Introducing Enterprise Integration Modeling

A Key Component to Integration Fitness

Contivo, Inc.
640A Clyde Court
Mountain View, CA 94043
United States of America
(650) 426-4000
http://www.contivo.com

EIM_whitepaper.doc
Publication Date: August 27, 2001

Table of Contents

1. Executive Summary

Integration is now a competitive necessity for enterprises. Whether spanning across internal applications or connecting with critical supply-chain partners, integration project architects face the problem of getting business data in disparate systems to communicate seamlessly together.

The solution today is to manually match and code the way each interface should talk to its corresponding interface in another system. But this approach has encountered two enormous problems: with each additional partner or application to integrate, the number of links to create increases exponentially. Furthermore, the moment one interface or standard changes, the original code linking them must be rewritten. This constant "reinvention of the wheel" is unscalable, dragging out deployment times and simultaneously incurring enormous and recurring labor costs.

Enterprise Integration Modeling (EIM) represents a core component to integration architecture. Rather than approach each new partner or application as a discrete integration project with its own set of data mappings, an enterprise can leverage the power of the computer for data modeling. This has significant implications on the behavior of the system:

- Repetitive, labor-intensive mapping tasks and code generation shift from manual efforts to computers
- Changes flow from a single source throughout the enterprise without disrupting individual systems
- Additional applications or trading partners trigger incremental, rather than Herculean, efforts
- Knowledge previously locked within individual personnel, departments, and consultants can be extracted and reused

2. Getting Integration off the Couch

Integration is like fitness—by all expert accounts, long-term health and survivability rely on achieving fitness. But actually getting there isn't all that easy. Not only do the paths to fitness vary from person to person, given individual limitations and preferences, but implementing the plan often involves far more effort than previously imagined. Getting up at 05:00 every morning for a three-mile run sounds a lot easier than actually doing it.

Likewise, every company's integration needs are different. While the high-level business goal of integration is to make and/or save money by sharing **information**—on customers, products, assets, employees, and suppliers—the existing applications, new incoming systems, and integration philosophies can differ considerably from company to company. However, most integration projects do share a common problem: the actual execution of the integration plan rarely proceeds as anticipated, leading to long delays, inflated expenditures, and short tempers.

Simon Yates of Forrester attributes this difficulty to three causes: 1) most companies have legacy systems that were never designed to communicate with other systems; 2) even packaged applications, such as SAP, have been customized. As such, one-size-fits-all-SAP-customers solutions do not work; and 3) standards, such as the various flavors of XML, have yet to gain wide acceptance.[1]

Furthermore, the way information technology (IT) systems have been developed over time increases the difficulty of functional integration. John Bermudez of AMR Research writes, "Frequently, the [enterprise application integration strategy] approach has been to create integration code on demand, leaving IT with a mass of classic spaghetti code that is extremely expensive to maintain."[2]

Even though enterprises may have adopted a uniform architecture for integration middleware, the code required to implement the process of transforming data still suffers from the same "spaghetti code" syndrome. As the enterprise grows, individual silos in different departments continue to develop their own characteristics, the knowledge of which is captured only in the minds of the individuals who developed it. Extracting this information and configuring each silo for sharing while maintaining the functionality of the system is much like tying shoelaces while running a six-minute mile.

As a result, integration remains an extremely resource-intensive process—which means that it comprises large swaths of IT budgets. Purchasing the integration software is a relatively small part of the total integration budget. The bulk of spending goes towards consulting services—the legions of people who plan, program, and assemble integration systems. According to a Forrester survey, 64 percent of companies spend between 60 and 80 percent of their integration budget on consulting services alone.[3] Most of these consulting services go towards programming the infrastructure that will connect and carry data throughout the enterprise.

That integration is so tough is hardly a surprise. What, then, is causing such persistent integration headaches? The answer is rooted in The Exponential Curve.

The Old Lady Who Lived in a Shoe . . .

The sheer number of systems to integrate—from the home-grown order-entry system to the occasional supply-chain contact—is a key obstacle to achieving integration fitness. Yankee Group analyst Jon Derome writes, "In other words, it is not simply about integrating applications; it has rather become an issue of integrating the extended enterprise and in some cases large value-chain communities. This approach is a definitive change from the previous era of integration products, which were written to allow customers to link any two applications together."[4]

Not only will enterprises continue to deal with the addition of tactically-driven, single-purpose systems, but also with external partners, each of whom may require a different depth of integration. As such, an integration architect can clearly see the looming problem: if all these systems must be connected, the addition of even one single system will result in exponentially more connections to make.

