consumers
- Metadata, including:
  - Descriptions of each source and consumer. The description of what a consumer wants uses the same formalism as the description of what a source provides. Brokering will compare the two. (Note that this is a higher level of abstraction, compared with having just source descriptions. The latter case usually requires that each consumer manually establish connections to a (possibly undocumented) consumer input interface. )
  - Descriptions of available transformation software (e.g., among data representations)
  - Vocabularies used by various sources and consumers
  - Inter-vocabulary relationships (e.g., that one term is a generalization of another)
- A repository to store and search the metadata, together with tools to provide interfaces to popular products (e.g., spreadsheets, DBMSs, request brokers, CASE tools and other repositories). These

interfaces provide a means of importing metadata that has already been collected for other purposes. This "repository" needs to be highly flexible in its support for searches and assimilation of structured, semi-structured, and unstructured information from multiple sources. It must be widely accessible (ideally, Web-accessible). We are not aware of existing products that completely meet these needs.

- A brokering process (including both manual and automated processes) which:
  - Establishes connections between sources and consumers based on the provided metadata describing them
  - Captures additional metadata from new matches, service discovery, consumer registration, or ad hoc creation by users (e.g., new vocabularies)
- An optimizer that determines the most efficient way to derive the result from the sources, and compiles the code (produced by multiple mediators) to run faster. (It may combine tasks into one process, push tasks into data servers, exploit multicasting, create intermediate caches, and so forth).

The metadata that must be provided should include information on several aspects of each resource, including:

- The semantics (meaning, e.g., in English) and representation of each data element and group of data elements (record or object).
- The signature (interface description) and semantics of each service (or object method) that can be invoked
- The scope and completeness of the data or service provided (e.g., that one system provides information on *all US* fuel depots *since 1970*, while another provides information on *some NATO* fuel depots *since 1990*). This is the sort of information about a system that tends to be *implicit*, but needs to be made *explicit* when the system is made part of a larger system and hence accessible by those not familiar with its contents.
- Delivery style (information push vs. information pull, whole vs. changes)
- Quality of service, including such things as data quality, timeliness, attribution, completeness, obligation (of the service to continue to support the service), cost, etc.
- Guidance for data merging (information on how to match with data from other sources, conflict resolution)
- Other server information, including server address, access language, details of protocols needed to use the service (e.g., how to login), security domains, etc.

To illustrate how this system would work, consider what happens when the system gets a new request from a consumer:

- The system must first *discover* if any known source offers this information.
  - One can check the repository to see if any source uses the consumer's vocabulary. (This is very easy to automate). If this is not enough:
  - The system can next check cross-vocabulary relationships and population relationships to see if any system has registered anything formally known to fit, or be close (e.g., a generalization). (This is fairly easy to automate). If this is not enough:
  - One can examine formal descriptions that are less accessible (e.g., not in a central repository).
  - The discovery process can also examine textual documentation to find additional resources. (For example, a programmer might do this). The human doing this work may discover new inter-vocabulary connections (e.g., if System1.MISSION.MissionDate and System2. MISSION.Day mean the same thing). In this case, the connections should be documented.
  - The humans negotiating about data availability may wish to define vocabularies for others to use. These should also be documented.
- The system must then *broker* differences between the discovered resources and the desired ones. (Filtering, fusion, and transformations may be needed.)

- One must then *optimize:* Given the resulting derivations for the desired connections, determine where to cache. (For example, what tables should be kept in a warehouse?). Then determine where each operation should execute (e.g., as part of extraction query, or in a separate translation module?). Today, most of this optimization will be manual; this may gradually improve.

It is important to realize that the steps we describe here (source discovery, brokering differences, and so on) are a normal part of establishing a connection between two systems. That is, they describe functional tasks that must be accomplished. We are merely proposing that the steps themselves should be part of a defined methodology, with gradually increasing support from software components, and that the results of these steps should be captured and documented by the system. In particular, support is provided for the *incremental* capture, either manually or automatically, of new metadata describing matches, vocabularies, and anything else of interest to the system. This is the means of incrementally extending and adapting the system in the face of changed requirements, composition, or circumstances.

**4.2: Tasks and Incentives to Participate**

Our goal is a spiral approach, which provides for both immediate benefits, and longer-term extensibility. In this section we will discuss in more detail the steps to be taken by individual systems, by individual cross-system development programs, by domain coalitions, and by overall technical management. In each case, we describe the benefits they provide for interoperability, and the benefits (i.e., incentives) to the ones doing the work. We also mention what infrastructure they require, but probably cannot be expected to fund themselves. The intended use of this information is to help managers decide what steps to take first, and how to get more mileage from the work that they are currently funded to do.

This paper is an evolving work, and this section is evolving fast. Later versions will appear at http://www.mitre.org/resources/centers/it/staffpages/arnie/.

What each system needs to do individually

**Step:** *Describe your own system in whatever vocabulary you find convenient.* One need not describe the entire system, just the portions for which one is currently changing or extending external interfaces.

