Fawcette Technical Publications | Winter Issue 2006, Vol. 4 No. 1

Enterprise Architect

Features

Scaling Over Time
Alex Krapf discusses the problems surrounding "scaling over time" and provides some practical solutions for solving the version control dilemma.

Combat Increasing IT Complexity
Explore how companies can conquer issues associated with IT complexity and achieve success in their architecture initiatives.

Incremental Architecture
Discover an incremental approach that aligns projects with strategic goals.

If SOA Looks Hard, You're Looking at it Wrong
Deconstruct the way you think about a service in service-oriented architecture.

Down With Downtime
Implement automated business application processing to gain significant increases in efficiency and productivity.

The Elephant in the Room
Is your enterprise architecture an effective management discipline? Mike Dunham explains how governance is essential to EA management.

Maximize Service Reuse

Explore ways to maximize reuse in your service-oriented architecture

Enterprise Architect Summit 2006

Strategies and Best Practices for the Real World

Enterprise Architect Summit returns to Florida in May for three informative days of keynotes, workshops, and breakout sessions led by experts in the enterprise architecture field. Arm your business to respond to emerging IT challenges – register today.

The Ritz-Carlton Key Biscayne Resort, Florida

May 15-17, 2006

Sessions Will Cover:

- Putting the A in SOA
- Solving Real-World Architecture Issues
- Building an Agile Enterprise
- Key Strategies to Implement Security Policies
- Metadata: the Key to Understanding IT
- Joining Enterprises with the Global SOA

Register by March 22 and save $300.

Call 800-848-5523 today

or visit us online at

www.enterprise-architect.net/summit

FTP FAWCETTE TECHNICAL PUBLICATIONS

Take Your Business to the Next Level-

View the preliminary list of sessions, workshops, and events planned for May

Monday, May 15

8 a.m.-6 p.m.	**Registration**	
8 a.m.-3 p.m.	Enterprise Architect Classic Golf Tournament	
	Workshops	
1-5 p.m.	**Best Practices:** Security Policy Development and Enforcement	**Strategy:** IT Drivers for Business Process Management
6-8 p.m.	Welcome Reception	

Tuesday, May 16

9 a.m.	**Keynote**	
	Best Practices	**Strategy**
10:30 a.m.	Putting the A in SOA	Solving Real-World Architecture Issues
11:45 a.m.	Best Practices for Database Connectivity in the Midst of Infrastructure Diversity	The Enterprise Architecture Office and the Ever-Increasing Organizational Need
12:45 p.m.	Lunch	
2 p.m.	**Keynote**	
3:15 p.m.	The Rocky Road to Compliance	*Panel:* Key Strategies to Implement Security Policies
4:30 p.m.	Building an Agile Enterprise	Metadata: The Key to Understanding IT
6 p.m.	Exhibitor Reception	

Wednesday, May 17

9 a.m.	**Keynote**	
	Best Practices	**Strategy**
10:30 a.m.	Model Driven Software Engineering	Joining Enterprises with the Global SOA
11:45 a.m.	*Panel:* Perspectives on Architecture Modeling	The Data-Centric Enterprise: A Blueprint for Enterprise Architecture
12:45 p.m.	Lunch	
2 p.m.	Managing Dependencies Across the Architecture	Selecting SOA Infrastructure
3:15 p.m.	Minimize Business-Disruption Risk and Cost Through Architectural Modeling	SOA Operations and Governance to Move Applications to the Network Architecture
4:30 p.m.	Build an Enterprise Security Architecture	SOA Models and Methodologies

Enterprise Architect

Contents

Winter 2006, Vol. 4, No. 1

COVER STORY

DEPARTMENTS

GO ONLINE @ www.enterprise-architect.net

■ Visit *Enterprise Architect* online for exclusive interviews, extended content, and more.

FTPOnline Blogs

Check out the FTPOnline blog page for insights from FTP President Jim Fawcette, contributor Peter Varhol, and other FTP editors as they sound off on IT issues.

