Enterprise Information Integration: Successes, Challenges and Controversies

Alon Y. Halevy[*](Editor), Naveen Ashish[†], Dina Bitton[‡], Michael Carey[§], Denise Draper[¶], Jeff Pollock[‖], Arnon Rosenthal[**]

ABSTRACT

The goal of EII Systems is to provide uniform access to multiple data sources without having to first loading them into a data warehouse. Since the late 1990's, several EII products have appeared in the marketplace and significant experience has been accumulated from fielding such systems. This collection of articles, by individuals who were involved in this industry in various ways, describes some of these experiences and points to the challenges ahead.

1. INTRODUCTORY REMARKS

Alon Halevy
University of Washington & Transformic Inc.
alon@cs.washington.edu

Beginning in the late 1990's, we have been witnessing the budding of a new industry: Enterprise Information Integration (EII). The vision underlying this industry is to provide tools for integrating data from multiple sources *without* having to first load all the data into a central warehouse. In the research community we have been referring to these as data integration systems.

This collection of articles accompanies a session at the SIGMOD 2005 Conference in which the authors discuss the successes of the EII industry, the challenges that lie ahead of it and the controversies surrounding it. The following few paragraphs serve as an introduction to the issues, admittedly with the perspective of the editor.

[*]University of Washington
[†]Nasa Ames
[‡]Callixa
[§]BEA Systems
[¶]Microsoft Corporation
[‖]Network Inference
[**]Mitre Corporation

SIGACM-SIGMOD '05 Baltimore, Maryland USA

Several factors came together at the time to contribute to the development of the EII industry. First, some technologies developed in the research arena have matured to the point that they were ready for commercialization, and several of the teams responsible for these developments started companies (or spun off products from research labs). Second, the needs of data management in organizations have changed: the need to create external coherent web sites required integrating data from multiple sources; the web-connected world raised the urgency for companies to start communicating with others in various ways. Third, the emergence of XML piqued the appetites of people to share data. Finally, there was a general atmosphere in the late 90's that any idea is worth a try (even good ones!). Importantly, data warehousing solutions were deemed inappropriate for supporting these needs, and the cost of ad-hoc solutions were beginning to become unaffordable.

Broadly speaking, the architectures underlying the products were based on similar principles. A data integration scenario started with identifying the data sources that will participate in the application, and then building a *virtual schema* (often called a *mediated schema*), which would be queried by users or applications. Query processing would begin by reformulating a query posed over the virtual schema into queries over the data sources, and then executing it efficiently with an engine that created plans that span multiple data sources and dealt with the limitations and capabilities of each source. Some of the companies coincided with the emergence of XML, and built their systems on an XML data model and query language (XQuery was just starting to be developed at the time). These companies had to address double the problems of the other companies, because the research on efficient query processing and integration for XML was only in its infancy, and hence they did not have a vast literature to draw on.

Some of the first applications in which these systems were fielded successfully were customer-relationship management, where the challenge was to provide the customer-facing worker a *global view* of a customer whose data is residing in multiple sources, and digital dashboards that required tracking information from multiple sources in real time.

As with any new industry, EII has faced many challenges, some of which still impede its growth today. The following are representative ones:

Scaleup and performance: The initial challenge was to convince customers that the idea would work. How could a query processor that accesses the data sources in real time have a chance of providing adequate and predictable

performance? In many cases, administrators of (very carefully tuned) data sources would not even consider allowing a query from an external query engine to hit them. In this context EII tools often faced competition from the relatively mature data warehousing tools. To complicate matters, the warehousing tools started emphasizing their *real-time* capabilities, supposedly removing one of the key advantages of EII over warehousing. The challenge was to explain to potential customers the tradeoffs between the cost of building a warehouse, the cost of a live query and the cost of accessing stale data. Customers want simple formulas they could apply to make their buying decisions, but those are not available.

Horizontal vs. Vertical growth: From a business perspective, an EII company had to decide whether to build a horizontal platform that can be used in any application or to build special tools for a particular vertical. The argument for the vertical approach was that customers care about solving their *entire* problem, rather than paying for yet another piece of the solution and having to worry about how it integrates with other pieces. The argument for the horizontal approach is the generality of the system and often the inability to decide (in time) which vertical to focus on. The problem boiled down to how to prioritize the scarce resources of a startup company.

Integration with EAI tools and other middleware: To put things mildly, the space of data management middleware products is a very complicated one. Different companies come at related problems from different perspectives and it's often difficult to see exactly which part of the problem a tool is solving. The emergence of EII tools only further complicated the problem. A slightly more mature sector is EAI (Enterprise Application Integration) whose products try to facilitate hooking up applications to talk to each other and thereby support certain workflows. Whereas EAI tends to focus on arbitrary applications, EII focuses on the data and querying it. However, at some point, data needs to be fed into applications, and their output feeds into other data sources. In fact, to query the data one can use an EII tool, but to update the data one typically has to resort to an EAI tool. Hence, the separation between EII and EAI tools may be a temporary one. Other related products include data cleaning tools and reporting and analysis tools, whose integration with EII and EAI does stand to see significant improvement.