This "exponential curve problem" is dawning on the enterprise software market. Amy Hedrick of AMR Research describes the problem:

> "Here's an example of how the math might go:
>
> - Two applications, one connection. Hey, maintaining one point-to-point solution is easy.
> - Three applications, three connections, That's okay.
> - 5 applications, 10 connections. Two new applications spawned seven new connections. Are the connections coded in the same language? How well are they documented so that we can change them later? Do we need to add programming resources? Can we still do this?
>
> Alas, these connections are seldom static, for as standards processes, applications, trading partners and programmers change so too must the connections. IT managers who expect the connections problem to settle down and/or go away are simply not being realistic."[5]

The following two diagrams further illustrate.

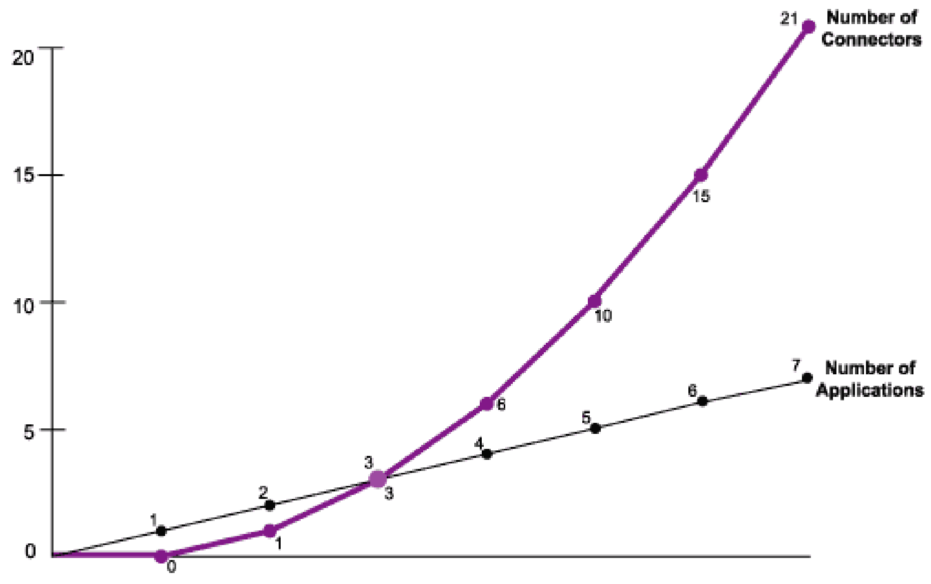
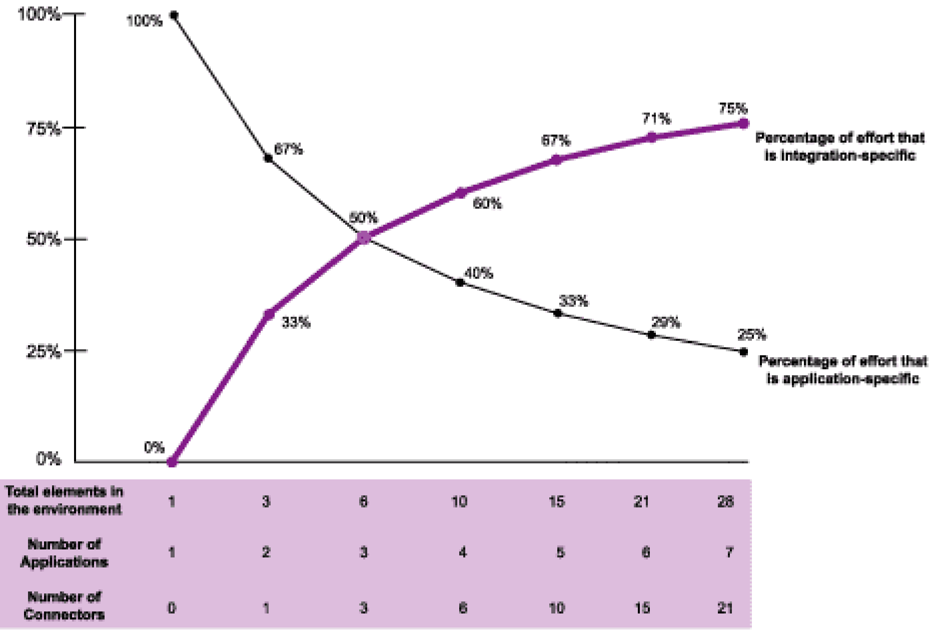


