Application Integration: Now That It's Mainstream, What's Next?
"Application" doesn't mean what it used to mean. Service-oriented architecture (SOA), the Web, event-driven architecture (EDA), business process management (BPM), integration and virtualization are changing the fundamental nature of business computing. But what is really new and what is timeless and inherent? This scenario summarizes current trends in application architecture and provides guidelines on what mainstream companies should do.

As multienterprise integration skills and technology mature, the concept of the "virtual enterprise" becomes more widespread and important every year. The boundaries of the conventional enterprise are increasingly porous, as information, goods and services flow more quickly within a company and out to its customers, suppliers and other partners. A virtual enterprise can be agile only if its components act in a coordinated manner, and if its processes can change readily as business requirements change. To successfully implement the virtual enterprise, CIOs must rethink the IT organization, application architecture, application acquisition strategies and their working relationships with the business units. IT must provide applications with more flexibility than ever before. Virtual enterprises often use virtual ("composite") applications that execute an activity by involving parts of two or more physical application systems, which may be heterogeneous in their technology and design, and owned and managed by disparate business units.
Key Issues

• What will be the business justification for expanding investments in SOA and application integration?
• How will the technology for SOA, EDA and Web services evolve?
• What will be the challenges, solutions and best practices for data integration, process integration and composite applications?

IT is pervasive in all midsize and large enterprises. Business strategies have become intertwined with IT strategies because almost any change to a business process, product or service requires change in the application systems that support it. You can't have enterprise agility unless you have IT agility. However, IT is too often viewed as an impediment to agility, instead of being an enabler of agility. When you need to change something, traditional applications are expensive and slow to change. Enterprises need to continuously refine the way they produce goods and run internal operations, so IT needs to change its approach to application architecture.

The state of the art in application design and application integration is therefore undergoing a transition because of escalating demands from the business side of the enterprise. Companies want to modify their applications more quickly, but find that they're saddled with huge investments in hard-to-change legacy systems. They need to adapt their portfolio of legacy systems and implement new applications that are designed from their inception to enable ongoing modification at low cost. Application integration is defined as "making independently designed applications work together." Every application development project is, in part, an integration project. However, no project is purely an integration project, because every project includes some new development of logic and data.
Key Issue: What will be the business justification for expanding investments in SOA and application integration?

The scope of application and process design has expanded over the years. Most of the energy that went into early "stove-pipe" record-keeping applications was spent just getting the individual systems to work well. During the 1990s, business analysts and software engineers began paying more attention to the end-to-end business processes. Business process re-engineering (also called business process improvement and BPM) aimed to improve the efficiency and effectiveness of the entire sequence of activities in a value chain. Steps were combined, modified or eliminated, and some processes were made "straight through." Improvements were made in the business processes and the application systems that supported them. Line-of-business (LOB) managers, analysts and sometimes even software engineers are now further expanding the scope of analysis to consider activities upstream and downstream of a value chain, and the interactions that occur between steps in separate value chains. For example, they may also be able to generate economies of scale by applying the same resources to multiple similar, but separate, processes. The idea is to optimize the company as a whole rather than focusing exclusively on one activity or process.

Action Item: Business analysts and application architects should work with LOB managers to improve all three levels of business architecture (individual steps, end-to-end processes and interactions across processes and steps in the value grid) and their counterparts in the application architecture.
Many functions in a modern enterprise are organized in the form of services. Companies use express mail services, cleaning services, auditors, advertising agencies, travel agencies, personnel recruiters, copy services, security services and many others. A service is a way of structuring work. One party, the service provider, performs a function to assist another party, the service consumer. Services make the company's organization and its business processes easier to understand, manage and change. A service is a consumer's view of a service provider's capabilities. Complicated jobs can be accomplished by combining multiple simple services. Services are described by a formal or informal contract that says what will be done, not how it will be done. Services can be shared: multiple people, departments or even separate companies can use the same service provider.

The same characteristics that make the service concept helpful in organizing business units are used in the design of SOA application software. However, software development is a relatively unforgiving discipline, so SOA requires a degree of rigor and precision not always required in non-automated services. Gartner coined the term SOA in 1994 and published the first report on SOA in the industry in 1996. The main benefit of SOA is the shorter time-to-change.