Meta-data Management and Semantic Heterogeneity: One of the key issues faced in data integration projects is *locating* and *understanding* the data to be integrated. Often, one would find that the data needed for a particular integration application is not even captured in any source in the enterprise. In other cases, significant effort is needed in order to understand the semantic relationships between sources and convey those to the system. Tools addressing these issues are relatively in their infancy. They require both a framework for storing the meta-data across an enterprise, and tools that make it easy to bridge the semantic heterogeneity between sources and maintain it over time.

Summary: The EII industry is real – in 2005 it is expected to have revenues of at least half a billion dollars. However, it is clear that the products we have today will have to change considerably in order for this industry to realize its full potential, and its positioning still needs to be further refined. I personally believe that the success of the industry will depend to a large extent on delivering useful tools at the higher levels of the information food chain, namely for meta-data management and schema heterogeneity.

2. TOWARDS COST-EFFECTIVE AND SCALABLE INFORMATION INTEGRATION

Naveen Ashish
NASA Ames
ashish@email.arc.nasa.gov

EII is an area that I have had involvement with for the past several years. I am currently a researcher at NASA Ames Research Center, where data integration has been and remains one of my primary research and technical interests. At NASA I have been involved in both developing information integration systems in various domains, as well as developing applications for specific mission driven applications of use to NASA. The domains have included integration of enterprise information, and integration of aviation safety related data sources. Previously I was a co-developer for some of the core technologies for Fetch Technologies, an information extraction and integration company spun out of my research group at USC/ISI in 2000. My work at NASA has provided the opportunity to look at EII not only from a developer and provider perspective, but also from a consumer perspective in terms of applying EII to the NASA enterprise's information management needs.

A primary concern for EII today regards the scalability and economic aspects of data integration. The need for middleware and integration technology is inevitable, more so with the shifting computing paradigms as noted in [8]. Our experience with enterprise data integration applications in the NASA enterprise tells us that traditional *schema-centric* mediation approaches to data integration problems are often overkill and lead to overly and unnecessary investment in terms of time, resources and cost. In fact the investment in schema management per new source integrated and in heavy-weight middleware are reasons why user costs increase directly (linearly) with the user benefit with the primary investment going to the middleware IT product and service providers. What is beneficial to end users however are integration technologies that truly demonstrate economies of scale, with costs of adding newer sources decreasing significantly as the total number of sources integrated increases.

How is scalability and cost-effectiveness in data integration achieved? Note that the needs of different data integration applications are very diverse. Applications might require data integration across anywhere from a handful of information sources to literally hundreds of sources. The data in any source could range from a few tables that could well be stored in a spreadsheet to something that requires a sophisticated DBMS for storage and management. The data could be structured, semi-structured, or unstructured. Also, the query processing requirements for any application could vary from requiring just basic keyword search capabilities across the different sources to sophisticated structured query processing across the integrated collection.

This is why a *one-size-fits-all* approach is often unsuitable for many applications and provides the motivation for developing an integration approach that is significantly more

nimble and adaptable to the needs of each integration application. We begin by eliminating some tacit, schema-centric assumptions that seem to be holding for data integration technology, namely:

- Data must always be stored and managed in DBMS systems,
- Actually, requirements of applications vary greatly ranging from data that can well be stored in spreadsheets, to data that does indeed require DBMS storage.
- The database must always provide for and manage the structure and semantics of the data through formal schemas.
- Alternatively, the "database" can be nothing more than intelligent storage. Data could be stored generically and imposition of structure and semantics (schema) may be done by clients as needed.
- Managing multiple schemas from several independent sources and interrelationships between them, i.e., "schema-chaos" is inevitable and unavoidable.

  Alternatively, any imposition of schema can be done by the clients, as and when needed by applications.
- Clients are too light-weight to do any processing. Thus a significant component of integration across sources must be, in a sense, *pre-compiled* and loaded into a centralized mediation component.

  This assumption is based on a 1960s paradigm where clients had almost negligible computing power. Clients of today have significant processing power and sophisticated functionality can well be pushed to the client side.

Many enterprise information integration applications do not actually require a schema-centric approach. A system called NETMARK, developed at the NASA Ames Research Center actually demonstrates the feasibility of a lean approach to information integration where schema-imposition is done minimally and only to the extent needed. The essence of the approach is that data is managed in a *schema-less* manner; also most computational or aggregation functionalities required are pushed to the client side thus reducing the mediator to a mere router of information. The technical details of this approach are presented in works such as [4, 5] and the claim is that this "schema-less" approach to information integration actually leads to scalability and cost-effectiveness in enterprise information integration with an increasing number of sources. Note that the argument is not against schema based approaches in-toto. There are and certainly will be applications where schemas already exist, and are in fact absolutely needed, and where a formal schema and mediated approach will be required to achieve data integration. Our argument is that on the other hand, there is a significant space of integration applications that do not require the use and investment in schemas.