Good formalism choices might include conforming to an XML schema, or using a form provided by your friendly neighborhood repository. But at a minimum, capture it in a somewhat structured form (using a set of defined fields). Word or Excel templates might be used for this purpose, with the data exported later into more generally-searchable formats.

In some areas, you may be able to pick a suitable vocabulary standard. Candidates include existing standards such as DoD 8320 for representation metadata (e.g., how to say "distance-unit = miles"), or Dublin Core for simple document metadata. Other candidates include standards being developed by consortia, and interfaces defined by popular products.

**Benefits (for interoperability):** Data producers and consumers will be able to discover each other more easily, by reading published information. They will no longer need to dig into internal system documentation, or developers lore. If the information is provided in searchable form (as web pages, or preferably in a repository supporting Web-like flexible search capabilities), then good discovery tools could be provided.

If the infrastructure includes comparison and mediation tools, then the benefits can be greater. After discovery, the comparators could identify which aspects match. Mediators (with access to a transform library) could shield integrators from many forms of diversity.

**Incentives to provider:** First, we mention benefits obtainable essentially from making descriptions available on-line. Producers will be discovered by more consumers; consumers who describe their needs can use these descriptions as a search specification. Both sides may have less need to field questions from developers of new applications and new connections. Also, when a system (producer or consumer) has

thoroughly documented its interface semantics, it has taken a step that may simplify training and improve data quality.

With stronger infrastructure, we can reduce the burden of supporting many similar interfaces. A provider would publish one interface. Based on this interface, the infrastructure (mediators, query processors, translators) would generate additional interfaces that use subsets of the data content. For example, instead of a provider being told "you must be able to publish data with HTML, XML, SQL-queryable, CORBA, DCOM, and SOAP interfaces", they might be told "Publish it once, in a form that provides the dataset and (if available) query interface. The infrastructure will take care of the rest."

Similarly, suppose an Air Transport provider (and their communities) had gathered enough metadata to connect to other services' vocabularies. Then they could publish their information once; the system would generate interfaces in other vocabularies and representations, e.g., Army Logistics, Navy Finance.

Another benefit is that a consumer can specify the information they *really* want, even if it is not yet provided. (This requires that the infrastructure manage distinct descriptions for "what is wanted" and "what you get now".) When attributes are added to a relevant exporter's interface, the new match can be detected, and a notification generated. With approval, one might extend the run-time interfaces and GUIs automatically.

What each cross-system development project needs to do

**Steps:** Interoperability is good in the abstract, but progress is often driven by funded efforts to create specifically-needed connections – a consumer needs to connect to a source that provides X. Our goal is for these projects to contribute descriptions that can be used by others. One cannot expect owners of existing systems to hurry to document all their interfaces. At any given time, many interfaces may not have been documented well enough for external users (occasionally called "queries from outer space"). Moreover, the system "owners" may have little funding to improve the descriptions, and may have difficulty doing it well without real "customer" feedback.

Connection-builders capture the same sorts of information that system owners might capture (if motivated), but only for the information required in the connection. We would like to see such knowledge explicitly captured, in a form that enhances the existing metadata and can later be reused. (The infrastructure should thus make it easy to provide and to exploit partial documentation, opportunistically).

Often the connection will be based on the discovery of semantic relationships, either "equivalence" (i.e., fully describes) or **can_use** (i.e., this data fits within an existing category, but has additional assumptions). At times, connection builders will extend or even create vocabularies. At other times, the new metadata describes representation. When a transformation is coded, its capabilities should be documented so it can be added to a reusable library. Connection builders often build glue that transforms formalisms (SQL to USMTF) or elements.

**Benefits to Interoperability:** The metadata captured is similar in type (if not in scope) to metadata captured by system builders. The potential benefits are the same – semi-automated generation of new interfaces.

Compared with owners of individual systems, connection-builders are often in a better position to contribute toward interoperability. By definition, they understand more than one system's perspective. Also, their descriptions have real customers and will tend to be more accurate – because humans and software use them to interpret and transform strange data. Connection builders will also contribute knowledge about how data elements in different vocabularies relate to each other, and may contribute transformation code. If managed well, these can be reused.

**Incentives:** The infrastructure should help in capturing descriptions of interfaces and relationships. This support could begins with GUIs and repositories, plus help in loading information from multiple forms (spreadsheets, other databases and dictionaries, and so forth). More intelligent tools (e.g., schema comparators) can also be provided. With such tools, it becomes easier to separate the analysis and the development tasks, and to reduce the skills required in each role. The developers would need to acquire

less knowledge of the legacy systems, while the analytical work requires much less programming skill. The infrastructure should help the connection builders do what they need to do to build the connection. The incentive for connection builders is that the infrastructure eases the tasks connection builders do, while at the same time capturing useful metadata in reusable form.