Video: Architecting for Scalability

Amazon's Pat Helland outlined what's necessary to build a scalable application for Software Architecture Summit attendees in San Francisco last month. Plus, he provided the design patterns you must follow in order to redeploy without writing new code.

Video: Business Process Modeling

Modeling and execution language, combined with service components and service-oriented middleware, make it possible for you to automate your business processes. Ted Buszkiewicz examines the technology that makes this possible.

Video: Moving Beyond SOA

John deVadoss introduces you to new architecture that moves beyond SOA and finds a way to connect cutting-edge software development and existing services infrastructure.

Enterprise Architect

Winter 2006

enterprisearchitect@fawcette.com

Jeff Hadfield, *Vice President of Publishing*

Editorial
Nina Goldschlager, *Managing Editor*
Lauren Dresnick, *Associate Editor*

Editorial Advisory Board
Janaki Akella, Chris Barlow, Toufic Boubez, Mike Ellsworth, Dan Foody, Steve Gillmor, Boris Lublinsky, Richard M. Marshall, James McGovern, John McDowall, Patrick Meader, Richard Murphy, Tom Pardee, Ken Rutsky, Lee Sherman, Vinu Sundaresan, Gordon Van Huizen, and Peter Varhol

Art & Production
Michael Hollister, *Vice President, Art & Production*
Bruce Gardner, *Senior Art Director*
Brian Rogers, *Art Director*
Kathleen Sweeney Cygnarowicz, *Associate Production Manager*
Lyndon Lloyd, *Senior Interactive Art Director/Web Producer*
Shane Lee, *Associate Web Producer*

Advertising Sales
Roy Kops, *Advertising Director*
Lisa Sidlow, *Western Regional Sales Manager*
Dennis Leavey, *Eastern Regional Sales Manager*
Susan LaCroix, *Executive Assistant to the Vice President of Publishing*

Circulation
Karen Koenen, *Senior Circulation Director*
Fred Perry, *Circulation Manager*

Marketing
Susan Ogren, *Marketing Manager*
Margaret Horoszko, *Senior Designer*

Conferences
Tim Smith, *Vice President, Conferences*
Brent Sutton, *Associate Conference Director*
David Seymour, *eAdvertising Manager*
Katie McGillivray, *Marketing and Editorial Planner*
Will Hansen, *Operations Planner*
José Porcell, *Customer Service Representative*

Operations
John Sutton, *Executive Vice President/Chief Financial Officer*
Darlyn Phillips, *Director of Finance*
Betty Tsang-Hwah Wu, *Staff Accountant, Cash Management/Payroll*
Elena Ostrovsky, *Staff Accountant, Accounts Payable*
Iain Neillands, *Collections Analyst/Cash Management Accountant*
Tin Cao, *System Administrator*
Pamela Davis, *Human Resources Manager*

FTPOnline
Nina Goldschlager, *Managing Editor/Business Unit Manager*
Lauren Dresnick, *Associate Editor*

Fawcette Technical Publications
James E. Fawcette, *President*
John Sutton, *Executive Vice President/Chief Financial Officer*
Aaron Weule, *Vice President, Chief Information Officer*
Michael Hollister, *Vice President, Art & Production*
Tim Smith, *Vice President, Conferences*
Jeff Hadfield, *Vice President, Publishing*
Wilson, Sonsini, Goodrich & Rosati, *Corporate Counsel*

Contact the Editors
enterprisearchitect@fawcette.com

Enterprise Architect Online
www.enterprise-architect.net
The online home of *Enterprise Architect*, with articles, expanded features, and more.