The key distinguishing features of the NETMARK approach are that:

- Data is managed in a schema-less system, eliminating the need for schema management and database administration.
- Commonly used business documents (MS Word, Excel, PowerPoint) are the interface to the integrated pool of enterprise information. The above system and approach have been used very successfully in many NASA enterprise information applications. From an economic and scalability perspective the claim made is the following:
- Traditional schema-centric approaches to information mediation are inhibiting the scalability and cost effectiveness of information integration for many enterprise applications.
- A lean, schema-less approach to information integration works successfully for a major space of applications leading to scalable and cost-effective data integration from the user perspective.

3. WHY EII WILL NOT REPLACE THE DATA WAREHOUSE

Dina Bitton
Chief Technical Officer
Callixa, San Francisco
dbitton@callixa.com

In the past fifteen years, data warehousing and its associated ETL (Extract, Transform, Load) technologies have delivered great business values to enterprises looking to integrate and analyze large volumes of disparate customer and product data. The data warehouse has successfully evolved from monthly dumps of operational data lightly cleansed and transformed by batch programs, to sophisticated metadata-driven systems that move large volumes of data through staging areas to operational data stores to data warehouses to data marts. However the forever increasing demand for lower cost and real-time information delivery is leading to the development and adoption of on-demand information integration technologies. For instance, real-time analytics and reporting, resulting from competitive and compliance pressures, are driving the need for a "virtual data warehouse" approach to integrate and present disparate data. Several EII (Enterprise Information Integration) technologies have come to market allowing disparate data sources to be integrated on-demand, without moving and replicating them, and providing a single SQL or Xquery interface to these multiple sources.

As often happens, some argue that this new technology makes the older obsolete, and that EII, even though now in its market infancy, will replace the data warehouse. In this short paper, we argue two points against that view:

- To be viable, EII technologies need to deliver levels of performance and scalability similar to the ones expected from RDBMS. And to adequately measure EII performance, we need a standardized benchmark – a la TPC.
- Even when it matures and succeeds, EII will not replace the data warehouse. Depending on the problem to be solved, integrated data will be persisted into a warehouse, or virtualized through EII.

Performance and Scalability for EII

EII uses queries to collect and integrate data and content from multiple sources. An EII query is a federated query, formulated against an integrated view of the data sources. To execute a federated query, an EII server goes out to the data sources to find the relevant data and processes it into the context of the application view. A simplistic approach that some early EII vendors used, was to pull out the relevant data from all the data sources into an Xquery processor and process it entirely there. To understand why this approach cant provide acceptable performance, just consider the example of a cross-database join query that joins 2 very large tables, from two data sources. Each table would be converted to XML, increasing its size about 3 times, moved across the network and then joined. There are two major performance problems associated with this approach. First a huge amount of data is moved across the network. Second, the join operation in a newly designed Xquery processor will not be optimized. Instead, some advanced parallel processing and query optimization techniques should be applied.

Since the sources reside on different hardware, operating systems and databases, a single query submitted to an EII engine must be decomposed to component queries that are distributed to the data sources, and the results of the component queries must be joined at an assembly site. The assembly site may be a single hub or it may be one of the sources. In some EII products, the hub is an RDBMS instance (e.g. IBM Information Integrator), or an Xquery processor. The component queries are usually executed through data wrappers that push down RDBMS-specific SQL queries to the sources. The nature of the component queries depends on the schema translator, which translates the integrated schema into the data source schemata, and on the query optimizer of the EII engine. Clearly, the more work the component queries can do, the less work will remain to be done at the assembly site.

This brief description of query processing in an EII engine clearly indicates that critical EII performance factors will relate to the distributed architecture of the EII engine and its ability to (a) maximize parallelism in inter and intra query processing; (b) minimize the amount of data shipped for assembly by utilizing local reduction and selecting the best assembly site. These problems have been well addressed by designers of parallel database servers, and similar techniques should be used by EII architects.

Needless to say, query optimization of an EII query is not a simple problem. Unfortunately, pressure to bring to market an EII product in a short time has often resulted in simplistic solutions that will not perform or scale.

EII and Data Warehousing

Assuming that EII will replace data warehousing is also very simplistic. Data warehouses will continue to be built and used for complex analytics. EII will fill the demand for on-demand, rapid and flexible integration in other applications. The question is really when to persist integrated data versus when to virtualize it. Here are a few guidelines that can be applied when making that choice.

Guidelines of Persistence:

1. *Persist data to keep history.*

   Warehouses store historical data that is captured from sources at an interval. As there is no other source for history, warehouses must be in place to store and save historical data.

