Real Time Enterprises

A Continuous Migration Approach

Vinod Khosla, Murugan Pal

March 2002

Real Time Enterprises – A Continuous Migration Approach

Vinod Khosla, Murugan Pal

Current market trends, global competition, and technological innovations are driving enterprises to adopt the practices of Real Time Enterprises. **Real Time Enterprises are organizations that enable automation of processes spanning different systems, media, and enterprise boundaries. Real Time Enterprises provide real time information to employees, customers, suppliers, and partners and implement processes to ensure that all information is current and consistent across all systems, minimizing batch and manual processes related to information. To achieve this, systems for a Real Time Enterprise must be "adaptable" to change and accept "change as the process".**

Any business process within the enterprise, including relevant processes in use by its trading partners (the extended enterprise), must be instantaneously reflected in all enterprise systems. In other words, all INFORMATION is "real time" within a "real time enterprise". All manual or batch processes related to information in an enterprise are inefficiencies in the delivery of products and services – unless the manual and batch mode processes (as in process industries) are required as part of the business nature. For example,

- In a Real Time Enterprise all the systems everywhere could recognize the new product entered in a catalog system so that billing and customer service can be done right from the moment that product becomes available.
- A wireless carrier could activate a wireless phone as soon as the credit card payment is processed with out any time loss or manual intervention.
- A credit card company could improve customer loyalty by automating the dispute notifications starting with the customer, and extending to the credit card company, the merchant's bank, and all the way back to the merchant.
- Last year, Lexmark had $1 million worth of nonconforming material returned to one of its plants in a single lot. Investigation revealed that providing engineers with adequate information online and in real time[3] could avoid this situation and future inefficiencies.
- Citibank, to avoid heavy call volume in Poland around paydays, introduced cellular text messaging (SMS) to inform all customers of any changes in their bank balances instantaneously on their cell phone.

Today's business practices and models demand an operational environment acting as a virtual enterprise, with insight into the status of customers, partners, and suppliers on a real time basis while lowering SG&A costs. Cisco and Dell, with better service and higher revenue per employee than their direct competitors, are great examples of leading technology adopters who have leveraged some of these capabilities. At the same time, companies like Lexmark[3] and Cutler-Hammer[7] have realized similar benefits through automation. Still, automation of an end-to-end value chain has not been widely adopted or fully achieved. Even though technologically this has been possible for some time, only now has it become realistic with the advent of Internet-driven standardization. This has led to orders-of-magnitude cost reductions plus the elimination of debate regarding the technical infrastructure to be used. With the advent of Internet technologies such as HTTP, HTML, standardization initiatives around XML, Web services, UDDI, and SOAP, it is now possible. Lexmark leveraged Internet and thin client technology to enable their engineers to monitor production processes at their suppliers in real-time, from anywhere in the world and put defective product on hold at the source. Dylan Tweney in his column states, "Web services will enable companies to link up their enterprise systems with the production processes, bringing executives ever closer to the ideal of 'real-time enterprise computing,' and in turn will make companies better able to respond rapidly to changing market conditions"[7]. We agree.

Cisco's ambition to close their books on a daily basis is the litmus test for a Real Time Enterprise. Consider the benefits of having all information current in all systems such that books can be nominally closed within hours of the close of a quarter (or day!). Cisco does that today, and after adjustments including managerial and auditor input can announce financial results within three days of the end of the quarter. Think of the cost savings in finance alone! Cisco Systems' much-vaunted electronic order-entry system has decreased the rate of errors for the company from 20% to 0.2%[5]. If a majority of the orders come in untouched by humans, think of the sales force efficiency and yield improvements. If most employee information (such as vacation days and 401K's) is "self service" on the intranet, think of the savings in HR. If order status, product configuration, and "available to promise" dates are self-service for customers on the Web, think of the improvements in customer service that are possible, while reducing costs. These improvements are not just about a Website but a structural change to Web enabled IT. The benefit of "self service" is enormous as data will be cleaner when the owner enters it and the process will be efficient because it is outsourced to the end customers themselves. This is how the Internet is being used for information transport rather than a browsing medium. It represents a change in the way finance, operations, HR, logistics, and the whole corporation works. According to Roland Berger's Geissbauer, manufacturers already using the Internet see annual cost savings of 6 percent across the value chain. From procurement to Web-based supply chain management and after-sales service, it may be possible to cut costs by as much as 8 percent to 10 percent[3]. We think the savings are potentially larger. On the other hand, Cisco failed to automate its supply chain deep enough into it's partners, resulting in hundred's of millions of excess inventory beyond what a normal demand forecasting error would have caused in a full real time, full visibility environment.

The reality is that the vast promise of IT, by and large, has been a mirage for most corporations. But does this have to be true? Does the promise only work for some organizations? Are missed opportunities for productivity gains just that, or can they be realized? Are budget overruns, process delays, and "additional costs" to recover the investments already made an unavoidable fact of life? We have gone from MIS departments with large in-house software development efforts, to inexpensive desktop enablement, to large packaged software applications, to productive application development tools, client server applications, portable environments like JAVA, and system integration tools. We have gone through IT consultants, outsourcing vendors, ASPs, the Big 5 and their system integration expertise, but the problems of IT remain largely the same. Even more tantalizing are the stories of "benefits" of good IT strategies. Cisco has substantially higher revenues per employee than its direct competitors, Nortel and Lucent. The cost of doing business is lower at Cisco, and their responsiveness and customer service levels are higher. Geissbauer stresses that many of the top challenges for manufacturers relate to competitive pressure and manufacturers need to respond faster to customers in order to achieve their top-line sales goals. Dell can offer "mass customization" and still maintain much higher inventory turns than its competitors, generating greater profitability in its PC business. Wal-Mart and Amazon have used IT technology as strategic weapons to increase their competitiveness. FedEx could not economically provide the level of customer service that it does without IT. The cost to FedEx of a "package pick up call" or a "where is my package" inquiry have declined substantially (greater than 10X) because of the use of appropriate technology. We guesstimate that each 1% (of sales) of increased IT spending or spending redirected from rigid and outmoded forms of systems integration in a corporation should reduce SG&A spending by 1.5% to 2% (of sales) beyond the improvements in IT productivity. It is important to note that the bigger role of IT is not managing IT functions and expenses, but rather to manage expenses and service levels for the "rest of the corporation". IT can be a strategic weapon and help reduce SG&A costs relative to competitors while improving customer experience. The rate of savings depends on industry sector; those sectors with high SG&A can realize the most significant benefits. This applies especially to business processes, collaboration environments, and personalized applications where the bulk of enterprise activity can be automated. It is clear that every business process, every manual or

batch process, and every human touch point that is properly automated and eliminated will result in an economic saving as well as an improvement in quality by reducing the risk of human error and improving the availability of information.

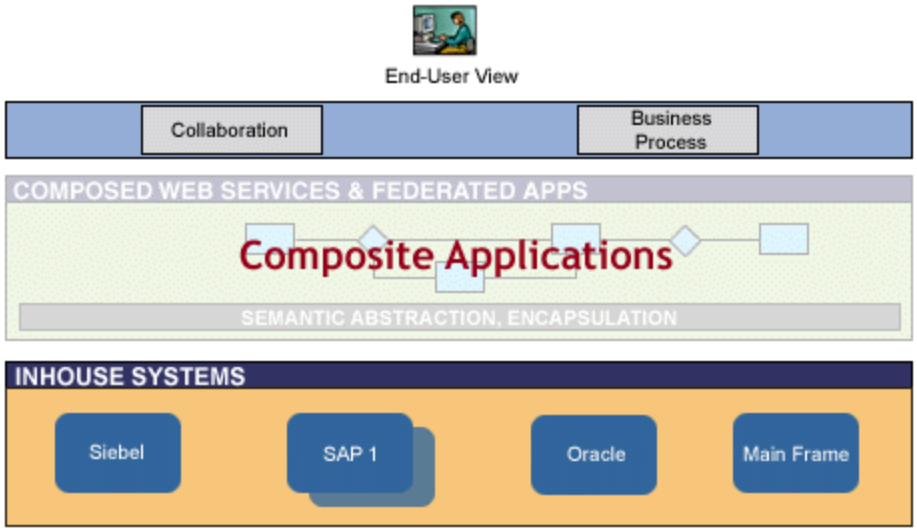
Any astute CEO or CFO will choose to automate and eliminate provided that the risks can be managed and results can be demonstrated in small projects with 90-180 day implementation and payback cycles. IT can become a competitive weapon, with the CIO becoming as critical in reducing costs or improving "product" as the VP R&D, in defining products as the VP of Marketing, in improving customer service as the VP of Customer Support, or in improving operations and reducing operations and administrative costs as the VP of Operation or the CFO. In summary, the CIO becomes a strategic leader.

The goal of this paper is to define the "ground rules" of an IT transformation from the stark reality of legacy applications forward to the promise of the future. In fact, the goal **should not be a radical transformation but rather continuous migration of systems, transforming the organization into an adaptive enterprise**. It is easy to define the ideal "new" world, particularly with the promise of Web services, but harder to achieve a practical reorientation towards such a goal. No magic bullets or ideal solutions exist. However, biases in certain directions and small probing steps do help. Differences in approach can result in a significant impact over a three to five year time frame. Ground rules and technology choices can make environments more flexible, adaptable and pliable over time. It is this paper's goal to highlight these, as well as the "current best targets" towards an ideal environment. Keep in mind we are dealing with first generation attempts at this new model and will run into many "gottchas", issues and practical problems. But iteration towards the above goal is necessary because it is unlikely that we will ever have an instantaneous or ideal tool for the transition. What we will need to do is dedicate an increasing part of our maintenance budget, the largest component of most IT budgets, to approaches that will enable the legacy environment to move towards the new vision.