Figure 1a: Each new application increases the integration burden
Source: AMR Research, 2001



Total elements in the environment	1	3	6	10	15	21	28
Number of Applications	1	2	3	4	5	6	7
Number of Connectors	0	1	3	6	10	15	21

Figure 1b: The integration learning curve is steep but tends to level out with experience
Source: AMR Research, 2001

The lack of reusability contributes to the steepness of the exponential curve. A META Group report notes, "A major drawback with implementations of RosettaNet Partner Interface Processes (PIPs—and indeed with IEI [inter-enterprise integration] solutions in general) is that process integration needs to be implemented repeatedly for each new relationship."[6] Code is written specifically to link two interfaces with little regard for how it can be leveraged in subsequent, similar situations. Moreover, programmers may have their own ways of writing code, making it difficult for another programmer to alter or repurpose it.

Second, the exponential curve problem worsens when factoring in maintenance. For example, when a field name changes, all existing maps referencing that field must be manually adjusted, if indeed the code can be understood by someone other than the original author. Or on an even broader scale, an upgrade of a standard, for example from xCBL 2.0 to xCBL 3.0, represents significant rework of existing maps and potentially crippling downtime.

3. XML: A Standard by Any Other Name

It was difficult for the industry to see the exponential curve two years ago. As with any shift in IT philosophy, the theory was initially sound and beautifully simple: adopt a message-based integration architecture, install packaged applications, and somehow, all will be seamless. But packaged applications, such as SAP, rarely avoided customization, since the needs of a financial services firm, for instance, differs greatly from an oil and gas firm. This created the need to extract the information into some sort of common format.

This "common format" arose in the form of eXtensible Markup Language (XML), made viable by the adoption of the Internet as an enterprise-level transport layer. With its ability to separate data from the processes that surround it—in other words, make data meaningful within a certain context—XML represented an incredibly flexible way to share information across any system. As a result, XML has taken the integration industry by storm with its promise of lower-cost implementation, adaptability, and simplicity. Widely touted as a replacement for Electronic Data Interchange (EDI), XML promised to revolutionize enterprise-level e-commerce.

However, the same characteristics that broaden XML's appeal have contributed to a proliferation of non-interchangeable XML dialects, built for specific verticals or particular trading communities. John Edwards writes in *Line56*, "Whereas a handful of EDI iterations exists, there are at least 500 XML specifications, ranging from Trading Partner Agreement Markup Language (tpaML), for the exchange of business contracts, to XForms, which governs how Web designers create forms that ask for your name, credit card number, clothing measurements, and other personal information."[7] As a result, trading partners within the same verticals may select different versions of XML, essentially throwing the integration problem back to square one.

4. Case in Point: Data Transformation

As long as integration remained manageable by *ad hoc* solutions, the exponential curve didn't emerge as a big problem. But with integration's increasingly strategic role in carrying out business imperatives, and the growing scope attached to the definition of integration, the importance of a **scalable** integration architecture intensifies.

Take data transformation, which is the process of converting the data within one document type (e.g. a purchase order in EDI) to another data format (e.g. ebXML). Traditionally, data transformation has been viewed as a line item task on a system integrator's statement of work. Because of the variability intrinsic in disparate systems, the idea of a **software solution** to data transformation never materialized.

An unglamorous task even in the best of times, direct programming costs associated with data transformation can constitute 20 to 30 percent of the direct implementation cost[8], not including maintenance and upgrades. Add to that a dependence on human-intensive analysis, and the cost can easily reach 50 to 70 percent of the direct implementation cost.