2. *Persist data when access to source systems is denied.*

   For many reasons (operational, security, or other) a database federating technology may be denied access to a source. In this case the source data must be extracted to a persistent data store such as a data warehouse

Guidelines of Virtualization (Federation):

These virtualization guidelines should only be invoked after none of the persistence guidelines apply.

1. *Virtualize data across multiple warehouse boundaries and virtualize new data marts.*

   Instead of redundantly copying data into multiple data warehouses, virtualize the shared data from one warehouse to another. By definition, a conformed dimension is shared between multiple data marts. Virtualize the conformed dimension to let it match the semantics of the non-conforming dimensions to increase usability. Another scenario where a virtual data mart is a great solution is where a new external data source that had not been included in the design of the warehouse needs to be integrated with an existing data warehouse or data mart.

2. *Virtualize data for special projects and to build prototypes.*

   Data can quickly be assembled for one-time reports or for prototyping new applications by building a virtual schema of the required data.

3. *Virtualize data that must reflect up-to-the-minute operational facts.*

   In applications such as dashboards or portals, data has to reflect the current state of operations. For these applications, virtual integration through EII is a must.

Conclusion

EII potentially offers a more flexible and more inclusive approach to integrating information than data warehousing. However, to perform adequately, EII servers need to build on the vast expertise embedded in parallel database servers, rather than replace them. Schema translation and federated query decomposition should aim at generating component queries that can be pushed down to mature database servers for efficient execution. Finally, even as EII technology matures, it is unlikely to replace the data warehouse because it does not solve the same problems.

4. EAI ... EII ... AYE YI YI!

Michael Carey
BEA Systems
mcarey@bea.com

As discussed in the Introduction to this article, the state of affairs today is that there are two different product categories in the "integration middleware" software space – EAI

and EII. Like several other major software vendors, BEA Systems has a product offering in each category.

For EAI, BEA offers a product called WebLogic Integration. A major feature of this product is its support for business process management (a.k.a. workflow management or service orchestration). Connectivity to a wide variety of enterprise applications and information sources is provided through the provision of a large collection of XML-based application adaptors as well as general support for invocation of Web services. Support for data transformation and normalization is provided as well, via a streaming XQuery-based data transformation engine as well as a runtime engine that enables XML-based access to non-XML (binary) data streams. WebLogic Integration also provides message brokering capabilities for enterprise messaging and routing. With these capabilities, including a visual programming environment for specifying business processes, it is possible for BEA customers to access, integrate, update, and orchestrate an arbitrarily wide range of applications and data sources in a procedural (but still relatively high-level) fashion.

For EII, BEA offers a product called Liquid Data for WebLogic. In its currently available form, Liquid Data provides a framework for accessing and integrating data from a wide range of enterprise applications and information sources. (Sound familiar?) Access to data locked inside applications and/or web services is provided via the same connectivity options offered by WebLogic Integration. In addition, of course, Liquid Data provides access to data that resides in relational databases, pushing query processing the the underlying databases when possible. Other supported data sources include database stored procedures, XML files, delimited files, and data from in-flight messages. Data integration is accomplished declaratively through the use of XQuery as a language for combining and transforming data from the various data sources (all of which surface their data as XML data with XML Schema descriptions).

So what's the problem? From the standpoint of a customer trying to solve a real business integration problem, the artificial separation of the integration world into EAI and EII is confusing – which technology should be used when, and why? EAI is "integration complete" in the sense that there's basically nothing that it fundamentally can't do. For example, picture a large enterprise that wants to enable its employees to have web-based access to all of the relevant information about them (address, phone, benefits, office, computer hardware, etc.). Such an employee self-service portal would surely need to integrate information from a number of backend applications and databases in order to obtain a "single view of employee" for presentation and other actions. One solution would be to use EAI technology to construct a business process that accesses the various backend systems, transforms their data into the desired canonical form, and stitches it together into a coherent single view of employee. Updates, such as an address change, could be handled by another business process that notifies the various systems that need to know when an employee's address changes.

Another solution to the "single view of employee" problem would be to use EII technology to create the desired view. For data access, this is actually the preferable approach, as constructing the EAI business process is like hand-writing a distributed query plan. If employee data can be accessed other than by employee id, e.g., "find all employees in department D", "find all employees at location L", and/or "find all employees with computer model M", different query plans are likely to be needed. Twenty plus years of database experience has taught us that it is likely to be much more productive to express the integration of employee data once, as a view of some sort, and then to let the system choose the right query plan for each of the different employee queries. However, there's more to data than reads. What about updates? A virtual database update model is often not the best fit for enterprise integration scenarios. "Insert employee into company" is really a business process, possibly needing to run over a period of hours or days (e.g., to provision an office and a phone and order computer equipment and have that order approved). Such an update clearly must not be a traditional transaction, instead demanding long-running transaction technology and the availability of compensation capabilities in the event of a transaction step failure.