The inflexible structure of conventional systems has long been the subject of loud complaints by top management. Today's customer expectations, evolving business models and technology trends demand the **need for adaptability.** It is important to accept **CHANGE AS A PROCESS**, rather than as an EVENT. IT spending must be evaluated against the total expenditures of an organization, and the potential savings from the effective usage of IT. These evolving business needs demand:

- "Inter-enterprise integration" to shift from Enterprise Resource Planning (ERP) to Inter-enterprise Resource Planning (IRP) and to resolve issues arising from the rigidity and cost of linking systems.
- "Intra-business functional integration" to unify business process beyond what packaged applications capture.
- "Self sufficiency" from a systems perspective, allowing adaptive, low cost iterations, customizations and change isolations, rather than a requirement to get it right the first time.
- Understanding that optimization for adaptability is more important than optimizing for cost, performance or features.

The ability to encapsulate existing systems, automating them as business processes, and letting users collaborate via appropriate interfaces are the keys to Real Time Enterprises as illustrated.



The value of real time enterprises is in capturing the greatest value obtainable from the systems people have created so far, and operating with the same data set that previously existed.

There are a number of philosophical approaches we will discuss later. System integration has been a problem. Customization has made systems static and unchangeable. We will recommend **federation,** not integration of applications, **configuration,** not customization, and a bias towards a more dynamic architecture. The big advantage of Web services is that it is inherently open, perhaps even **"micro-open and multi-vendor".** Web services are inherently multi-vendor, but preserving this may take some conscious decisions. Security, authorization, and entitlement are major issues and hence the very basics of a Web services vision have to include a comprehensive **entitlement system** in multiple granularities.

Goals

Today's enterprise IT problems are tied to the "islands of information" caused by many legacy architectures distributed across geographies, business units and M&A subsidiaries. The technology evolution has forced many enterprises to buy new software and hardware resources. This has resulted in "best in class", sometimes "most in class", and many times "Try, Buy, Throw" environments. The challenge is to leverage existing operational systems, evaluate "most in class" systems, and reuse the most meaningful. Many times the real problems are legacy processes rather than legacy systems. In such cases, integration is a wrong approach to achieve the necessary plasticity and adaptability. As per Gartner, 70% of all infrastructure efforts fail or substantially miss their objectives. Large integration projects like Boeing's I-Man portal fail to achieve the desired results 60 percent of the time, according to Giga Information Group analyst Julie Giera[6]. She says companies put too much faith in technology's ability to cut costs and fail to adapt old processes to make use of the new technology. There are many reasons that such projects fail.

Our recommendation is to achieve change in small steps. Projects should last from 90 to 180 days, whenever possible. The key to a successful migration is identifying internal champions who realize the benefits of Real Time Enterprises and are willing to evangelize the required efforts. The other dimension of this strategy is to offer end-user configurable processes so that **"iteration" becomes a specification methodology**: getting things approximately right and iterated by the end-user.

Optimization of systems can be achieved in four dimensions: flexibility, cost, performance, and reliability. Our recommendation is to prioritize flexibility ahead of cost, reliability and performance. The evolution from MIS built applications to packaged applications to today's need for more adaptability is a tough challenge. The right answer is found neither in in-house applications nor standardized third party packaged applications; it is somewhere in the middle. A little bit of both allows business to achieve customization via configuration. These are the "**composite applications**" using processes (pieces) from different packaged applications to assemble (configure) a business process to match how things are really done in the enterprise. But more than any other single factor, **flexibility and adaptability** are the *most* important selection criteria for new technology given the rate of change in both technology and enterprise requirements. Unfortunately, this tradeoff is seldom made, with features, performance, and price often winning out.

Automation of processes configurable to end user needs is the key (where the end user is defined interchangeably as a human being or a machine capable of understanding specific semantics). This migration can be a continuum and need not be a one-step change. Evolving architecture changes are not only demanded by technology and/or business changes, but also by the rate of adoption. The processes themselves can be assembled from sequence of other processes or can span into a supplier organization using different technology stacks. All these processes operate in a world of common data, semantics, protocols, and translators that we call an "**information base"** for a Real Time Enterprise. Jeff Jensen[6] of Boeing identifies pulling data together into a single repository as key for their I-MAN project's success. This single repository must be represented as a "virtual" repository — an abstraction for all coexisting data sources based on a flexible schema. Current data repositories, such as relational databases, do not support such a model. We address some requirements under the section "Information Base" (iBase), but as a first step, documents are extracted from the company's various legacy systems and converted to XML as queries are invoked. Boeing plans eventually to move data currently residing in its multiple systems into this repository for easier administration[6].

The challenge is to retrieve data from multiple business units, sort and analyze them. The Web services vision has much of what is needed and we recommend it. It builds on existing Web technologies and accommodates legacy systems to a reasonable degree through a process of **node enablement** we will define later. It works very well for green-field implementations, but is also workable for real legacy environments. The goal for every enterprise betting on this vision should be to create an **"application assembly environment"** where end-users can create **"composite applications"** or **"composite Web services"** suitable to their personal or corporate work style. This environment operates within the constraints of corporate business processes, often coded into the end user programming environments by business analysts (often as rules or objects, not as computer programs) and to limited extent by programmers to create components and services in languages like Java and C#. Think of an Excel or Visio as the front end for programming Web services to suit the business process modeling needs. Most non-technical end-users can "program" their application into an Excel spreadsheet within the constraints of "macros" that might be defined by sophisticated end-users or business analysts. This creates a **"mass customization environment for business processes, workflow and collaboration"** with sufficient support tools for administration, personalization, versioning, upgrading, and more — allowing end-users to keep their own store of templates and a template exchange to make them productive. The recommended changes should articulate benefits both at macro and micro levels (i.e. each individual project must justify that their value adds individually and as an encapsulated service within a wider scope). This could be a big challenge given the scale of many corporations but planning this migration is *a priori*. For example, changes in an "Order Status" process may benefit all users within a group (micro level), but an organization also can be impacted at a macro level by offering the process as an automated Web service incorporating entitlement and personalization to key customers. Although this migration can be painful in the short term, savings in maintenance costs and the ability to

enable end-users to serve themselves will be a huge benefit in a long run. The lack of an ideal environment or too many immediate requirements should not be an excuse to keep doing yesterday's thing. Every organization, to the maximum extent possible, should encapsulate the old world through a process of "**enabling legacy nodes**". These nodes would work in a Web services world, making iterations in functionality, the maintenance spend, the Web services, Java/C#, UDDI, SOAP, XML, XML and schema world of Internet technologies. Many companies who gained their knowledge from Y2K exercises and large-scale system conversions can leverage that experience to abstract legacy functionality as encapsulated modules and expose those modules as Web services — making the migration process easier.

This automation requires all the "point" applications — CRM, ERP, supply chain systems — to provide information and interfaces to business processes that have touch points in other systems or by humans within the extended enterprise. EAI is a first generation solution catering to data synchronization needs, leveraging or extending into the Web services environment by node enablement of much of the legacy world. Many older world tools, often developed for Y2K transformations, will be useful to this node enablement process, as will more modern tools like portals and "Web service publishing tools". Tools like Citrix can encapsulate many legacy environments from the PC world with Web front ends of packaged client server applications. Node enablement is only the start of the solution. Replacing the many interfaces to older layers underneath the Web services is the key for migration. Imagine a company with 10 billing systems, each dealing with a different customer segment or region, but each offering basically the same services. These systems will have many thousands of interfaces that can be presented as Web services. All of the **business services** offered by those systems can be migrated (possibly as common abstractions) as services that route appropriately to the underlying systems, with every new system talking to the Web service layer. This first step makes it possible to start building new, workflow-driven business processes without reference to systems, but referring to services. Over time, the cost/benefit balance will gradually tilt towards forced migration of all older interfaces. The resulting enormous benefit is the ability to restructure, rationalize, and consolidate those 10 billing systems as appropriate. When this type of migration (occurring over several years) happens in every area, *then* you are able to completely realize the even bigger benefits of integrating new businesses and outsourced services (such as billing, order management, and data management). The goal is composite applications that leverage legacy applications to model the business processes, collaboration and workflow needs of the enterprise, done in an incremental way.

The grand vision of federated Web services will happen in stages. Security and other practical issues (like billing, SLA's, etc) will constrain most Web services to work initially within the firewall as applications inside the corporation are updated to work together. The level of granularity will test our wits and increase only slowly. Semantic and ontology differences will limit the ideal world of possibilities into a world of very useful but far from ideal possibilities. Think of Web services as the next leap from the Web request architecture (typical HTTP, CGI, and Application Server requests) to a **Web services request architecture.** This goal is as achievable as the transformation from client/server to Web request architectures in the late 90's.