**Action item:** Companies that adopt SOA should proceed with a succession of small projects, measuring project success against business and technology metrics.
In traditional IT architecture, all application-level intelligence (business-level logic and data) is held within separate application systems that communicate with each other relatively infrequently. The network is a simple pipe for moving packets, and integration logic is hard-wired into the sending or receiving application modules. The applications have to be smart because the network in the middle isn't. This familiar architecture is gradually giving way to new forms of application architecture built on an intelligent, middleware-based SOA and integration infrastructure based on an SOA backplane. This approach changes the meaning of the term "application" because processes are no longer organized around discrete application systems. Instead, the control of automation shifts from the endpoint applications into the "glue" or connectivity "fabric" in the middle. Applications become flexible coalitions of service components that cooperate to execute composite activities and multi-step end-to-end business processes. The infrastructure implements functions such as orchestration, transformation, content-based routing and monitoring using services that are plugged into enterprise service buses (ESBs), middleware appliances, integration suites, application platform suites and other SOA backplane middleware. This organizing intelligence is "softwired" using model-driven development tools so that it can be modified more readily. End-user access functions and the back-end data-facing service components are treated as endpoints in this architecture.

Action Item: Business executives and IT leaders must understand the implications of an SOA infrastructure because those who do not realize that they are implementing one will end up with a bad one.
Business applications traditionally consisted of a few large programs that were hard to change. Monolithic applications were a custom-fitted sequence: end-user-facing presentation logic, business logic, data logic and data. In contrast, SOA is an architecture style founded on modularity and layering. Discrete functions are packaged into encapsulated, shareable modules (service provider components) that can be invoked in a loosely coupled manner by multiple disparate local or remote consumer modules. The metaphor of a "black box" describes how encapsulation hides the internals of each component. An SOA service is the contract between a consumer and a service provider — it's the consumer's view of a provider's capability. SOA applications are developed in separate stages, generally by two or more kinds of developers.

Service components in the back-end layer implement sharable ("reusable") functions encompassing data and business logic. Other developers focus on the user-facing (presentation) layer and the middle integration layer (often including orchestration functions). SOA applications can be developed, extended and maintained in small, easily understood increments, facilitating an "organic" approach to ever-changing business applications. The SOA domain starts small and gradually expands to handle additional tasks and business processes. Each application or business process might use one or many services — the percentage of its function implemented in SOA services may vary greatly.

Action Item: Organizations should become familiar with SOA-enabling technology and learn relevant technical and organizational best practices, as soon as possible.
Sharing Data Works, but Only Within a Limited Scope

**Problem:** Stand-alone stove-pipe applications have redundant data which are hard to keep consistent.

**Theory:** Applications that share one database can avoid data redundancy, thereby avoiding inconsistent data.

**Reality:** A large enterprise has disparate versions of certain data in multiple business units and applications.

Modern companies have redundant versions of data on customers, products, employees and other important entities. Redundant data has obvious drawbacks, particularly with respect to synchronization. When one copy of the data is updated, the change must be propagated to all the other application systems that hold data on that entity. In theory, an enterprise should consolidate data into a single database to be shared in real time across all application systems. Database management system (DBMS) software that enables this, at least from a technical point of view (transaction atomicity, data integrity, concurrency control and durability of updates), has existed since the 1970s. In practice, companies are only able to share data among a limited set of applications operating within one domain under the control of a single IT group.

Many sound business reasons explain why it is often less expensive and more effective to maintain redundant, overlapping copies of the data. For example, a purchased package may be predicated on its own data model and it could cost too much to modify the package to work with another database. Or a particular new application may require a variation on the data model because it needs new attributes. In some cases, data duplication results from poor development practices or irrational needs to control one's own data, but, in many cases, data duplication is pragmatic and sensible. Master data management (MDM) strategies confine the use of duplication to appropriate, limited circumstances.

**Action Item:** Companies should migrate their data consistency solutions from point-to-point batch jobs to message-based transfers where business processes will benefit from improvements in speed or quality.
Sharing SOA Services Works, but Only Within a Limited Scope

**Problem:** Stand-alone stove-pipe monolithic applications have redundant logic and data.

**Theory:** SOA applications can share common services (a service provider component typically includes logic and data).

**Reality:** A large enterprise will have disparate versions of certain SOA services in multiple business units and applications.

As a general goal, developers should attempt to share SOA services rather than implement their own new versions of the services. Creating a new service rather than sharing a pre-existing similar service is not usually a reflection of a character defect in the software developers. Just as many sound business reasons exist to implement partially redundant copies of data, many good reasons exist to implement a new version of a service. For example, the effort to modify a new packaged application to use a previously installed service in a legacy or packaged application may be prohibitive. It is often better to use the new version of the service that comes with the package. Or, when building a new custom application, the unique nature of the business requirements may require implementing a service that has some overlap with older services but which cannot be fully satisfied by the older service.