First, business analysts document the differences in meaning between, say, a PIP3A4 purchase order field, "ShipToAddress" and a legacy system "ShipAdd1." This requires significant business knowledge of what the fields actually mean[i]. Business analysts enter these relationships into a spreadsheet, such as Excel, then send the requirements to the computer programmers. Next, programmers then manually write the code to describe how these relationships should function in a run-time environment. This heavy dependence on individual knowledge (in the case of the business analyst) and custom code (in the case of the programmer) makes any sort of reusable solution an unlikely proposition.

Enterprise Integration Modeling

In response, Contivo has introduced a new approach for integration design-time called Enterprise Integration Modeling (EIM). EIM addresses the exponential curve problem through the use of modeling. It forms the foundation of a scalable data transformation and management solution, drastically reduces implementation times, and extracts and reuses knowledge currently confined in individual heads and systems.

A model is like a picture in that it represents something that can be expressed in many different ways. For example, a picture of a dog can elicit the word *perro* in Mexico, *hund* in Germany, and 狗 in China. Despite these different expressions, the meaning of the picture remains the same and can be reused to elicit a variety of words in different languages. Likewise, once a model of a particular SAP IDoc has been created, it can be deployed over and over again—as either a source or target interface—with minimal incremental effort.

EIM leads to the creation of an enterprise vocabulary, much like a thesaurus, which contains the definitions of significant data elements created during modeling. When used to define integration mappings, the enterprise vocabulary eliminates the combinational expansion in the number of maps necessary to integrate multiple applications. This leads to several changes in the behavior of the integration ecosystem.

Applying Computer Power

Data transformation is highly redundant. Regardless of how familiar a programmer is with the "Invoice Number" field in an SAP IDoc, he will have to re-write code depending on to which data type that Invoice Number is mapping, and the rules by which that particular field is transformed. Up to

[i] For example, "credit" and "debit" can mean the same thing, depending on whether they appear on the Assets or Liabilities side of a balance sheet.

this point, mechanizing this process has been difficult because most enterprises have highly varying internal systems and constantly-changing external integration requirements: in other words, few situations could be reapplied others.

With EIM, the situations themselves need not be replicated. In a sense, EIM makes components of the parts that constitute documents, so that creating a map involves simply reassembling and reusing the components. When combined with products that automatically generate code based on this reassembly, the error rate decreases significantly and scalability increases.

Single Point of Control

The expense of an integration projects continues well after the initial software purchase and implementation. Maintenance has proven to be of ongoing concern, since it directly impacts the integration environment's total cost of ownership.

Existing methods of dealing with standards changes or new software are increasingly cumbersome, as the pace of change accelerates. For example, a change in a field name, its length, or the rule associated with its transformation results in a host of reprogramming often disproportionately large compared to its initial cost. Likewise, when an XML "flavor" is updated, those changes must somehow percolate through the entire system.

Using a modeling approach, changes are made within the enterprise vocabulary, which then flow through to the affected components. Using products such as Contivo's, code is automatically generated, eliminating the need for another round of hired programming help.

Linear Systems, Linear Effort

Automatic telephone switches eventually replaced live telephone operators precisely because of the exponential curve problem: if humans had to physically connect every telephone call made, the majority of the population would be sitting at switchboards[ii]. Similarly, EIM guards against the eventuality in which the need to match and create data links would consume the majority of integration efforts.

The benefits of EIM run deeper than merely creating a common enterprise vocabulary. When the complexity of the data increases—for example, an Address field whose particular meaning depends on qualifiers—EIM proves even more valuable. To take the address example: in EDI, the field "Address" can have several qualifiers, including "Address>ST" for "Ship to" and "Address>BT" for "Bill to." When manually mapping, a rule providing this additional meaning must be written each time the field is used. Solutions

[ii] *Useless Trivia No. 47:* The last manual telephone switch in America was retired in 1983 in Bryant Pond, Maine.

incorporating EIM can model the qualifier itself, saving considerable time not only with regards to matching individual fields, but also in the generation of rules.

Extracting and Encapsulating Knowledge

Much energy is spent on discovery and distributing knowledge about documents. EIM helps to extract and encapsulate this knowledge for reuse and collaboration.