Web Request	Web Service
• Ad-hoc • Stateless • Simple and light-weight to define semantics • No Encapsulation for object transport • Undefined behavior resulting from Exceptions	• Pre-defined, well negotiated, deterministic and reliable • Support for stateful transactions • Semantics can be defined and embedded • Encapsulation supported for XML Schema objects • Infrastructure available to support exception handling behaviors

Example: Cutler-Hammer[7]

With more than 61,000 orders processed electronically last year, Cutler-Hammer realized their design-to-delivery vision by using Bid Manager on complex assembled products manufactured across 26 satellite plants across the U.S. and Mexico. Bid Manager handles small but significant details. For example, it can automatically compose labels that specify capabilities, such as speed and power, of each motor and its components. Then it can direct the nameplate engraver to print the label. In the past, a technician would type the nameplate information, increasing the likelihood of error and slowing the process. Cutler-Hammer's customers, field reps, and distributors are able to electronically design and place 95% of their orders remotely, bypassing plant engineers. One such customer, Grove Madsen Industries, a supplier of equipment to Las Vegas casinos, feeds their design directly to the Cutler-Hammer assembly line 2,050 miles away.

Over time as this subset of corporate processes migrates or infiltrates into the whole corporation and forces migration of legacy systems, the fusion of new technologies and legacy systems with rigid processes may pose a paradigm mismatch, thus complicating the migration process. In such instances, we recommend encapsulating those areas and abstracting them via well-defined interfaces and behaviors.

Why Now

Many will ask why an enterprise should transform into a Real Time Enterprise now, and how IT can play a role in this process. Changing business models, evolving market, competitive pressure, and cost benefits of leveraging existing systems drive the need for this transformation. The balance between Return On Investment (ROI) and Risk Of Not Investing (RONI) is the key factor in answering the "why now" question.

Return On Investment (ROI)	Risk Of Not Investing (RONI)
Infrastructure renovation in an evolutionary way	Legacy systems not matching current dynamics as in Boeing example[6]
Cost savings + real time adaptability	Expenses, both cost and competitive losses
Improved responsiveness, customer loyalty	Lack of real time insight, process overheads
Leveraging technological improvements	Dwelling on past accomplishments

Five years ago, if someone quoted a scenario such as our Cutler-Hammer example, it would have raised many eyebrows regarding infrastructure for communication, consensus on semantics, and agreements on exception handling, encapsulation and implementation mechanisms. Today, one can model a solution using HTTP (S), SOAP, XML, Rosettanet, XMLschema, Web services, and language-independent abstracted interfaces for this seemingly intractable problem.

Some Guidelines

There are four golden rules and corollary implications vital to the IT transformation process:

- Plan on being wrong.
- Adopt thy partners.
- Design thy architecture.
- Implement thy solutions in steps.

Our proposal is based on our experience with the IT industry and technology evolution. Interestingly, similar recommendations and guidelines can be found from *The Forrester Report's* "Start Using Web services Now"[13] and "The Web services Payoff"[14], and the HBR article "Your Next IT Strategy"[12].

Plan on being wrong

As previously discussed, it is important to identify the IT customers, their expectations, behaviors, usage trends and future needs. Cutler-Hammer enlisted not only software writers, but also experts at the plants, sales engineers, and many others to compile the requirements. It is also important to consider various types of customers such as:

- Interacting customers via browsers, bandwidth constrained devices, and voice access systems.
- Machine-based connectivity via XML2XML, EDI, Partner Interface Processes (PIPs), and other client environments.

Beyond the obvious attention to users (customers) is the far more important and often ignored fact that most requirements cannot be fully specified prior to actual use. Often 30-50% of the cost of a new implementation goes into specification of screens in a hypothetical environment. It is much more reasonable and feasible in the Web services world to only attempt an approximate solution to the business process. Through actual use, the end user can modify the "application Web service", personalize it, and compose it with informal practices and processes. These modifications can be based on the idiosyncrasies of each user or practice group and subject to the continually evolving constraints placed on the user by the business decisions (through a business analyst). This process leads directly to the need for a mass customization language for business processes utilizing the Web services environment. Enabling real use of IT because of increased relevance to each user may be the single biggest benefit of the Web services architecture. It is important to understand the variations and evolutions (e.g. multiple and customized versions with different granularity) that the customer community will demand. For example, Applied Materials (AMAT) could have a generic selling process that has to be customized specific to Intel and Motorola's needs and derived from the generic process. Change and adoption are *more* critical in dynamic environments than getting today's environment exactly right. For example, end user modifications/programming at the level of Excel programming is very desirable in a system where end-users fine-tune the environment to fit their needs.

Ground Rules:

- Assemble a virtual team of people (not exceeding 3 or 4) to identify and compile the requirements. The team should represent a cross section of the customer community along with developer and business analysts, and have a senior techno-functional person as the moderator, as well as clearly defined deliverables, and specific timelines. Qwest Communications gets these teams together for intense 3-4 day "sessions" before a 90-day implementation cycle commences.
- Understand existing current solutions, their gaps, and what end-users actually need. For example, a trading broker using Excel would not want to use a browser interface even though it gives real time capability. The right solution is to provide an Excel interface with real time capability.
- Implement a prototype (with stubbed interfaces) that can be quickly customized for end user needs to understand real caveats of the proposed functionality.

Adopt thy partners

The Real Time Enterprise must span the physical boundaries of an enterprise and should extend as a virtual enterprise through the entire end-to-end value chain. For example, Colgate-Palmolive trimmed its inventory 13% and saved $150 million by attaching its order planning systems to thousands of Wal-Mart and Kmart[9] cash registers. This extension includes the collaborative partners, suppliers, sub contractors and vendors (all referred as "partners" hereafter). Each one of these partners may vary in size and have their own infrastructure for implementing their IT systems. Certain partners may exceed the expected enterprise guidelines and some vendors may not have any infrastructure at all. Your enterprise and your IT organization should adopt all these partners and adapt infrastructure support to their level. This eases the paradigm mismatch in information flow across partnering organizations and helps transform your enterprise into a Real Time Enterprise. The goal is to create a partner eco-system around your organization — helping them help you become successful. A successful eco-system allows the partner set to be easily and rapidly changed based on evolving business needs and level of sophistication. Partner collaboration is another important aspect from design to implementation, as well as during execution. The usage model, user management, rich security model, and transparent seamless processes across enterprise boundaries are key factors of a collaborative application environment. Web services are a great way to implement partner collaboration systems: they abstract the interfaces across partner systems, do not depend on the implementation language or object types, and provide loosely coupled connectivity (unlike RMI, CORBA or RPCs).

Ground Rules:

- Identify short-term projects; with incremental value added and well defined loosely coupled interfaces (initial iteration).
- Identify candidate value chain companies (customer, tier 1 supplier, sub vendor, contractor), going 3 or 4 levels deep "into your close partners[13]", as suggested by Frank Gillett.
- Pick the right type of projects and partners for your domain needs[15].
- Define semantic interfaces, exceptions, and data types for information flow across the chain (Rosettanet PIPs); provide pre-implemented plugins that can run within a standard runtime (servlets or Web server plugins), if necessary.
- Agree on periodic iteration updates and be willing to spare your own resources to help implement partner solutions.
- Include partners when talking to solution providers and create usage scenarios and proof-of-concept environments based on real partner needs.

Design thy architecture

Once you identify your customers' needs and connectivity requirements to your partners, the right architecture must be designed. Your architecture's mindset, belief, and vision should cater to the short and long term goals of your enterprise. The architecture should be based on:

- Open standards and multiple vendors.
- Flexible, loosely decoupled interfaces considering rapid changes and evolutions of underlying systems.
- A production system that can manage, monitor, and support versioning of the Web services life cycle.
- Ability to leverage existing systems and reduce cost.
- Flexibility as more critical than optimization for cost, performance or features.

The designed architecture should not tie you into any proprietary environment and should accommodate all software lifecycle phases without the overhead. The right architecture will be modular and abstracted, with simple and consistent interfaces. Good architects are the scarcest resource and using existing talent is the most common and most significant mistake made in IT. To quote *The Forrester Report*[1], "Projects succeed or fail because of software architecture". Adaptability and flexibility is often determined by architectural choices. Allowing for a complex set of diverse suppliers (possibly two in each critical category) and forcing these disparate choices in the current implementation may be the only way to accommodate the unforeseen needs of the future.

The traditional architectural approach integrates runtime environments for data synchronization and information flow, without consideration for how user management, globalization, and personalization (key elements of Internet solutions) can be supported across various integrated point solutions. For example, consider a business process that touches a CRM system built as client-server system, a supply chain (SCM) system built as Web based solution, and a sales commission system built in mainframe using COBOL. Each system has its own user management system: CRM using Windows NT authentication; SCM system using an LDAP solution; and the mainframe using case-insensitive passwords. One challenge is to map the user password information from the Web solution to a different backend. Another challenge is language translation (globalization) for various level solutions since the SCM system may support Unicode, the CRM system may support Windows globalization, while the mainframe may not support any globalization.

The goal here is to design a

- Reasonably flexible system that can adapt to future needs — i.e. it can embed any type of future application environment.
- Cohesive framework — i.e. provide common methodologies (open standards based) for business object modeling, process definitions, and data source mappings.
- Utility-based service architecture for authentication and authorization (**entitlement**), globalization, personalization, monitoring, caching, presentation, business rule definition, and execution that can coexist with existing legacy systems in an evolutionary manner.
- Comprehensive framework to support various types of users (Intranet, Extranet and Internet) based on their entitlement within different administration models (self managed or delegated administration).