As a result of these factors, SOA only eliminates duplication of logic within a set of applications that is developed by one team or a group of cooperating teams; it cannot eliminate redundant logic and data across the whole enterprise. A set of SOA services (a domain or "service community") is internally coherent because it grows serially, and its developers have knowledge of each other's interface specifications and data models when they design new modules, databases and processes.

**Action Item:** Developers should share services wherever practical, but architects and managers should recognize that some partial duplication of services is as inevitable — and actually desirable — as having some duplication of data.
ESBs are a relatively new kind of middleware that combines support for SOA and Web services with features from several older types of middleware. "ESB technology" has a clear meaning but the term "ESB product" is somewhat arbitrary. ESB technology has five required aspects. An ESB must: 1) implement program-to-program communication; 2) support core Web and Web services standards including XML, HTTP, SOAP and WSDL; 3) support service binding, in conjunction with an embedded name space or external registry; 4) support typed messages to enable functions that require an understanding of the message schema; and 5) have an extensible, intermediary-based architecture so that added features can be plugged in. ESB products contain more than simple ESB technology; they bundle some set of value-add plug ins and related features. The label "ESB product" overlaps several other categories of SOA infrastructure products, such as SOA suites, business services fabrics, integration suites, integrated service environments (ISEs), BPM suites (BPMSs) and application platform suites (APSs). In most cases, companies will acquire ESB technology as part of one of these larger SOA infrastructure products or in a packaged application, rather than as a stand-alone capability.

Action Item: Companies that deploy large-scale or long-lived SOA applications should use an SOA backplane based on ESB technology to mediate the interactions among service components; point-to-point connectivity is not enough.
SOA applications can be categorized into two broad scenarios:

**Uniformly-designed SOA service communities** are custom-built incrementally over some period of time. Developers share interface definitions (for example, using WSDL files), database schemas, process models and software technology decisions, so the components they write are inherently compatible with each other. There is no duplication of data or logic and technology is almost always homogeneous.

**Application integration scenarios** encompass independently-designed systems, including purchased packages, legacy applications and software from autonomous developers. Some of these integration scenarios, particularly those that use asynchronous communication, involve few or no new software components or databases. However, some other integration scenarios involve composite applications which are a mix of some new service consumer components and previously-built, heterogeneous, independently-designed service provider components (sometimes wrapped legacy applications). SOA infrastructure products that are aimed at these scenarios include integration features such as transformation, protocol gateways and adapters.

**Action item:** IT managers and architects must evaluate their business requirements and architectural strategy to understand how much of each style of development they will pursue because the answer to that question will determine which type of integration infrastructure products will be most appropriate.
Background/Tutorial: The drive to achieve uniform design, data sharing, service sharing and company IT standards is forever at odds with the forces that cause application and technology diversity.

The Tension Between Uniform Architecture and Heterogeneity

Uniform Design
- Central control and funding
- Custom applications
- Rip-and-replace
- Centralized architecture
- Cooperating developers
- Consistent data model
- Shared logic and data
- Homogeneous technology

Application Integration
- Decentralized decisions and funding
- "Buy before build" strategy
- Leverage legacy applications
- Dispersed architectures
- Autonomous developers
- Diverse data models
- Redundant logic and data
- Heterogeneous technology

Many IT managers and architects initially chart a course to pursue the uniform development scenario (left side above) because it seems logical to share services and eliminate redundant data and logic. However, as new applications and business processes are implemented, analysts find increasing need to leverage some externally-based services, legacy applications and purchased packages. This motivates the use of composite application design patterns (middle of the chart) in which the interface definitions are copied from previously designed services. Leveraging services from several different sources begins to introduce some overlaps in databases and some conflicts in semantics. The application portfolio begins to incorporate data models, process models, service interface definitions and other aspects of system design that diverge from the internally-developed application architecture. Then, situations that require asynchronous integration arise that push the strategy further to the right. For example, a business unit may find that it cannot rely on a composite application that is dependent on an external SOA service because of availability, reliability, security or cost considerations. Next, the business unit will typically build a new, local version of the data and logic under its own control (it may reconcile this new data with external data using an asynchronous transfer of database updates). Such organizational, technical and business realities cause a type of "entropy" that pushes companies toward the right, away from the purity of the uniform scenario.