For instance, an EDI programmer with expertise in the ANSI X12 standard may need to connect a purchase order between her company and a partner that uses RosettaNet XML. In ANSI X12, a purchase order is known simply as an "850"—and it has a standard structures, set of contents, and method of communicating with other EDI systems. Even if this programmer logged onto the RosettaNet website, she may not realize that the equivalent of an ANSI 850 is a RosettaNet PIP3A4. Further, she may not know how to correlate X12's concise header level information to the PIP3A4's verbose content. If such information was stored in the model, deployment accelerates and domain expertise is preserved, regardless of the individual programmer.

5. The Benefits of Integration Fitness

What does EIM mean for "integration fitness"? It means first overcoming the "build only when needed" IT mentality and applying a conscious design process to integration. It means an integration architecture flexible enough to meet changing business needs while detailed enough to allow the development of tightly integrated business processes. It means that the effort to add the 101st partner or application is no more difficult than the first.

An initial investment in designing and modeling an integration architecture generates immediate and enduring benefits.

Benefits to the Business

Faster deployments. A primary business benefit of EIM is the ability to integrate applications, customers and trading partners more quickly and reliably. Establishing a shared view of data is a significant portion of the work necessary to roll out a new enterprise application or partnership. If that data is already modeled, then the incremental effort becomes simply mapping to the model, rather than creating a new set of maps from scratch. In a B2B scenario, faster deployments enable marketplaces to achieve liquidity faster and overcomes supplier reluctance to join public and private exchanges. In an EAI situation, faster deployment enables accelerated achievement of business interoperability and streamlined costs.

Better decision-making. Improved data quality improves the results of data-driven business decisions, both internally and externally. A "one view of data" approach offers true real-time information to make meaningful internal decisions. At the same time, the selective knowledge of external partner data, such as inventory levels, optimizes trading relationships and leverages the principles of strategic sourcing.

Better project planning. The root cause of many cost overruns is imprecise planning. Integration architectures who use a modeled approach can predict project timeframes more reliably and anticipate costs more accurately.

Benefits to IT

Adaptability. EIM enables a gradual evolution of enterprise applications from their current state to future states and beyond. With its ability to reuse code, effort, and knowledge, a modeled approach combats the exponential curve and maintains pace with the constant evolution of standards.

Reduced time and costs. Time and cost savings in an EIM approach accrue from three main areas: 1) fewer number of links to make; 2) reuse of integration code and an accompanying reduction of integration errors; and 3) less time and cost to document information requirements for subsequent efforts. These translate into sooner and higher return on investments (ROIs) for integration projects.

Better inter-team coordination. An EIM approach is inherently collaborative. Different teams, whether within or outside a company, can easily see and affect changes to the model. This reduces communications requirements, better coordinates project scheduling, and transforms individual knowledge into team knowledge.

6. Summary

The race for competitive advantage is increasingly stressing the importance of integration as the bridge between IT and business requirements. Yet, integration remains devilishly complicated, and approaches to integration remain fairly diverse. As Gartner notes, "Vendors do not share a common vision of what should be included in an integration middleware project set."[9]

Enterprise Integration Modeling describes an approach that relies on modeling to stimulate reuse and flexibility. When applied to data transformation, or any other endeavor that involves the transfer of structure and meaning between two different systems, EIM proves to be a powerful source of accelerated ROIs, leveraged effort, and scalable maintenance.

Sources

[1] Simon Yates, "Demystifying B2B Integration." *Forrester Report.* September 2000, pgs. 8-9.
[2] John Bermudez and Bob Parker, "Enterprise Commerce Management: Getting Started." *AMR Research Executive View.* July 2001.
[3] Simon Yates, ibid.
[4] Jon Derome, "Application Integration in the Brave New Internet Era." *The Yankee Group Report.* January 2001, pg. 9.
[5] Amy Hedrick, "The Good, the Bad, and the Ugly of Integration Implemetnation." *AMR Research Repor ton Enabling Technologies.* April 2001. p. 8.
[6] META Group News Analysis, "CompTIA Study Challenges RosettaNet Hype." 20 July 2001.
[7] John Edwards, "I'm Not Dead Yet." *Line56*, May 2001. p. 55.
[8] AMR Research response to a research inquiry. June 22, 2001.
[9] Roy Schulte, Ross Altman, "Application Integration: Success Amid Turmoil." *Gartner Group Article Top View*, August 8, 2001.