The design must account for existing systems; for example, globalization utility service must not only be based on Unicode, but should map underlying systems such as Oracle, SAP, and other functional components seamlessly in multiple levels (data, business object, API and business process levels). It should address globalization in multiple layers (data schemas, formats and styles like currency, menu systems, messaging components, presentation templates, and locale based rules). Every one of these subsystem layers must be locale aware, automatically detected during runtime based on a user's preferred locale, and pre-generated and cached where and when possible to improve performance. All these locale specific components must be abstracted and encapsulated in the development framework so that translations are performed in a scoped manner and narrowed to resource files, directories, and definitions.

The concept of configuration vs. customization is a key to end user enablement. Many solutions claim to support configuration but in reality do customization. Configuration is, by definition, something you have changed to adapt to your business needs and is guaranteed to upgrade without any modifications. Configuration can happen in many levels.

- Building Blocks
    - Providing basic building blocks (e.g. email notification, approval routing, file attachments etc.) and let the user assemble a business process.
- Development Configurations
    - Re-implementing the functional logic for the same interface, more towards the end of customization, with upgrades guaranteed.
- Deployment Configurations
    - Declaratively rerouting functional or control flow or changing the presentation components.
- Runtime Configurations
    - Rule Based, depending on different user roles and authorization levels.
    - Data Driven, depending on data loaded or derived based on specific calculations.

The architecture must separate programmer level customization from business analyst level configuration. While there are some performance impacts, the configurations at the development and deployment levels can be converted to a byte code level (Java classes) and cached after the first invocation. Only the runtime level configurations are left to the interpretation mode.

Ground Rules:

- Be open to customer needs, to technology infrastructure changes over time, and be flexible on reiterating the design, since architecture is the key to adaptability.
- Design a cohesive reference architecture that can adapt to future needs or can be extended. The architecture should be modular — "what you need is what you use", portable, and standards-based as much as possible.
- Clearly demarcate between the roles of programmers and business analysts. Stress the importance between configuration and customization.
- Define a common object model and data-modeling environment that enables a flexible schema (refer to iBase section in building blocks) to extract and operate on data from various systems underneath (ERP, SCM, Mainframe, client-server systems).
- Define a sand box of data types, exceptions, process interfaces, application integration methodologies, and development models (covering a wide spectrum) across the enterprise.
- Decide on make vs. build and focus on your core competence. If you can find a product catering to your needs, even with reduced functionality, you should adopt it since it can take care of future maintenance and support costs.

Implement thy solutions in steps

The solutions and implementation roadmaps should be phased in over 90 to 120 day increments, whenever possible. This is possible more often than is generally assumed. The development of LINUX is proof positive of the feasibility of this approach. The solutions must be scoped in such a way that the incremental implementations over a period of time result in the ultimate solution, but every phase should result in meaningful and useful addition of functionality. Every deliverable becomes trackable, modifiable, and implementable. Avoid the large multi-year system integration projects whenever possible, even when you are told incremental implementation cannot be done. The promise of Web services guarantees migration in small and evolving steps. One can abstract the interfaces and the implementation

can grow in modules. The assembly or orchestration of these modules, resulting in a more meaningful business process, can take place in the future. For example, a traditional Order Management ERP system can expose Order Status (O/S) as a Web service today. It then can link the O/S Web service with the Order Entry (O/E) Web service that will be implemented at a later point of time as an automated Order Management Web service. An implementation beyond twelve months is likely to be wrong by the time you get there, as the business requirements will have. When lifecycles for applications are short spending 12 months in the evaluation phase while a reasonably usable solution could have been built in 3 months is imprudent. As a minimum, the shorter process will clarify system requirements and caveats, which in itself is a success.

Ground Rules:

- Decide on enterprise-wide adoption time lines and plan on coexistence in diversified system implementations.
- Identify and scope candidate application areas for individual vendor exercises. The life expectancy for these applications should not exceed 9 to 12 months.
- Have vendors provide a migration strategy describing how they would migrate to other vendor's solutions or to the entire enterprise.

Pick the right vendors and a phased implementation approach, migrate other existing solutions to the right approach, and use the new approach as a nucleus to extend the base functionality.

Approach to Transformation:

The transformation to enable Real Time Enterprises becomes smooth when the following areas are approached properly.

- People.
- Architecture.
- Meta-Architecture - Architecture to connect architectures.
- Node Enablement & Legacy Encapsulation.
- Shifting the maintenance environment to the new vision.
- Composite Apps and application assembly environment.
- Building Blocks.

People

People are very instrumental in the success of a project. A strategically diverse team ranging from field service individuals that interface with customers, to line of business managers, key architects, and candidate programmers should be assembled. Often IT organizations are more conscientious when choosing the technologists than when involving the broader constituent base. This increases the likelihood of mismatched expectations. The gene pool for this team is very important. The constituents should be open and aware of evolving technologies and standards, and be willing to take a shepherd approach rather than a sergeant approach in testing new solutions. Qwest forces three-day intensive interactive, offsite sessions before the start of a project where all the business users, vendors, and IT participate and iterate the needs, statement, build prototypes, and otherwise hone in on expectations. Every IT organization should change the mindset from "Consultants" to "Resultants" — i.e. the implementers (especially the system integrators or vendors) must be paid after the results are

delivered, not for their consulting hours. The notion of "fixed scope of work" rather than "solutions" creates disincentives for the right, iterative approach, increases the upfront cost of specification needlessly, and sets up a conflict between the vendors need for billable hours and the clients need for iteration at low cost.

Architecture

- *Federation NOT Integration*
  Focus the application assembly on how each application can be federated rather than integrated. The difference between federation and integration is in the meaningful sequencing of processes loosely coupled with best possible efficiency to implement the business process, while preserving ADAPTABILITY. This is different from traditional integration of connecting point-to-point solutions for synchronization, thus resulting in high-level abstraction and coarse-grained process automation. A simple analogy is how an automobile differential gear engages the wheels with the motor. The differential gear drives the axle with the best possible efficiency, accommodating the speed difference between wheels, and is adaptable to varying conditions by coupling "loosely". Some efficiency is lost but a large degree to flexibility is gained. Web services hold a similar promise.
- *Configuration NOT Customization*
  The assembly of these coarse-grained processes must enable the business analysts and business users to be self-sufficient, serving their own needs rather than depending on an IT person or a Systems Integrator to the maximum extent possible. Loose coupling allows Web services to cater to the need for configurability.
- *Reliable and Self Recovering*
  The implemented systems should be capable of handling failures on their own and of failing gracefully. Low consequence failure mechanisms (e.g. backup or skeletal operation) are more important than "high availability" in many situations. For example, if two different processes are assembled and if the down stream process fails, the former process should be capable of recovering from all known and unknown exceptions (generic handler), as defined by the behaviors established by the business user rather than the programmer.
- *Evolutionary NOT Revolutionary*
  The choice of architecture should allow modular components that can be implemented in an evolutionary way. Having a creative way of assembling existing applications in a meaningful manner is more desirable than simply having one more new way of developing applications. The architecture choice should enable various degrees of solutions and bridge all potential point systems rather than sticking to certain religious choices. The Asera[18] system, based on meta-architecture approach, and the general process of node enablement, discussed later, are early attempts at this. Layers of abstraction as an architectural feature are key to preserving evolutionary capability for systems.

Meta-Architecture - Architecture to Connect Architectures

Automation of processes across multiple systems and applications requires a flexible multi-layered architecture. This meta-architecture is a simple layer rather than dedicated platform. This layer will provide large-scale reusability rather than fine grained component model architecture. In addition, this layer will provide semantic abstractions and well defined and agreed interfaces to bridge into legacy systems. This layer will also automatically reduce instances of redundant systems within an enterprise. No single vendor can meet all needs (be best of breed) across varying application areas over many years and product versions. Plan on a

multi-vendor architecture and avoid spending 3x-5x the cost of the system with system integrators. The nature, orientation and characteristics of Web services rightfully fit this model. Web services and SOAP protocol inherently support loosely coupled interfaces independent of language implementation and transport protocols. XML's interchangeability and SOAP's capability to abstract HTTP or asynchronous queues enables building a seamless value chain spanning inter, extra, and intra enterprise systems. Web services may not be the only way to solve these problems, but it is the one that naturally fits the model and is available now. We paid special attention to this set of needs when founding Asera. Asera builds solutions by first inserting this "abstraction layer" and tailoring solutions as "composite applications". This approach enables the world of new business process orientation by leveraging both the processes and information embedded in existing IT infrastructure. In addition, Asera provides common services around these legacy and composite applications, such as globalization, entitlement, and personalization, among others.