Action item: Analysts and application architects must balance the tradeoffs between uniformity and heterogeneity in all projects.
SOA infrastructure suites that are targeted primarily for uniform design scenarios, such as APSs and integrated services environment (ISEs), always include a general-purpose application server and related development tools for building data-facing aspects of new applications. Such products tend to downplay integration features, particularly those for interoperating with competitors' application servers and for file-oriented asynchronous integration. By contrast, SOA infrastructures that emphasize integration scenarios are optimized to work across heterogeneous application servers. Such products may be categorized as integration suites, SOA suites or (if they are simple SOA backplanes without a lot of accessories) plain "ESB products." Vendors of these products assume that customers will use plain old java objects (POJO), Java servlets, .NET or a JEE application server from another vendor to host any new data-facing logic. ESB-based SOA infrastructure products provide load balancing, failover, transactions and other features to augment basic application platforms bringing them into competition with JEE and .NET application servers. New ESB-based SOA suites are also adding new features for deployment, policy management and service assembly using the emerging service component architecture (SCA) or Windows Communication Foundation (WCF) specifications. Action Item: Architects should understand what SCA and WCF do, and consider including products that implement them in SOA development methodologies and middleware infrastructure.
Application Integration: Now That It's Mainstream, What's Next?

Decision Framework: The scope of a middleware selection decision can be limited to a specific project, directed at all new applications within a business unit, or intended to serve as an enterprisewide standard.

Examples of SOA Infrastructure Products

Application Server-Centric
- "APS"
- "ISE"
- "Business Services Fabric"
  - BEA SOA 360 (AquaLogic Service Bus)
  - Compuware Uniface
  - Cordys Platform (Cordys ESB)
  - Fujitsu Interstage (Service Orchestrator)
  - IBM WebSphere Business Services Fabric (WebSphere ESB)
  - JBoss Enterprise Middleware Suite (JBoss ESB)
  - Magic Software iBOLT Business Integration Suite
  - Microsoft .NET 3 (WCF)
  - Oracle Fusion (Oracle ESB)
  - SAP NetWeaver
  - Sun Java Composite Application Suite (CAPS ESB, Sun Open-ESB)
  - Sybase EAServer

Integration-Centric
- "ESB Product,"
- "BPMS,"
- "Integration Suite,"
- "SOA Suite"
  - Apache/Logic Blaze Fuse (ServiceMix)
  - Apache/Iona Celtix Enterprise (CXF)
  - Axway Integration Platform
  - Cape Clear 6 Server
  - Fiorano Software SOA Platform 2006 (Fiorano ESB)
  - Iona Technologies Artix (Artix ESB) and Celtix Enterprise (Celtix Services Engine)
  - Intersystems Ensemble
  - MuleSource/Codehaus Mule
  - ObjectWeb Celtix
  - PolarLake Messaging Integrator
  - SOA Software Management Server
  - Software AG CrossVision (Service Orchestrator)
  - Sonic Software Sonic SOA Suite (Sonic ESB)
  - Sterling Commerce Gentran Integration Suite
  - Tibco BusinessWorks (Matrix Service Bus)
  - Vitria BusinessWare
  - webMethods Fabric

The SOA infrastructure middleware market is still evolving. There is no industry agreement on the terminology — as evidenced by the plethora of overlapping product categories and labels. Web services and Java-oriented standards for SOA and EDA are still being refined, and product packaging varies from vendor to vendor. IBM includes WebSphere ESB as part of WebSphere Process Server; it offers advanced transformation and routing in another product, the WebSphere Message Broker. Microsoft has packaged its WCF ESB technology into the operating system, and it offers transformation, BPM and content-based routing in a separate product, BizTalk Server. Several open-source SOA infrastructure suites are offered through Apache, Codehaus and ObjectWeb; these are supported by Iona Technologies, LogicBlaze, MuleSource and other companies.

Action Items: If the dominant mode of application acquisition is custom development on one designated application server, then buy the SOA infrastructure offered by that application server vendor.

If the dominant mode of acquisition is to leverage purchased applications, legacy systems or custom applications on heterogeneous application servers, then buy integration-centric SOA infrastructure that is optimized to support heterogeneous application servers.