For the reasons cited above, BEA recommends the use of both these technologies as a best practice. EII eases the development of the data access part of enterprise integration applications, while EAI makes possible the kinds of updates and other operations needed to solve a complete business problem. However, this separation is painful today, as using both requires developers to say the same sorts of things at least twice (once for the EII system and at least once for the EAI system, maybe more times, e.g., possibly once per business process). For example, the EII single view of employee query will access and stitch together an employee's data from the various systems of record – the HR database, the facilities management system, and so on – and this integration will be expressed as a query of some sort. In the BEA product family, it is expressed declaratively as an XQuery function. The various EAI update operations will each touch some, possibly many, of the same enterprise systems – but the update operations are expressed via a very different programming model, in BEA's case via a business process definition language called JPD (Java Process Definition, similar to the BPEL language which is now gaining traction in the EAI world). As a result, the developer again has to define the ways in which the data for the single view of employee is mapped to and from the underlying enterprise systems. This is not, by the way, a BEA problem – it is an industry problem. (For example, IBM has even more EAI and EII products that a customer must sift through and combine in order to solve a given enterprise integration problem.)

So what's the bottom line? EII must die – EAI must die – and they must be reborn, together, as simply an EI (Enterprise Integration) solution. EAI alone can do the job, but in a manner analogous to writing database applications in the BC period (Before Codd). EII alone cannot do the job, because the *virtual database* view that current EII products provide fall too far short when asked to stretch beyond simple read-only applications. A combination can do the job, but who wants to say the same thing twice using two different programming models? Customers need Enterprise Integration. It's time for those of us in the database R&D community to pay attention to what they really need from us and start working on solutions that work...!

5. THE NIMBLE EXPERIENCE

Denise Draper
Microsoft Corporation
denised@microsoft.com

In 1999 I started working at a company called Nimble Technology [2]; our goal was to create a query-driven system for integration of heterogeneous data, a technology that later came to be known as EII. Over the next five years we developed a novel EII engine and related tools, and went to market with a solution for the enterprise. We had some successes, but not near as much as we thought we deserved. In this note I'll try to lay why I think that was, and the implications for EII in the future.

5.1 EII, a Gratifyingly Hard Technical Problem

EII (defined as "answering queries over distributed heterogeneous sources"), is one of those really gratifying problems: it has both hard technical challenges, and at the same time a clear, elegant outline: you need to solve the problem of mapping between data models in an extensible way, you need to be able to decompose your queries across multiple sources, you need to be able to optimize queries in this environment, etc. Of course, there is already a long history of work in this area, ranging back to [1] and including substantial research groups such as [3], but it is far from a solved problem.

The Nimble system had novel technology in a couple of areas. The most obvious was that we chose an XML-based data model and query language rather than a relational one, and we did some novel work in XML query processing. But this had less significance for EII, at least at this time, than we originally thought. There is a good argument that XML has a natural affiliation for data integration, based on XML's ability to represent "messy" data structures (missing data, duplicate data, variant representations). However experience with customers taught us that the state of application development (even to the level of putting some data on a web page) has not evolved to the point where the application infrastructure can take advantage of—or even accommodate—"messiness". So while I remain convinced of XML's value in the long run, at this point it's potential was barely being used in the applications people were building, and it doesn't by itself solve many of the other hard problems of EII.

There were some other technical choices we made that I feel were good ones, including:

- Within our SQL adapters, we modeled the individual quirks of different vendors and versions of databases to a much finer degree than I have observed in other systems. This had a decisive impact on our performance on every comparison we were ever able to make, since it meant we could push predicates that other systems wouldn't.

- We used views as a central metaphor for our system. There wasn't too much to this from a technical standpoint (except putting a lot of work into optimization over unfolded views). But it had a profound impact from the perspective of usability. Simply put, integration of data from heterogeneous sources is a pretty messy job, and views allow one both to factor the job into smaller pieces, and to keep and re-use those pieces across multiple queries. This realization points towards one of the main areas I see for EII going forward, which is integration into a greater landscape of modeling and metadata management.

Working with initial customer scenarios, we soon added a couple more features, not part of the "pure" definition of EII, both of which I believe are essential:

- A materialized view capability that allowed administrators to pre-compute views. In essence, the administrator was able to choose whether she wanted live data for a particular view or not. Another way to look at this was as a light-weight ETL system; more on that later.

- A record-correlation capability that enabled customers to create joins over sources that had no simply-computable join key. It turns out that if the data sources are really heterogeneous, the probability that they have a reliable join key is pretty small. Our system worked by creating and storing what was essentially a join index between the sources.

5.2 EII, Not an easy Business

We implemented all the technology outlined in the previous section, and more (a graphical query builder, client-side SDK's, etc.). The raw performance data we saw still had room for improvement, but it was pretty good and within acceptable overhead for a middleware layer. Yet still we found it was difficult to grow a business from this. Why? There are many contributing factors, but I believe the major factor was the difficulty in distinguishing the value of EII as an independent product.