Node Enablement

The software choices you make should provide systems (node) and process enablement at the right levels of granularity, synchronization, and behavior. Corporations should strive to transfer as much as possible of the maintenance spend on legacy applications and environments into new paradigms as they evolve, instead of doing the patches and enhancements in the old world. If nodes are encapsulated, abstracted, and thus enabled to participate in the new world, maintenance can be gradually shifted to the new paradigm - currently, the Web services paradigm. The purpose of this enablement is to improve self-productivity and to re-leverage existing systems and processes. The process should include identifying the areas to automate, partners to include, information islands to connect, and end-users needs. The new "process enablement" should result in morphing linear and sequential processes into collaborative processes accommodating today's business needs. Each legacy application should be published as a Web service that can be used in a "composite business process". This way, users do not have to traverse 50 screens across 5 applications to do order entry; workflow and collaboration can happen easily on these legacy processes and information, independent of departmental and corporate boundaries and restricted only by the entitlement system.

The level of granularity depends both on what needs to be reused and what the end user community can handle. For example, if a mainframe data object is extracted as an XML object and merely presented as an HTML table, it may not make sense for the end-users since they may not understand how to interact with the table. At the same time, if that particular data object needs to be fed into another process, it cannot be fed as a COBOL object in that runtime environment. The granularity can be in multiple levels, purely in the data attribute level (data types), in the business object level (XML format), in the business process level (APIs or process invocation entry points), or in user interface levels (screen emulation). There are no right or wrong levels. It depends on the level of functionality that needs to be leveraged, time and budget available for integration, life expectancy of the underlying system, and the new integration layer.

Reuse of legacy systems depends on what has been already implemented, and what needs to be leveraged. For example, there could be a generic pricing rule with manual overriding capability implemented as part of the ERP implementation. In order to adapt and leverage, all "generic selling" processes may call into an existing pricing rule within ERP. But, when it comes to a privately negotiated exclusive relationship, you may want to add a business rule in a different layer that can "simulate" the manual overriding capability within the underlying system. This enables the existing process to adapt to specific needs by adding additional business rules in a different layer. The existing process is encapsulated with well-defined interfaces and behaviors.

An open approach should be adopted for legacy enablement. For example, if the user community is comfortable with existing user interfaces (mainframe screens or client UI), the goal should be to reduce the operating costs rather than migrating the users to a new interface. A suggested migration path is to have the screen interface exposed in a portal environment (a Citrix enabled emulation) with other portal objects available in the same Web page. Using the same screen increases user comfort and acceptance while reducing client operating and license costs. In addition, the user community will be able to adapt their learning pace to the capabilities of other portal objects and the power of Internet. This will facilitate the adoption of new technologies over time and ease migration of legacy user interfaces to either thin client interfaces or "native" client interfaces. We refer to "native" client interfaces as productivity application suites such as Microsoft Office that communicate to the backend through the Internet.

Application Assembly Environment and Composite Applications

The design time environment for process assembly (i.e. to build end user centric business processes assembled from pre-defined components or sub processes) is the key to end user enablement and leveraging of existing IT investments into higher ROI's. This approach meets the custom needs of the enterprise while minimizing custom software code. The resulting business processes can be called "Composite Applications" or "Composite Web Services". The design time environment should provide an intuitive interface (e.g. Visio) for the business analyst using services provided by the IT programmer. This environment should include graphical modeling of the business process-flow (e.g. a drag and drop interface) and tools for process assembly. A robust design environment should also include emulation tools that validate assembled process flows and translators that generate common repository formats suitable for the runtime environment (i.e. workflow languages like BPML, WSFL or XLANG, though transparent to the business analyst). Design time environments that generate common repository formats do no limit future design changes in runtime platforms. The design time environment should support multiple versioning (configurations specific to custom needs derived from basic versions) and support role-based entitlement capabilities for automatically picking the correct version during runtime. The key challenge here is to identify the common repository format for specifying user interfaces common to HTML, WML, XML2XML, XML2EDI that can also accommodate sequencing of screens and declarative interfaces to define UI elements for the design time environment to generate. There is not one standard that can support all these requirements. This provides an opportunity for picking a key vendor and actively participating in the definition of standards. Solving this piece of the puzzle is a critical step toward enabling real time capabilities through the composition of underlying "application Web services". The utility of current packaged applications will be increased by enabling participation in the composite application that services a corporation's business process, as practiced by each group.

Shifting of maintenance investments to the new visions

Identifying and projecting the life expectancy of applications and systems is very important for the real time enablement of an enterprise. For example, a mainframe system requiring end-users to process many of the logistics prior to data entry loses an opportunity for providing real time capabilities. Instead, a plan for migrating functionality to the new system with a retirement timeline should be established. An alternative to investing in the maintenance of old systems is the building of a parallel new system providing the same capabilities. Y2K exercises provide good case studies for this approach. Providing "native" client interfaces to Internet-enabled back ends will be the next wave of paradigm shift within enterprises. Think of having Excel, Word, and Outlook replacing browsers while providing the advantages of Internet without the thin client element. Case studies from MSDN provide good examples for using office documents[10].

The majority of IT expenses in most corporations are on "maintenance spends" on old IT systems. If some of these expenses are used to encapsulate old IT systems and enable Web services that can participate in "composite applications", then there will not be a need to rewrite or modify these applications and the business process/logic they incorporate. A "continuous migration" of legacy applications, "maintenance spend", and "extensions" to legacy will evolve in the Web services world resulting in maintenance money going to the "right" side of the legacy/new equation. This is achieved through node enablement. The common argument that there is no choice in adopting the new because most resources are spent on extending and maintaining legacy systems is replaced by the possibility of enabling participation in new processes and automation by extending the legacy systems through a series of small steps.

Building Blocks

The building blocks for the transformation can be categorized into five categories:

- Node Enablement.
- Information Base.
- Applications Enablement.
- Process Enablement.
- User Enablement.

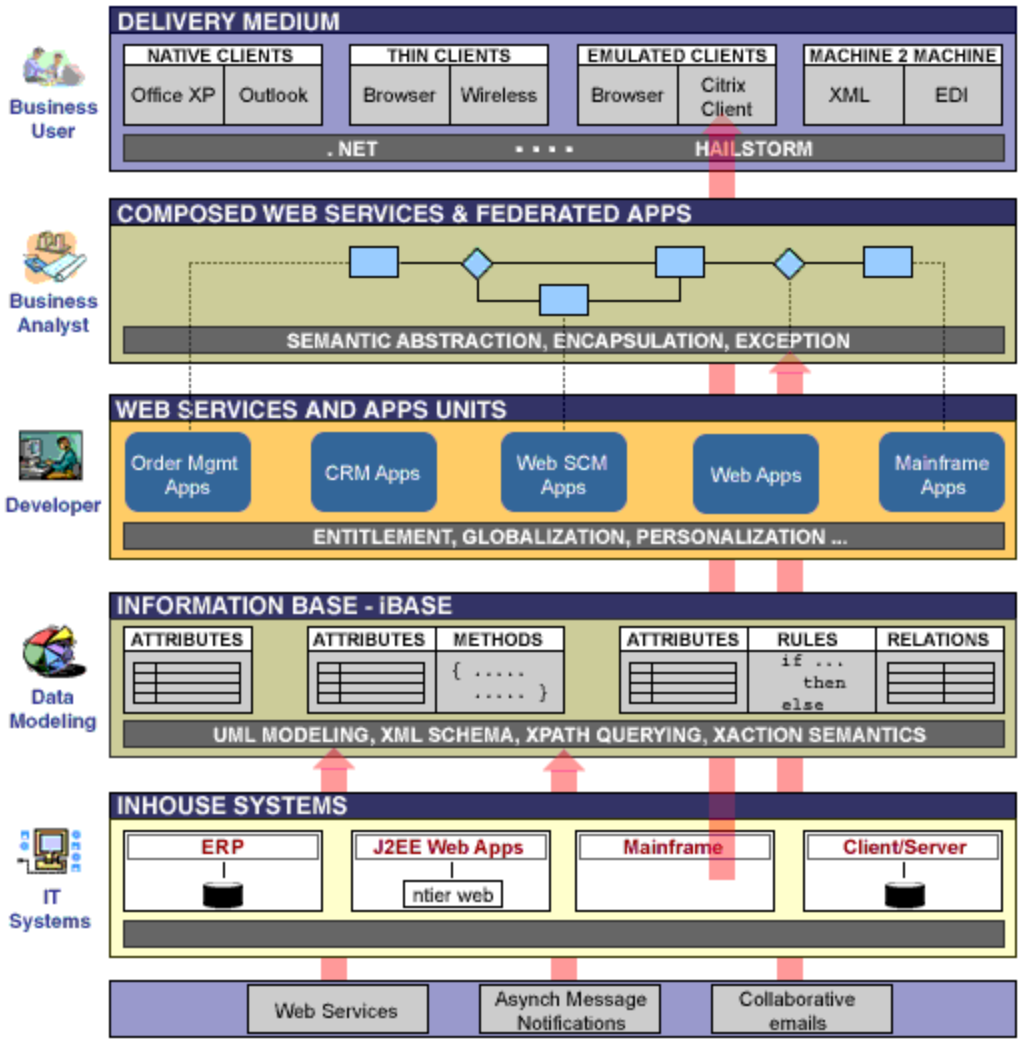
These building block layers in fact resemble the evolution of IT systems over the past 3 decades as illustrated in *The Forrester Report*[13].