Media Advertising
www.ftpmediakit.com

Enterprise Architect (ISSN: 1547-4569) is published by Fawcette Technical Publications Inc., 2600 South El Camino Real, Suite 300, San Mateo, CA, USA, 94403. Tel. 650-378-7100; Fax. 650-570-6307. Customer Service: For subscription orders, inquiries, or address changes, call (866) 387-5776; international inquiries call (847) 559-7309; send a fax to (847) 291-4816; e-mail ea@omeda.com; or write to *Enterprise Architect*, PO Box 3484, Northbrook, IL 60065-3484.

Moving Beyond SOA to the Web

by Jim Fawcette
President, FTP

Seven years ago, this publishing company acquired a series of Web conferences. Our thinking was that Web design couldn't remain a totally separate discipline from software architecture. We thought the Web would become a development platform and we'd leverage our strengths in IT-oriented publishing to span both disciplines.

Well, it took a lot longer to materialize than we thought, but suddenly 2006 is becoming the year that the IT community decides it must do more than focus solely on the server and must stop treating user interaction with Web-based applications as merely a design issue of colors and interface elements.

Part of this is driven by competitive positioning. The community of everyone-but-Microsoft originally pushed browsers delivering simple HTML as adequate for every use. In true "if all you have is a hammer, everything looks like a nail" fashion, IT vendors that failed to compete with Microsoft in selling PC software declared that the PC was dead, that all you need is a browser. And for a while, that movement had tremendous momentum.

But after a rapid rush to the browser, the user community recoiled, asking for all the UI elements it had gotten used to on the PC—whether you derisively call them "fat clients," or in Microsoft-speak praise them as "smart clients." I remember one consultant quoting an IT client, "We paid $10 million for this application and I have to tell people, 'Be careful not to hit the Back button or hit Refresh.'"

In late 2004 to early 2005, some in the Microsoft ecosystem (although not Microsoft itself) went as far as proclaiming that "HTML is dead," because browser development had basically stopped.

Adam Bosworth, now at Google but formerly of BEA, Crossgain, and Microsoft, was one of the early visionaries to discuss how a rich user experience could be delivered through the browser, and extensive efforts at www.eclipse.org were put behind tools that downloaded and ran in the browser.

The materialization of Ajax (Asynchronous JavaScript and XML), RSS, Ruby on Rails, Wikis, instant messaging, and bots is making the Web emerge as a development platform. Even if, like me, you have to hold your nose at the hype of Web 2.0, the moniker is valuable in that it offers a single term to encompass all these technologies.

The next step is to bridge the rich Web with SOA, to deliver applications that are robust and scalable but also usable.

In his opening keynote for our Software Architecture Summit, John deVadoss, director of architecture strategy at Microsoft, said that "there is something fundamentally happening… and if Web 2.0 is one end, then SOA [service-oriented architecture] is the other."

As deVadoss described in an *eWeek* article by Darryl K. Taft, "The consumer edge is the peer-to-peer, Web 2.0 world and the enterprise edge is the SOA, ESB (enterprise service bus) model. In addition, the consumer edge is an asynchronous communications model based on the REST (Representational State Transfer) scheme, and the enterprise edge is based on the Simple Object Access Protocol scheme.

"'REST is a dominant model on the consumer side, and SOAP is the model on the enterprise side,' deVadoss said.

"'As architects we have to think very hard about what's happening on the consumer edge, this Web 2.0 edge… We could wait, but I believe this is the cusp,' deVadoss said.

"'These edges are bridging. It's time we put the user back into SOA.'" ■

Maximize Reuse of Services Within Your SOA

Leverage existing infrastructure, create appropriate organizational structures, and adopt collaborative development practices.

by Theo Beack

Service-oriented architecture (SOA) is one of today's hottest topics. No matter where you go, or which customers you engage, the same topic is foremost in their minds. People want to know how to use or build an SOA to (a) solve some of the most pressing business problems, (b) create more integrated business solutions, and (c) reduce the cost of building and maintaining existing IT infrastructure and applications.