One way to look at this is to consider that EII is in competition with ETL into a warehouse or mart. From the customer's perspective, both address the same problem, the problem of integrating heterogeneous data.[1] Thus for EII to have an advantage, it either has to solve some different class of problem (or the same class but "better"), or it has to be less costly than ETL by some measure. Let's look at both sides of this equation:

One main distinction between EII and ETL is that EII is a "pull" or on-demand technology, while ETL (as I am using the term here) is a "push" technology. So one of the advantages of EII is that the customer can get access to the current "live" data. There are scenarios in which this is very important, but one of the things we were surprised by was how little most customers actually valued live data, especially if their alternatives were fairly low latency (24 hours or less).

Another way EII could provide more value would be if data sources were dynamically discoverable and queryable (compared to the a priori set up and data mapping done in both EII and ETL systems today). To do that at a level any deeper than keyword search requires some ability to automatically map and extract information from sources that is much greater than we have today. Based on the kinds of real-world data integration cases I have seen, I am not

[1]This is a very database-centric way of looking at it. Someone building a web site is more likely to think that EII competes with a bit of script he creates manually, which leads to a somewhat different trade-off discussion.

confident that we will have that kind of capability in reach anytime soon.

On the cost side, there would seem to be a clear advantage for EII: there is no need for a warehouse or other repository for copied data. But at least the cost of the warehouse, and the processing time required to load it, are predictable and can be budgeted for. In contrast, EII is not well understood, and thus unpredictable in performance and load. And there are significant classes of queries to which EII is simply not suited for performance reasons.

Finally, the greatest cost in an ETL model is the human cost of setup and administration: understanding the query requirements, understanding the data sources, building and maintaining the complex processes that clean and integrate the data. But EII has exactly the same costs, for a deep reason: while the tools and languages may be different, ultimately the same logic is being performed.

5.3 So Where to go?

Does this mean that EII is a lost cause? A dead end?

I don't believe so, but I believe that the value of EII will be realized as one technology in the portfolio of a general data platform, rather than as an independent product. The ETL approach to the problem cannot be the final answer because it simply is not possible to copy all potentially relevant data to some single repository. The dynamic of our world has been inexorably towards more and more connectedness, and more and more demand for meaningful access to data, and copying data doesn't scale indefinitely.

In order to meet the demand, we will need the ability to dynamically find, model, and query data sources. Adapters need to disappear, replaced by sufficiently standard interfaces (web services + standard metadata and query interfaces?). Users need to be able to find data that they need (search and navigation are a start), and they need decent mechanisms that then allow them to query including feedback about expected performance. We need ways for dynamic/pull/EII queries to be migrated to high-performance ETL processes—which is easier if they both share the same data model and query/processing language. And we need to help administrators figure out where the high-demand data that should be pre-processed is, or to automatically and adaptively do it for them. When we get to this point, EII and ETL are essentially choices in an optimization problem, like choosing between different join algorithms. Of course we all know how easy optimizers are...

A connected thread to this is to address modeling and metadata management, which is the highest cost item in the first place. Even if end users can go and find their own data, they won't get enough value through that alone. There will always be need for "high value" model creation from trained people who understand data models and query languages deeply. Tools that help them attach logical models to data and track those models through transformations and queries would be of immense help. Simply getting enough of a data modeling standard in place that vendors can actually interoperate would be huge step forward all by itself. And then there the next challenge: handling change (impact analysis and response to changes, such as upgrading applications, which change data models or semantics). Improvements in these areas help both ETL and EII, and preferably treat them together (modeling the relationships and meaning of data separately from the aspect of when and where it is computed).

Some of these items are in place already, some won't be for years, but I expect to see continued progress, and EII will continue to be part of the overall solution.

6. EII GOT THE SECOND " I " WRONG

Jeff Pollock
Network Inference
Jeff.Pollock@networkinference.com

The notion of "integrating" information is fundamentally flawed in all but rare cases. In fact, the second "I" in EII should have been for "interoperability", which implies a loose, not tight coupling of shared data. In order to couple data loosely, and in order for isolated data to become information, formal semantics must be added. But no existing EII tools use formal semantics, so *information integration* accurately describes the current EII space after all.

Ten years ago I wrote my first database driven web site, an HR application for Amgen's internal training courses that used FileMaker Pro and AppleScript to cull data from different HR databases and allow employees to register for upcoming classes. From that point forward I became acutely aware, and deeply involved with the challenges of sourcing data that originates from different systems to new applications. Prior to that I had only dealt with mini and mainframe VAX systems and the occasional client/server type systems common in the early 1990s. Building composite applications, portal type environments, and deploying modern middleware is galaxies away from those old vt420's I used to hack on.