Node Enablement

Node enablement is the process of re-using existing systems and applications without any modifications or additional expenses. The stress should be on encapsulation and legacy reuse. Key characteristics of this process should be predicting the obsolescence of the application over time, choosing the right granularity for encapsulation, reuse vs. restructure vs. rewrite of the process logic and representation, and in defining the invocation interfaces and semantics for invocation. Technologies exist today to provide legacy node enablement, as do software vendors who specialize in this area.

As explained in the previous sections, the key decisions to node enablement are:

- Cost justification – evaluate costs by hardware, software licensing, new extensions, performance impact and training for new interfaces.
- Life expectancy - project the life expectancy or obsoleteness of the underlying system over a period of time before investing in node enablement.
- Reusability factor for legacy systems – based on the current business model compared to the original time of legacy system implementation.
- Level of desired integration - depends on the kind of interaction needed for other systems relying on the legacy node. This can range from data attributes, business objects, business processes and UI levels.
- User community readiness – to accept new systems and interfaces. Continuous migration with a phased deployment approach will be helpful.



Information Base - iBase

Information Base (iBase), also known as XML database[2], is the key differentiator for the proposed model. iBase provides an universal schema for all types of data objects and models in a loosely coupled way. iBase introduces the notion of "**Beyond Databases to Data Sources**". As explained by Scott Dietzen[2] (CTO of BEA), much Web data is not in a database and service-based architectures have to hide the data model. iBase, in our opinion, is not a radical concept. A long standing requirement is that each enterprise must choose an approach to implement and accommodate changes in object/repository specifications, business partner interfaces, and communication modes for achieving federation across partners and competition-turned-partner or vice versa. iBase will be based on XMLschema for object definitions, meta-data structures, meta-schema, meta caching, and some extensions for simple rule and relationship definitions. iBase will provide a language independent environment to define and implement methods associated with object definitions in an encapsulated way. iBase will support encapsulated object models comprehensive enough to support data objects from various sources including RDBMS, ODBMS, flat files, Web services and sites, main frames, EDI, or any data source. The need for basic object level services such as synchronization, translations, messaging, security, and entitlement and the importance of metadata will drive the migration from database to information base for corporations.

Applications Enablement

The goal of this block is to bring packaged applications into the Web services/composite applications world. Web-enabled client-server applications and new Web solutions have matured, making this a reasonable building block for leverage by IT environments. The only abstraction that is essential for transformation is to package the building blocks as "apps units", or "**application Web services**" that have well defined encapsulation and delegation for entitlement, globalization, and personalization for use in "composite business processes". The higher-level building blocks can delegate requirements with respect to collaboration, intra-application state management, entitlement, and globalization to this layer. For example, a portal page with Siebel's CRM object should be able to delegate the user locale and process information to the underlying Siebel application rather than translating the rendered contents in the portal layer. Here the portal component should assume the role of aggregator rather than translator. Companies like Asera, BEA, and Bowstreet, in combination with application server and EAI vendors, can be used to implement this block. Again, the granularity of integration or enablement depends on what level of integration is needed and what resource and time commitments are available. The application enablement can be achieved in various levels ranging from the data level, object level, API level, business process level, Web services level, and presentation level. The level of granularity to use will be a critical decision in this enablement. At the same time basic services like entitlement, globalization and personalization may not be available as the levels move from data to presentation level.

A typical process for application enablement includes:

- Deciding on the functionality to be exposed.
- Modeling right business objects in the common object format (ex. XMLSchema).
- Mapping business objects to data source formats (ex. Trading object to Auction System's native format).
- Writing connectors and adaptors to the underlying system.
- Mapping delegation interfaces to the underlying system with respect to application services like globalization, entitlement, and user management. If you want to provide missing services, you may have to write in the new layer.
- Processing business objects by routing them to a process-enabled workflow, then through delivery medium.

Process Enablement

Process enablement, also known as "Federated Applications Environment", allows business analysts to assemble process sequences for "their end user" needs using intuitive design-time environments like Visio or Excel as well as custom business rules that can be expressed declaratively as in Excel macros. The process sequences should support standard branching rules, parallel execution capability, event monitoring, and trigger facilities interfaced with collaborative emails. This is a lightweight layer built on a standard application server's runtime platform or other application runtime platforms to automate and manage processes in a coarse grained environment. This block will provide collaboration, inter-application state management, semantics encapsulation, abstraction and exception handling for assembled business processes. In addition, it will support fine grain version control in order for the end-users to maintain their own versions of customized business processes. This block transparently and appropriately delegates error handling, business intelligence tracking, access control, monitoring and reliability to all the underlying blocks.

Process enablement is the key paradigm shift from traditional environments and evolving requirements. Internet-enabling 24x7 assistance to customers, vendors and suppliers, and automation of end-to-end value chains require end-users to configure their own services to meet their needs. Self-productivity and configuring/assembling business processes from a number of possible permutations are very important. A declarative-based design tool, which can support multiple versions, different granularity levels, entitlement-based customized versions, a publishable repository for the enterprise, and Extranet partner to leverage are just a few characteristics of this block. Semantics for encapsulation of "application Web services", exception handling, and interface abstractions are essential requirements. For example, let us assume Applied Materials (AMAT) is defining a basic spare parts selling process. Any small customer would go through this generic business process. An AMAT rep working on the Intel account may want to derive this generic process and customize to a new version specific for Intel or for Motorola. The system should support different custom needs for each of these AMAT customers, but at the same time be confined to sand box definitions of exception handling and interface signatures, while providing mutual exclusivity between these two processes (Intel process hidden from Motorola customers). In addition, the discount calculations for each one of these customers may be different and hence the need for the system to support declarative business rules. These simple declarative rules (sometimes complex) have to be associated and managed with either generic workflows or with each customized version of the business process flow.

There are evolving tools and standards available today to implement this block. It is not a matured area, but it is an opportunity for your enterprise to adopt one of these early vendors to influence them and the standards to cater your needs, making it a win-win situation.

User Enablement

This is the top-level building block that provides user interfaces to the end-users and personalization capabilities with respect to content rendering. This block should provide interfaces via native clients (Office, Outlook) for power users, thin clients supporting multiple devices, emulated clients for legacy users, and faceless clients with machine-to-machine automated connectivity. Portals, Rosettanet PIPs, and UDDI interfaces are good examples of faceless clients.

"Mass Customization" is a process that enables end-users to **"configure"** their processes and to tailor their **"custom"** needs into their **"native"** format or environments. Although this may appear to be paradoxical, the following example provides clarification:

Today's energy trading brokers retrieve their base data from mainframe computers via ftp, copy it to their disks, go to their desktops, use Excel to process the cost model, derive recommendations, execute them, and repost the data back into mainframes via ftp. The need for these business users to leverage their **"native"** Excel tool (power environment) and run **"custom"** macros developed individually to process or dissect their data to achieve a **"configurable"** solution is a good example of **Mass Customization**. Obviously, there exists a need to connect the front-end power user environment (Excel) to the back end systems (mainframes). We propose this as the goal for the user enablement block.

The user enablement block should provide a rich set of design-time tools, a version control environment, a template store, an exchange environment, and an emulation environment for legacy systems and interfaces to "native", thin, wireless, and machine-to-machine interfacing clients.

Building blocks in future

How will these building blocks evolve or adapt to the needs of IT in the future? IT could move in the direction of "bandwidth aware" smart applications, "live document" embedded applications, and "scenario optimized" automated applications. The user enablement block addresses the first two types of applications, while usage tracking, combined with a rule-based personalization within the process enablement block, addresses the last one. Even though all usage types cannot be anticipated, it is interesting to have these building block layers evaluated for growth and change. In his report[13], Frank Gillett indicates the growth stages of Web services will follow those of the Web itself.

Vendor Landscape

Open Standards, Emerging Technologies, and ISV Offerings are three key dimensions that influence the migration towards Real Time Enterprises. Several vendors who enable implementation of the building block layers are listed below. Even though this is not a comprehensive or an exhaustive list, it is a good selection of vendor candidates. Detailed vendor studies can be obtained from Gartner and Forrester materials (References 10, 13, 14, 16, 17). The characteristics of a comprehensive e-business architecture are described by Daryl Plummer in the "Technology Drivers for E-Business and Global Computing"[17]. Daryl's commentary on "4 Web services platform"[16] categorizes Web services platforms into 4 different types: to produce, consume, manage and provision Web services.

Node Enablement

- System Vendors – include IBM, Microsoft, Sun, Oracle and BEA. They provide execution environments where the basic threads of execution are managed and monitored.
- Legacy Access Vendors – include SEEC, Microfocus, Sapiens, Most Software, Netron, and Jacada. They provide access to legacy systems and expose them as Web services.

iBase

- Tools and Modeling Vendors – include Metamatrix, Enosys, Excelon, Right Order, Rational Rose, and Together Soft. They provide XML repository, caching, object modeling, and code generation tools.

Applications Enablement

- Enterprise Application Vendors – include SAP, Siebel, i2, and Matrix One. They provide packaged applications for enterprises, including ERP, CRM, SCM and PLM solutions.
- EAI Vendors – include Webmethods, Tibco, Vitria, and See Beyond. They provide integration capabilities such as messaging, queuing, transformations, collaboration, etc.
- Core Service Vendors – include Netegrity, Entrust, Oblix, ATG, Epicentric, Plumtree, Savvion, Versata, Core Change, and Mobile Q. They provide essential services such as security, directory services, portal, personalization, business process management, wireless device access, etc.