Much has been written on the value of SOA and how it will help organizations meet all the aforementioned objectives. How to design and build a solid SOA is another common topic, and it appears that everyone has an opinion on what constitutes a "service." IT architects love to discuss the best approach for designing an SOA. Yet, it appears that relatively little has been written about one of the most important aspects of SOA: how to foster reuse of existing services.

In this article, I will explore one of the prime reasons we are using SOAs to solve the most crucial integration and business problems, namely the reuse of existing applications, business processes, and infrastructure. Many organizations struggle to reach wide adoption of services within their SOAs, despite creating them with reuse in mind. This article explores the ways in which practitioners of SOA

Theo Beack *is Chief SOA Architect at Software AG North America. For the past nine years, Theo has focused almost exclusively on integration technologies and has extended his skills to become an expert in service-oriented architecture, XML integration, Web services, enterprise information integration, and semantic integration.*

can maximize the adoption and reuse of services by leveraging the existing infrastructure, creating appropriate organizational structures, and adopting collaborative development practices.

SOA Assumptions

When reading SOA-related articles or conversing about SOA, the majority of people often make the same basic assumptions:

- You don't need to rip and replace existing systems.
- SOA and the use of Web services standards help ease the pain of integration.
- SOA allows the reuse of existing applications (home-grown and packaged).

Everyone seems to agree that reuse is one of the prime reasons for SOA's widespread use and success. Few question what "reuse" really means, or how it can be achieved.

Before continuing, you should consider two sets of questions. First: What does reuse really mean in the context of SOA? Does reuse imply the reuse only of newly created Web services? Does it also include the reuse of existing applications? Does it include the reuse of existing programming and technology best practices, and/or software development guidelines?

Second: How does one achieve reuse? Many organizations that have embarked on the SOA path have realized that achieving true reuse is not as simple as creating Web services and making them available for use. Many different factors seem to wreak havoc on service use within an enterprise.

Understanding the nature of reuse within the context of SOA, and how one can generate the adoption of services, is an important step toward achieving one of the important promises of SOA.

What Is Reuse?

Reuse can take many different forms as described earlier, but my simple definition of the term is the ability of various people or service consumers to use the exact same service, component, procedure, guideline, or process repeatedly to fulfill a given task. From this definition it is clear that reusability extends beyond the mere reuse of Web services or existing applications (exposed as Web services). It also implies other focus areas and disciplines that are crucial to building a culture of collaboration—thereby providing the means to create a truly reusable approach based on the principles of SOA.

When engaged in planning and designing an SOA, establish a culture of reuse within your organization by focusing on these areas, which I call the "Services Adoption Framework" (see Figure 1):

- Best practices.
- Guidelines and policies.
- Architecture blueprints.
- Organizational structure.
- Communication.
- Services and metadata repository.

Reuse of services is often more complicated in practice than in theory. Many factors can stall reuse, such as lack of organizational support; lack of guidelines and best practices; interoperability; lack of standards; services discovery, and poor communication. Next, I'll address each of these issues in more detail.

The first factor that frequently stalls reuse is a lack of organizational support. It is important to have appropriate support from both developers and management when implementing an SOA. The biggest challenge is often a cultural challenge rather than a technical challenge.

Implementing an SOA implies change—sometimes a lot of change—and people deal with change in different ways. To some extent, most people are resistant to change, especially when they have not been included in a decision that directly impacts them, or when they do not agree with an approach or decision (even if they have been consulted in the decision-making process).

Firm support from all levels within the IT organization is necessary to implement an SOA successfully and reach appropriate levels of reuse. Management support is most crucial because managers drive the IT strategy, fund IT initiatives, and provide backing when important decisions have to be made.

IT architects, development managers, and developers play an important role as well. Without their solid architectural direction, some of the other issues discussed in this article have the potential to become major obstacles. Likewise, without the support and participation from the de-

Services Adoption Framework

Figure 1. When engaged in planning and designing an SOA, establish a culture of reuse within your organization by focusing on these areas.