Since those days I have built CORBA, EAI, and many application server based n-tier systems, where the hub and spoke pattern of protocol switching and data integration played out with ever so many forms of syntactic sugar. My years with Ernst & Young's Center for Technology Enablement exposed me to some of the largest and most complicated software environments throughout the Fortune 500. I was at E&Y in 1997 when XML was gaining momentum and, like so many, I bought into the XML hype after the W3C recommended it in early 1998 - back when XML was set to solve world hunger and cure cancer. Remember the glory days of the internet and the rise of XML?

Now with the battle scars of several large system deployments that employed ETL, custom data feeds, EAI, and XML based data interchange - I can finally see that the fundamental hard part is the same - no matter what data syntax or structure you happen to choose.

Enterprise Information Integration, as a market category was branded by the Aberdeen Group in May of 2002. I know and remember this well because at that time I was CTO of a small San Francisco startup that was desperately trying to brand the EII category as "Enterprise Information Interoperability". Our little 30 person company was dismayed when Aberdeen claimed the TLA for their own. At Modulant, our motto was, "it's all about context, baby." We understood then that for information to be efficiently shared, yet still be federated, you had to account for meaning (and that meaning occurs only within a context). We decided then that EII didn't embody that spirit of semantics and therefore we ditched the EII acronym and learned a sound business les-

son the hard way - startups don't claim and name markets, analysts do.

I have a shockingly simple view of today's EII: it only does three things well, and it only comes in three flavors. First, the three things it does well (compared to the myriad of other integration techniques) are: (1) highly efficient federated queries, (2) internal management of common data views, and (3) management of metadata designed for transformations and query. Second, the three flavors it comes in are: (1) relational-based systems, (2) XML based systems, and (3) Object-based systems.

This is all well and good, and in my honest opinion EII has a place in the world, but it still doesn't solve the fundamental problem that I bumped into ten years ago with that simple little FileMaker Pro database website: the data structure contains no formal semantics. Sure, the EII folks will tell you that their internal metadata repositories (they all have them) contain semantics - and they are right! The semantics within Brand X EII tool work for exactly Brand X, no more and no less. You see, software has always had semantics. Meaning in software has always been in the for and do loops we write while traversing arrays, or the Y—N database flags that convey crucially important information to the business objects doing their chores. Yes, semantics have always been in code.

So long as semantics are in compiled software, scripts, and SQL we will forever run into "information interoperability" problems. *Information integration* is doable - write enough code and I will connect every software system anywhere. But then things change.

It's like the painters on the Golden Gate Bridge, by the time they've finished rust proofing one span with that magnificent International Orange coat of rust preventing paint - the other side is withering and they need to start over. So it goes with integrating IT systems. And so it will go indefinitely; until the data contains meaning outside of code and proprietary metadata.

In one sense I could personally care less about what happens to EII as a market space. I can tell you that federated query algebras, view management, and query metadata are irreplaceable contributions for dealing with the structural aspects of information interoperability. As it may be, I will predict that either (a) EII will adopt the foundational technologies of the Semantic Web and survive in name with much improved adaptive information capabilities, or (b) EII core technology will gracefully be folded into a new class of software built on newer data formats with formal semantics and strong connections to XML, objects, and relational systems.

Clearly I've made the gamble professionally and personally with Network Inference that the formal semantics provided by the Web Ontology Language (OWL), and the Resource Description Framework (RDF) as a part of the W3C Semantic Web vision will fundamentally change the old rules of the EII space. Perhaps in retrospect I should have guided Modulant to more forcefully maintain the distinction between "our EII" and "Aberdeen's EII", but after all, it's just a minor semantic difference right?

7. EII: EMBRACE CHAOS, AND THEN GUIDE IT

Arnon Rosenthal
The MITRE Corporation
arnie@mitre.org

Personal experience

MITRE helps the US government and its contractors to assemble vendor products, contractor-produced applications, and administration processes into giant systems, creating all sorts of *enterprise integration (EI)*, not just one new integrated view. Like several other presenters, we take broad EI as the goal. Our consumer-side perspective differs from most researchers, who ask how to build products. We researchers also provide advice and guidance for the large scale administration process, e.g., what should a community data standard include?

Main observations and lessons

In (apparent) decreasing order of popularity, EI techniques include integrated packages (e.g., SAP), messaging, warehouses, and EII (as distributed views). All may exist simultaneously, for overlapping subsets of the data. Note the coincidence – usage seems to be in reverse order of the length of research history.

EII does not, long term, seem a viable stand-alone product. *It's the metadata, stupid!* The enterprise's cost is administration, not vendor software, and it is not tolerable to capture overlapping semantics separately for each product, or for read, update, annotate, and change- notification. A vendor who shares metadata with message mediation, ETL, entity instance resolution and data cleaning offers a much better package. EII companies should prepare to be assimilated.[2]

For a large enterprise like the Department of Defense (DOD), it is also important to share metadata across vendors. The EI community has not eaten its own dogfood – EI metadata is unintegrated. EI standards (e.g., MOF) are neither broad nor widely adopted, and do not even address inter-system relationships. Fortunately, the semantic web offers promise that the system- and inter-system formalisms become the same.[3] OWL emphasizes ontologies, but the same transitive relationships can represent matching knowledge and many value derivations, with inference.