Process Enablement

- Web services Platform Vendors – include Altoweb, Avinon, Bowstreet, Cape Clear, Grand Central, Infravio, and Iona. They provide a production platform for Web services.

Given all these components, there is a need for a meta-architecture (architecture connecting architecture) to house different systems and to federate them, which we will call the **Generic Purpose Production Platform**. This platform provides basic services including globalization, personalization, workflow orchestration, monitoring, caching, and business object management. These services must be available as standard services and the platform must offer more ways to interface (beyond Web services). Microsoft has attempted to provide these services via .NET, while IBM and BEA propose their own equivalent stack on J2EE. A preferred approach is to use a multi-vendor approach like Asera's generic purpose production platform, based on a referenced meta-architecture and partnered with some of the above listed vendors, enabling customers to develop, deploy and manage composite business process.

Challenges

The following are some challenges you could face during the transformation.

- How do you find people with the right design experience?
- How do you define correct semantics, interface contracts, abstraction and encapsulation?
- How do you make systems fail gracefully when spanning different systems?
- How do you model human intervention as automated processes during node enablement?
- How do you incubate your partners or suppliers while not exposing the competitive edge to their other customers?
- How do you stay ahead of the leading edge on technologies but not get caught in the hype?
- How do you define Quality of Service and Delivery to guarantee deterministic behaviors in a loosely connected world?

It is important to reduce the risk of obsolescence by making change easy enough so that you are not constrained to using only larger, entrenched vendors. This almost always will also coincide with lower total cost.

Conclusion

As we understand the importance of Real Time Enterprises and how IT strategies can enable transformation into Real Time Enterprises, it is important to consider the following:

- Select your platform, development environment, standards strategy, and vendors/partners based on your architecture.
- Small implementation steps result in incremental efforts.
- Insure your chosen eco-system does not lock or tie you into a proprietary environment.

Intentionally, this document did not cover issues with regard to

- Cost of ownership.
- Environment management.
- Details of workflow and collaboration.

Our focus is on technology and architecture for continuous migration of software infrastructure.

Our recommendation is to adopt this paper as a guideline for modeling your architecture and migration path based on your specific needs. It is important to identify your organization's

steering committee, which will agree upon plans to implement the strategy. We defined the "ground rules" for an IT transformation (to facilitate continuous migration) and an approach to implement them. We suggest adapting to biases in certain directions and taking small probing steps to help migration. Again, we are dealing with first generation attempts at this new model. There will be challenges during implementation, but the reward is worth the effort to migrate into this exciting path. These guidelines will help you automate your business processes in a collaborative environment and provide you an economic saving as well as an improvement in quality. The savings in SG&A spending, quality improvements, and time spent serving your customer needs are the crux of the benefits you will realize from automating IT and enabling collaborative business processes. IT migration strategies should include ease of change management and evolution as a managed process. Many times, cutting edge technologies come out of startups, while IT shops too readily adopt large software vendors.
Before one can impact the cost structure of an enterprise outside of IT, namely the SG&A costs, IT must be built in a responsive and evolvable way. Many of the promises of IT will become deliverable, and IT will become a strategic weapon for the corporation. Even ambitious goals that everyone can appreciate, like doubling the revenue per employee for a corporation, will become feasible.

Reference:

1. Laura Koetzle, "Putting J2EE to work," *The Forrester Report*, July 2001.
2. "An interview with Scott Dietzen," *Infoworld*, Nov. 7, 2001. http://www.infoworld.com/articles/hn/xml/01/11/06/011106hndietzen.xml.
3. "Factory floors go online." http://www.internetweek.com/ebizapps01/ebiz031201.htm.
4. "The E-Factory Catches On." http://www.business2.com/articles/mag/0,1640,14874,FF.html.
5. "Where the Web is really revolutionizing business," http://www.businessweek.com/magazine/content/01_35/b3746669.htm.
6. "Boeing links apps via XML." http://www.internetweek.com/enterprise/enterprise111201.htm.
7. "What's going on down at the plant." http://www.business2.com/articles/web/0,1653,34859|2,FF.html.
8. "Know your IT, Know your customer." http://www.businessweek.com/bwdaily/dnflash/oct2001/nf20011023_3424.htm.
9. "Era of efficiency." http://www.businessweek.com/magazine/content/01_25/b3737701.htm
10. "XML Web Service-Enabled Office Documents." http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexcl2k2/html/odc_xlB2B.asp?frame=true.
11. Dale Vecchio, "Legacy Software: Junkyard Wars for Web Services," Gartner Symposium, IT Expo 2001.
12. John Hagel III and John Seely Brown, "Your Next IT Strategy," *Harvard Business Review*, October 2001.
13. Frank E. Gillett, "Start Using Web services Now," *The Forrester Report*, December 2001.
14. Simon Yates, "The Web Services Payoff," *The Forrester Report*, December 2001.
15. Simon Yates, "Which Industries Will Adopt Web Services," *The Forrester Report*, December 28, 2001.
16. Daryl Plummer, "4 Web services platforms," *Gartner Research Note* COM-15-1387, January 2002.
17. Daryl Plummer, "Technology Drivers for E-Business and Global Computing," Gartner Symposium, IT Expo 2001.
18. www.asera.com

Appendix A – Requirements for Node Enablement

Node enablement is the process of re-using existing systems and applications without any modifications and additional expenses. For example, a Citrix client can enable end-users to use all mainframe applications without any new client licenses or new hardware. Similarly, a browser-enabled environment can reduce UI training for an existing client-server application. The node enablement can happen in the application as a whole (as in the Citrix case) or in the object/data, logic/process, and/or interface levels. The enablement process can be planned over a period of time, moving from application level or UI level to the object level (the deepest). Key characteristics of this process should be predicting the obsoleteness of the application over time, choosing the right granularity, reuse vs. restructure vs. rewrite of the process logic and representation, and defining the invocation interfaces and semantics. There are software vendors who specialize in this area and appropriate technologies exist today to provide legacy node enablement.

As explained in the previous sections, deciding on the

- Cost justification,
- Life expectancy,
- Reusability factor for legacy systems,
- Level of desired integration, and
- User community readiness

are the keys to node enablement. The following sections briefly discuss each one of these factors with specific examples.

Cost Justification:

The main goal of legacy enablement is cost savings and justification for new expenses. For example, a global enterprise that has 50,000 corporate SAP users who do not need fine level access control can save money on at least 40,000 SAP client licenses by providing screen scrapping solutions to emulate SAP screens on desktop clients. This represents a huge saving with respect to the corporate budget, and the users will only experience a minimal performance impact. However, if there is a need for fine level access control and a need to maintain user identity, a different solution may be needed. The other dimension to cost savings from node enablement is to use existing hardware (powerful to emulate terminals or thin clients, but not as powerful for desktop processing) and not to spend any more money for new hardware. An additional advantage in this model is the savings from training expenses not needed because of the simple user interface format.

Life Expectancy:

It is very important to project the life expectancy or obsoleteness of the underlying system over a period of time before investing in node enablement. For example, if the business model or product lines have changed, your enterprise may not depend on the old catalog or product information systems. In this case, it is easier to opt for a new system while the old system may exist on a short-term basis. Sometimes it may be cheaper to build a new system rather than extending a functionally required old system that is expensive to extend or upgrade. It is important to identify the influencing factors and carefully evaluate before making a decision.

Reusability Factor for Legacy Systems:

As discussed in the approach section, the reusability factor of legacy systems depends on the kind of business practices modeled and how they need to be evolved for current needs. A simple example is the basic selling business process with manual intervention for specific discounting models. In this case, the base-pricing rule can still be leveraged. In contrast, a manufacturing process modeled within an ERP system may not hold if the business model of an enterprise has changed to outsource it's manufacturing. In this case, the reusability factor is zero.

Level of Desired Integration:

The level of integration depends on the kind of interaction needed for other systems depending on the legacy node. It can range from data attributes, to business objects, business processes, and UI levels. If the underlying node does not interact with any other systems UI level would suffice. If a specific rule has to be triggered on a data attribute level (example: total order price exceeding 100,000$), then a data attribute level integration is needed. Companies like Vitria (data level), SEEC (business objects, processes), and Citrix (UI level) provide different levels of integration[11].

User Community Readiness:

This brings the people factor (which is the toughest) into the picture. Unless the user community is ready to migrate to a different interface, anything other than UI level interface (emulation) will not help. A slow and "evolutionary" migration approach would be more appropriate in this scenario.

Approach

The purpose of node enablement is to leverage existing resources: people, software, and hardware with a thin overlay of new or creative devices or methodologies. Node enablement can happen in multiple levels:

- Application level – application as a window.
- User interface level – rendered contents.
- Business process level – invocation of reusable well-identified business process snippets.
- Business object level – usage of business object as a whole.
- Data level – directly tapping into the data repository.

The node enablement process should include evaluation of the following:

- Predicting the reusability criteria of the component or application over a period of time.
- Granularity of reusability – right levels as listed in previous section.
- Reuse vs. restructure vs. rewrite.
- Model human interventions and defining semantic gaps.
- How to handle exceptions, data type mismatches and transactional behaviors.

The key challenge is to decide when to move from the legacy to a new environment. The easier answer is to encapsulate the legacy systems with right abstractions treated as a black box.