Service Design Spec

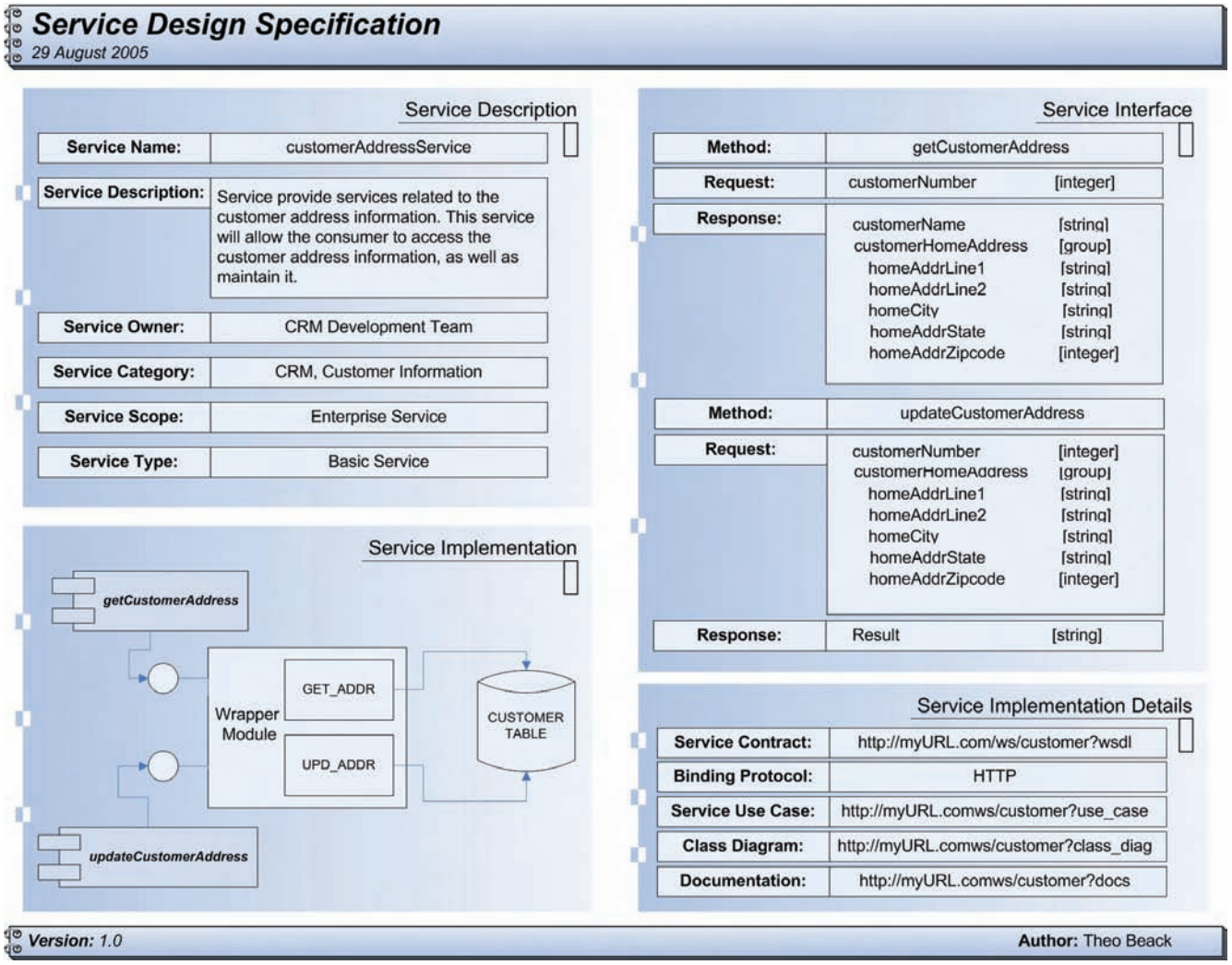


Figure 2. An example service design specification should address these aspects of the service.

velopment teams, an SOA initiative is doomed. When the development teams are committed to the strategy and follow the established guidelines and best practices, you can be assured that your organization will be much nearer to achieving reuse of the SOA approach.