As one added greater mediation capabilities, tools could exploit the metadata to generate some of the required interfaces. For example, if a consumer advertised for information in XML and the source provided SQL, the tool (rather than the database guru) would generate the request to convert to XML. Similarly, the tool could provide an interface in which element name changes and representation conversions had already been done. But perhaps the biggest saving would be in maintenance. Now a change to one aspect of an interface should require coding at most for that aspect's connection; at best, the connection could be regenerated automatically from the new description.

What a Domain Coalition (or Community of Systems) must do

**Steps:** Define some common vocabulary, for issues of central concern to your community. Make it available on-line. Perhaps provide some validation aids, to help systems know that they really do conform to the intended semantics of a term.

On issues of peripheral concern, find and recommend (choose) some external vocabularies (one or two in each area). Some domains might later go beyond just a vocabulary, coming up with agreed-on process descriptions or object models (as eBusiness consortia are doing).

**Benefits for interoperability:** If well done, both the definitions and the choices can reduce unnecessary diversity. They also serve as the basis for automated comparison ("are these two really compatible in all ways") and automated mediation of differences. Major interoperability improvements can be achieved by vocabulary resolution activity. It is an opportunity to resolve many of the hard issues, and to share information about common problems throughout the domain (even beyond data integration).

**Benefits for provider:** Some resources (e.g., glossaries, training) can be shared. The effort of building future connections will be reduced.

Infrastructure Requirements

The infrastructure will be built gradually, as the business case can be made and as technology advances. It must coordinate closely with the metadata-capture strategy, to help all stakeholders gain value from their metadata. Infrastructure may be acquired by large systems and groups of systems. As with other parts of the system, we anticipate diversity. For example, we cannot imagine a single portal product being suited to all DoD needs.

We now describe some of the first steps for infrastructure providers (in each community of systems). We exclude areas (e.g., brokering algorithms) that require extensive implementation and will need to be obtained from vendors, when available.

One high priority is to establish interoperability for the metadata that is used to establish interoperability. The amount is not enormous, and the legacy is small. We need "good enough" definitions of the metadata to be captured, plus (crucially) tools that use these definitions, in order to make it easier to use the standard than to go one's own way. *Capture* tools will be needed first, but we must very quickly provide *exploitation* tools that give benefits to the providers.

First, the infrastructure must define the set of "pins" or "aspects" to be described, and provide a formalism for describing each. (Additional aspects will be added later, of course.) In some cases (element semantics, formats, units) the infrastructure should define a vocabulary of terms. It will be important to minimize diversity by establishing a schema to hold descriptions of relationships among vocabulary elements.

The infrastructure should also aim to create new, higher-level interfaces. In general, rather than explain to developers what each mediator does, we want to describe the virtual machine it supports. For example,

- If one has representation metadata, a library of transforms, and a "representation" mediator (all suitably integrated), then developers can see "abstract" attributes like Airspeed or TargetPhoto, without being concerned about how the data is represented.
- If one has vocabulary mappings, a developer can see many elements from foreign systems under their local names.
- If one has population descriptions, then a developer can see an interface that has pulled in all available instances relevant to their declared interests.

Further research is needed in a number of areas. Some features (e.g., mediation algorithms) are well studied in academia and are gradually appearing in products. But there will be gaps. Certainly our sketch of "how to describe a data requirement" is incomplete, and needs to be extended. (For example, where would one document whether there is a continuing obligation to provide certain information? Or, how to distinguish "real" consumers' needs from intermediaries whose services are no longer needed? Quality of service has received much attention but still seems immature.

## 5. Conclusion

The grand vision of universal transparent data access is fine as a goal statement, but should not obscure the need to build systems that provide current value, can be enhanced incrementally, and give incentives to those who must implement them. In this paper, we have described why attempts to satisfy "grand visions" generally fail, and suggested ways to keep from failing, while making realistic progress toward the future of systems that are never really "complete", but continually get better. .

## Author Biographies

Arnon Rosenthal has worked in recent years on data administration, interoperability, distributed object management, migration of legacy systems, and database security, at the MITRE Corporation, Bedford MA. At Computer Corporation of America, ETH Zurich, and the Univ. of Michigan, his interests included active databases, query processing, database design, and discrete algorithms. He holds a Ph.D. from the Univ. of California, Berkeley. http://www.mitre.org/resources/centers/it/staffpages/arnie/

Frank Manola is a Senior Principal Staff member at the MITRE Corporation. His research interests include Web and distributed object technologies, database systems, and mobile agents. Previous employers include the Naval Research Laboratory, Computer Corporation of America, and GTE Laboratories. He holds degrees from the University of Pennsylvania and Duke University.

Scott Renner is a Principal Engineer in MITRE's Center for Air Force C2 Systems. He is the lead scientist for the Common Data Environment (CDE) office at the Electronic Systems Center (ESC/DI). He holds a Ph.D. in computer science from the University of Illinois at Urbana-Champaign.