Technology Requirements Outline:

Various ways for node enablement at different levels include:

Application Level

- Application Semantics – Client/Server.
- Active X/COM.
- Applets.
- Wrap existing client objects for Unix platforms – "Not available".

UI Level

- Thin Clients
    - XFORMS/WSXL.
    - WSXL.
    - HTML.
    - XML/XSL.
- Native Clients – Application Level emulations
    - Mainframe emulators.
    - X windows.
    - Raster graphics rendering.

Business Process Level

- Macro recorders and invocation shells – "Not available".
- Code extractors.
- Entry point APIs.

Business Object Level

- Standard APIs.
- XML Object extractors.
- XSLT – Translators, EDI-XML interfaces.

Data Level

- Database interfaces.
- File repository translators.

We need to identify a good player in this field and brainstorm this outline with them.

Appendix B – Requirements for iBase

Information Base (iBase), also known as XML database[2], is the key differentiator for this model. iBase provides an universal schema for all types of data objects and models in a loosely coupled way. iBase involves the notion of "**Beyond Databases to Data Sources**". As explained by Scott Dietzen[2] (CTO of BEA), much of Web data is not in the database and service-based architecture has to hide the data model. iBase will be based on XMLschema for object definitions, meta-data structures, and some extensions for simple rule definitions and relationship definitions. iBase will provide a language independent environment to define and implement methods. iBase will support encapsulated object models comprehensive enough to support data objects from various sources including RDBMS, ODBMS, flat files, Web services and sites, main frames, EDI, or any data source. It will include translators (e.g. EDI to XML). Key characteristics of iBase include query capability (XMLQuery), transaction support, caching, aggregation semantics, ability to create taxonomies, categories, groups, and ability to define navigation methods (depth vs. breadth first), validation and exceptions. iBase should leverage existing databases, but in addition should provide all the essential characteristics as listed above.

iBase in our opinion is not a radical concept. It is long standing requirement and each enterprise will decide to implement in own way. The Boeing[6] case study is a great example. The challenge is to define the common schema that can cover comprehensive requirements of a corporation. Current advancement in technology standards (XMLSchema, XPATH, XQuery, XAML, WebDAV) makes this concept implementable. The only challenge is to develop an efficient storage format for XML documents (tree structure vs. relational tables and object database formats). But, a simple XML document (with the right constraints) can be modeled in relational databases. Mapping existing schemas to iBase schemas is very achievable with transformation engines like XSLT, commercial products like Mercator, and standard ETL tools available from data mining companies. The key is leveraging existing tools for new applications. Support for both asynch (event based, pub/sub, queue based) and sync messaging is required depending on the data sources involved. Separation of logical business object layer (business object manager) from aggregation layer responsible for validations, caching, transactions (adaptors) and extraction layer responsible for connection management, query mapping, result generation and mapping (connectors) is very important to iBase implementation. Standards like JCA, JMS from J2EE and standards organizations like OpenAdaptor.org can be leveraged to implement the layers.

iBase at its core should provide a framework to support Metadata, Meta schema, Meta caching and Aggregation. The second level of services should include messaging, translations, XML repository support, and synchronization with sibling data via federation. Synchronization and translations driven by quality of service can result in different contexts for operating on the same data. For example, a customer's data aggregated from 20 different systems has to be synchronized in real time for an order approval process, while this is not the case for an order history summary report.

Approach

iBase provides an universal schema for all types of data objects and models in a loosely coupled way. iBase utilizes the notion of "**Beyond Databases to Data Sources**". IBase provides data repository, query, and transaction capabilities for any data source.

iBase should provide a configuration-based, plug and play approach to add, modify and operate data sources. The configuration mechanism should allow namespaces and scoping definitions for these data sources. Any mapping details and schema transformations should be able to be registered in a registry such that iBase can utilize that information during runtime for transformations and mappings.

iBase Structure

- iBase will be based on XMLschema for object definitions, and meta-data structures.
- iBase will support encapsulated object models comprehensive enough to support data objects from various sources including RDBMS, ODBMS, flat files, Web services and sites, main frames, EDI, or any data source.
- iBase should support a staged repository or a "transparent connectivity" to the underlying data sources based on configuration.
- iBase should support some extensions for simple rule definitions and relationship definitions based on an XML derivative.
- iBase will provide a language independent environment to define and implement methods.

Key characteristics

- Query capability (XMLQuery).
- Transaction support (XAML).
- Caching both results based and aggregated objects from backend – "Not Available".
- Aggregation semantics – like relational join.
- Ability to create taxonomies, categories, groups, relationships.
- Ability to define navigational methods (depth vs. breadth first).
- Ability to handle validations and exceptions.

iBase should leverage existing RDBMS and ODBMS database at the same time the very basic paradigm mismatch between relational, hierarchical (network) and tree (XML) structures may demand a new architecture for supporting iBase. The query language should support both native (SQL in case of RDBMS) and canonical (XMLQuery) for any type of data repository format. The iBase should expose only one type of schema structure (XMLSchema) irrespective of the underlying system's native format (SQL tables in case of RDBMS). The translations and performance impacts must be transparent to the user. It must be the responsibility of the underlying system to resolve these issues.

Questions to think about

- How to come up with the canonical repository format for tree structured XML documents?
- How to support 2 phase commit for various backend devices without much semantics agreement?
- How to make sure ACID properties are supported for all iBase objects?
- How to handle data rich (like multimedia) elements?
- How to support data objects from runtime systems based on various programming languages?
- Concept of Universal Object Locator?

Appendix C – The Asera Approach

We will be discussing how Asera solves these problems to cater the needs of Real Time Enterprises in a separate white paper (interested readers are encouraged to access the materials from Asera's Website http://www.asera.com). This section is a precursor to that paper. Our intention is to showcase Asera as a case study practicing many of the guidelines we outlined. Interested readers are encouraged to approach the authors for additional details. Our goal is to illustrate that these guidelines are feasible to implement and that we have successfully deployed customers using solutions built around these concepts.

As mentioned in the main sections, we paid special attention to this set of needs when founding Asera. Asera's design philosophies are based on the ground rules we listed for designing architecture:

- Open standards and multiple vendors.
    - Best of Breed using XML, Java and HTTP/Web services.
- Flexible, loosely decoupled interfaces considering rapid changes and evolutions of underlying systems.
    - Plug and Play approach with abstracted interfaces (Web services) to component services.
- A production system that can manage, monitor, and support versioning of the Web services life cycle.
    - A service-oriented architecture delivering infrastructure utilities as services.
- Ability to leverage existing systems and reduce cost.
    - Seamless connectivity to existing systems in all levels (data, object, API, business process and presentation).
- Flexibility as more critical than optimization for cost, performance or features.
    - Fast deployment cycle and ability to configure and customize to evolving customer needs.

Asera's mission is to provide "Solutions differentiated by Infrastructure Services". Asera's differentiators are based on the meta-architecture approach (built on standard app servers), XMLSchema-based business object model support (iBase approach), and process enabled via a business process management framework (XML based workflow engine) with support for customization and declarative business rules. Asera's solutions are built on the Service Oriented Architecture and assembled from building block components. Asera offers globalization, personalization, entitlement, business process automation, rules execution, business object life cycle management, global session management, single sign on, portal aggregation, caching, monitoring, and persistence connectivity, etc. These services are available for any application that is developed on Asera's infrastructure. Asera's solutions are assembled from common building blocks like Catalog List Manager, Inbox Manager, Approval Routing Manager, Task Manager, Generic Search Facility, Content Manager, Attachments Manager, Data Object Import/Export Manager, Email Notification Manager, and Menu Manager, etc. Asera's solutions are delivered as horizontal business processes, assembled from building blocks combined with specific data objects. For example, an Order Fulfillment solution is built using a variety of tools:

- The Catalog List Manager for catalog display;
- Approval Routing Manager for order entry approval;
- Attachments Manager for line item details attachment;
- Data Object Import/Export Manager for local upload/download;
- Email Notification Manager for order status notification;

- Menu Manager for rendering Web pages; and
- XMLSchema-based business objects like Order Object, Customer Object, and Product Object.

Asera's horizontal business processes are targeted for different vertical domains (e.g. Electronic/High Tech Manufacturing or Chemicals/Process Industries), enabling fast deployment using Asera's ability to configure/customize these solutions declaratively.

Asera currently leverages its own integrated development and deployment environment (IDDE) to model business objects, develop workflow, define business rules, declaratively define presentation components and manage upgrades. Though Asera leverages its own IDDE, the IDDE is loosely coupled with standard toolsets like development IDE (Visual Café), code versioning system (Clearcase) and HTML authoring tools (Frontpage). The customized versions of workflow or business rules can be merged and tracked for changes across versions using Asera's development environment.

Here are few areas Asera will be working in the future:

- Provide end user enablement via mass customization (i.e. use Visio for workflow modeling and Excel to define business rules). For example, a business analyst can define a collection of template business processes and have every interested individual within that organization/partner enterprises modify, use and manage their own versions of these customized business processes through declarative configurations.
- Provide ability for end users to assemble their own business processes from basic building blocks and a library of business objects.
- Provide ability for other systems to call into Asera's infrastructure and leverage services as and when they need - similar to a component-based service offering or "what you need is what you use".