The EII community also needs a better story for service oriented architectures. Our n-tier application systems (e.g., Air Force mission planning) emphasize object schemas and services, seeing data as unintuitive physical tuples in the basement. But the tiers fulfill their insulation mandate too well, keeping useful new data captured at one tier invisible at the others.

EII does not seem popular for mapping data to objects, despite the high costs (especially testing!) of evolving the

[2]Semantic matching, and attribute representation conversion also seem unviable, standalone – small benefit and much interfacing.

[3]As competition, XML schemas are popular, but semantically weak and terrible when there are disagreements about whose elements belong on top. UML class diagrams are sometimes used. Entity-relationship standards (e.g., IDEF1X) seem to have only vendor-specific APIs.

existing data-to-object mappings. Today, programmers often code Read, Notify of changes, and Update methods in a 3GL+SQL. EII typically supports the first, but will become popular only if it helps with the others. It should be possible to generate Notify methods automatically. Update methods (e.g., for Java beans) must change the database so the Read view is suitably updated. These are not terribly complex business processes, but do require semantic choices, e.g., if a constraint is violated, should the request abort, or invoke a repair method? Given the choices, the update method should be generated automatically.

EII tools that interpret at runtime seem unattractive in many military settings. Administrators are unavailable in Iraq, and startups cannot support them 7 x 24, worldwide. Tools that generate standard code (e.g., SQL, X-languages) are more attractive.

The strongest case for knowledge-driven EI tools – ease of change – gets lost in the government's acquisition process. Rather than tell a contractor how to design a system, the government tries to set testable requirements. But we do not know how to write "agility" into contract language. Research question: Provide ways to measure data integration agility, either analytically or by experiment. We want a measure for predictable changes such as adding attributes or tables, and changing attribute representations.

Going forward: Be proactive

EI has been for passive wimps![4] Most EI research merely asks how to exchange data that fate supplies, at most trying to discover it in a wider sphere. But enterprises are not passive – they fund acquisition of new data-producing systems, and select and promote data standards. We need to broaden: from semantic integration to "semantics management," which also includes actively guiding semantic choices in new ontologies and systems – e.g., what concepts should be used as descriptive vocabularies for existing data, or as definitions for newly built systems.

The broader semantics management perspective requires many additions to the research agenda. (We present a couple of highlights here; [6] presents more details.) First, most prior research assumes that higher authority is either all powerful (centralized) or absent (peer to peer). There is a need for research on enterprises, where managers have partial influence. Our customers have learned from painful experience that integration progress happens incrementally by developing and exploiting data standards within focused communities of interest. What services do communities need for declaring and influencing standards, and managing and mapping among large numbers of applications and schemas (for the U.S. Department of Defense, perhaps $O(10^5)$?) Practitioners need researchers to put communities on firmer methodological footing.

Finally, as organizations move toward more agile data integration in loosely coupled environments, there is a need for tools to manage *data supply chains* in the face of evolution and decentralized control. One needs agreements that capture the obligations of each party in a formal language. While some obligation conditions are familiar from ISP's service level agreements (e.g., availability, payment), others are more data-specific. For example, the provider may be obligated to provide data of a specified quality, and to notify the consumer if reported data changes. The consumer may be obligated to protect the data, to use it only for a specified purpose. Data offers opportunities unavailable for arbitrary services, e.g., detecting if an existing agreement covers part of your data and automated violation detection for some conditions [7].

[4]This section was written while on a California sabbatical , where I learned the rational style of my governor, Arnie S.

8. REFERENCES

[1] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. R. Jr. Query processing in a system for distributed databases (sdd-1). *ACM Trans. Database Syst.*, 6(4):602–625, 1981.

[2] D. Draper, A. Y. Halevy, and D. S. Weld. The nimble integration system. In *Proc. of SIGMOD*, 2001.

[3] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, Athens, Greece, 1997.

[4] D. Maluf, D. Bell, and N. Ashish. Len middleware. 2005.

[5] D. Maluf and P. Tran. Netmark: A schema-less extension for relational databases for managing semi-structured data dynamically. 2003.

[6] A. Rosenthal, L. Seligman, and S. Renner. From semantic integration to semantics management: Case studies and a way. *ACM SIGMOD Record*, December 2004.

[7] L. Seligman, A. Rosenthal, and J. Caverlee. Data service agreements: Toward a data supply chain. In *Proceedings of the Information Integration on the Web workshop at the Very Large Database Conference, Toronto*, 2004.

[8] M. Stonebraker. Too much middleware. *ACM SIGMOD Record*, 31:97–106, 2002.