The second hurdle to reuse is the lack of guidelines and best practices. Many SOA implementations flounder without focus or amid conflicting agendas. By first establishing pragmatic best practices and programming guidelines, you can create a clear SOA discipline. Most programmers learn to code by example, so the most effective way of helping programmers adopt Web services and SOA is by providing them with practical examples and guidance on how to create and consume Web services. Simultaneously, you must steer programmers away from the most common mistakes and bad programming techniques, which can lead to all kinds of inefficiencies and interoperability issues.

Interoperability, the third hurdle I'll address, is one of the main problem areas preventing consumers from easily interacting with service producers. Often a consumer can interact with a Web service that uses the Remote Procedure Call (RPC) binding style, yet it cannot interact with a similar Web service that exposes itself using a Document/Literal binding style. Other times a consumer can't deal with complex data types that are created by exposing a legacy application.

By establishing programming guidelines and best practices and following recognized architectural blueprints, developers can prevent these interoperability issues from occurring. Also, anticipating such issues allows architectural teams to put a framework in place that addresses these incompatibilities when they arise.

Fourth, a lack of standards can lead to competing approaches for creat-

THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

Come visit the Journal's new home at **www.ArchitectureJournal.net**. The new site contains a full library of articles from previous Journal issues in addition to upcoming highlights of our next issue. Browse the content today and post comments and letters directly to the editor!

Now live at www.ArchitectureJournal.net!

Microsoft®

ARC

ing, exposing, and consuming services. Web services are not mandatory when creating an SOA. Many developers prefer to use XML or XML RPC, or even a Representational State Transfer (REST) style of Web services.

Different standards can lead to many practical issues when implementing an SOA. It will serve an SOA team well to establish relevant standards and adhere to them. This can be enforced through the use of a services repository, which can perform a compliance test against the WS-I Basic Profile (for example) to ensure that all published services comply with relevant specifications.

A fifth reuse hurdle is services discovery. Many organizations have no effective means of determining which services exist and how to access them. This will remain a problem area until you have an approach to manage services and provide a central location to store metadata and artifacts related to the SOA services.

Finally, poor communication can hinder reuse. Does everyone within your organization know where to look for services? If not, how will they locate the services they need? How do you make changes to existing services, and even more importantly, who are your service consumers? If you don't know who your consumers are, how will you be able to notify them of the potential impact of changes?

How to Get Maximum Reuse

When implementing an SOA, you are faced with the question of where to begin. You might be tempted to start building Web services immediately, and many organizations take this route. This approach might work if the scope of your project is only to create a number of Web services. But when designing and implementing an SOA, diving right in is not the best approach.

The initial methods used to create, deploy, and consume Web services usually establish a pattern of behavior that might be difficult to change later. Establish practical guidelines and best practices first. Experience is the best teacher, and it is through trial and error that you can determine what works and what doesn't, or the best way of solving a particular problem. If you don't have the necessary resources or experience, it is worth the effort (and expense) to obtain help from someone with proven experience planning and building SOAs.

Next, define the architectural roadmap, or blueprints. Blueprints usually are extremely useful in guiding development teams because they address specific design areas within applications. Your blueprints might include any of these and more:

- Common security model.
- Service orchestration model.
- Metadata management.
- Process integration mode.
- Web services compliance model.

Too often, development teams have to reinvent the wheel to solve a particular problem. By solving problems once and documenting their solutions in architectural blueprints or best-practices guidelines, you achieve a certain level of reuse. Repeatability takes many different forms, and blueprints can have a positive effect on how services are created and used within the established SOA infrastructure.