Companies that prefer to assemble their own best-of-breed SOA infrastructure should buy or build ESB technology and add third-party products and custom code.
Large vendors promote "one-stop shopping," which suggests that customers buy the whole technology stack — including portal, application server, ESB, DBMS, development tools, integration tools and management tools — from one source (them). In theory, products in a stack may be pre-integrated, metadata can be shared and development tools may have a consistent look and feel. Of course, the stack-centric approach also maximizes account control for the vendor by locking the user into one infrastructure. Companies that sell hardware (for example, Fujitsu, Hitachi, HP, IBM and Sun Microsystems) include hardware in the bundle; others (the same vendors plus Microsoft) include the operating system; those that emphasize the DBMS (for example, Oracle and Sybase) may start the stack diagram at the DBMS level; some (for example, Oracle, SAP and Microsoft) extend the stack to include the application itself. The benefits of homogeneous infrastructure stacks can be substantial, but they are only fully achievable in a controlled environment under a single chain of command or under the direction of a disciplined and strongly enforced technology architecture program.

Action Item: Enforce a corporate standard for a single-vendor software stack only if development teams are under central control and most applications are custom developed or bought from the same vendor that sells the technology stack.
Most large companies have heterogeneous software stacks. They buy packaged applications from many vendors, retain legacy applications, and have multiple semi-autonomous business units that may not always use the software stack that is ordained by a central architecture group. A service provider component may run on an application server from one vendor, use a DBMS from another vendor and be hosted on an operating system from yet another source. It may also be invoked by a consumer component running on an entirely different software stack. The ability to mix and match disparate technology platforms within the same application and business process is a key benefit of SOA and EDA. Heterogeneity is practical because the coupling between software modules is loose and the SOA infrastructure (including the ESB if one is used) insulates consumers and services from each other. If all of the components share the same ESB, SOA and EDA applications are relatively simple to develop, deploy and maintain even they are on different stacks. But not all environments have just one ESB.

**Action Item:** Architects must coordinate their SOA, ESB, Web services and integration strategies because of the application design interdependencies and the overlap in the enabling middleware.
An ESB helps resolve the challenges that arise when using heterogeneous application servers, but what does a company do when the ESBs begin to multiply? No one wants multiple disparate ESBs, BPM tools, transformation utilities, metadata repositories, management consoles or other tools, but large companies are finding themselves deploying them anyway. The same factors that lead a large company to acquire heterogeneous application servers and DBMSs also cause them to acquire heterogeneous SOA infrastructures. Most companies will have three or more ESB technologies, largely because of the autonomy of the business units. More than 90% of SOA, EDA and integration messages are confined within one application domain (generally in one business unit), which can be internally consistent if it uses a single ESB (one ESB domain to match one application domain). However, some business units have two or more ESBs, and most must interact with disparate ESBs in other business units. The proportion of inter-domain traffic is increasing as business processes span organizational boundaries more frequently. Having disparate ESBs imposes redundant support costs and performance penalties and complexity in the gateways between the domains. Some protocols available in one ESB will not be supported precisely the same way in another because ESBs implement different dialects of the same protocols.

Action Item: Companies should anticipate having heterogeneous ESB technology and use an integration competency center or SOA center of excellence to cope with the resulting complexity, performance impediments and technical support expenses.
Application Integration: Now That It's Mainstream, What's Next?

Strategic Planning Assumption: By year-end 2007, approximately 67% of all large enterprises will have an SOA COE or ICC, up from approximately 50% at the beginning of 2006 (0.7 probability).

You can't buy agility, you have to work for it. Designing for change, and for sharing data, SOA services and business events, has little to do with technology and a lot to do with having the right IT organization structure and sensible development practices. The companies with the best chance of success implement an SOA COE or an ICC. A single COE cannot support a large company, so a centrally-coordinated federation of COEs will often be needed. The development side of each COE provides expertise to assist the application development teams in their SOA and integration work. It also should maintain metadata for SOA and integration, including the interface descriptions, canonical message and document definitions (for example, XML schemas, WSDL files and business process models) and their respective versions as they evolve. The operational side of the COE installs and maintains the SOA infrastructure. Companies should develop architectural standards for SOA and integration technology. Although most companies cannot converge on one ESB, orchestration engine or transformation tool, they should try to minimize the number of each of these. Development methodologies and programming models for SOA and EDA are still immature, but enough is known to enable the establishment of a reasonable set of preliminary company standards.

Action Item: Executives should create a full-time, centralized or federated SOA COE or ICC to design, deploy and maintain the SOA and integration infrastructure, including its middleware technology, SOA service definitions, business process definitions, EDA message schemas and other integration metadata.
Recommendations

✓ Optimize the enterprise, not just one application or process.
✓ Use SOA (interactive and event-driven styles) in all large, new business applications that are built or acquired.
✓ Add ESB technology to your IT strategic plan and application architecture.
✓ Implement a federated or centralized SOA COE or ICC.