Reuse Through a Services Design Approach

When creating services, it is important to start with a services design specification. Starting here allows you to identify which services will be needed, what their interfaces should look like, what the scope of each service will be, and the granularity of each service. Often, services are created without considering these aspects of the service. The extent to which you consider all aspects of a service has a direct impact on how useful it will be to potential service consumers.

By following a services design process, you can evaluate the requirements for different services and how they will be used. A thorough evaluation results in a balanced view of the different services for consumers' unique requirements. A good services design process yields a highly reusable set of services. An example service design specification should address the aspects of the service shown in Figure 2.

When taking the first steps toward an SOA, you usually start with the familiar. A large percentage of IT organizations have legacy systems that form the core of their mission-critical applications. This is usually the starting point when building an SOA.

Ask yourself how to reuse these systems or identify which parts to expose. First, determine which parts of a particular application will prove the most value when exposed as services. Next, ask what type of information is needed by other parts of the organization and what functionality would solve the most pressing request for information.

Then determine the different approaches you can follow to expose the identified functions as services. Generally, you can integrate with legacy systems in one of three ways: on the Session level, the Transaction level, or the Data level. By exposing these legacy applications as services through one of these approaches, you can reuse existing application functionality in new applications.

Session integration is the ability to intercept and interpret the screen information that is passed back and forth between a client and the server, such as z/OS, AS/400, Unix, and so

on. The terminal session (or screen information) can be packaged using different protocols, such as 3270, vt100, and 5250. These protocols describe the data related to the user interface and how it should be interpreted and rendered by the receiving application (terminal emulator). With session integration, it is possible to intercept the terminal emulation protocol data and render it in nontraditional ways; for example, the 3270 session data can be rendered as XML or as a Web service. This can be done without the requirement to reengineer or modify the legacy application in any way.

Transaction integration refers to a style of integration in which existing legacy transactions, such as BATCH programs or online CICS transactions, can be accessed from distributed platforms. External applications should be able to call these transactions without having to know any of the implementation details. These legacy transactions need to be wrapped in such a way that they are callable as Web services without disrupting the original state of the application.

Data integration is the ability to provide a standard level of connectivity (typically ODBC or JDBC) to disparate data sources. This functionality is important for legacy databases that do not support SQL or provide ODBC connectivity natively. The ability to expose these data sources as Web services enables data to be accessed in new ways and for new purposes.

Role of Services Repository

Many organizations start their SOA initiatives by creating ad hoc Web services. The existence of these Web services is usually known only by select developers, and the information related to these Web services is usually shared in an informal way, such as e-mail. This approach usually works with a small number of Web services and consumers. Managing the artifacts related to the Web services (WSDLs, XML Schemas, XSLT) is not a pressing need, and usually an informal agreement exists about how to deal with upcoming changes in the service contracts.

However, this approach breaks down when adoption expands to dozens or hundreds of services in the organization. Suddenly the organization is faced with new problems:

- Where do you go to determine which services currently exist within the organization?
- How do you determine whether the service contract that you have located is the most recent version?
- How do you determine who are the consumers of a particular service?
- How do you determine the potential impact of a change to a services contract?
- How are new services documented and where do you publish a newly created service?

These considerations have a direct impact on the reusability of the existing services. A services repository can play a prominent role in determining the answer to these questions and the success of SOA projects. Organizations have found that in order to properly manage services within their SOA infrastructure, they must have a central services repository where all the services can be published and documented.

The services repository provides standardized interfaces, such as UDDI, through which service producers can publish their services. Repositories also allow service producers to document their services by providing additional metadata that help consumers find appropriate services through different classification and search mechanisms. Consumers can be assured that whenever they bind to a service, they will do so with the latest service contract. Likewise, service producers now have the ability to track how their services are used and by whom.

Implementation of a service repository greatly increases the communication between the service producer and consumers, as well as the development teams. It is the central mechanism from which the various development teams can obtain the latest information regarding a necessary service. Most IDEs today provide UDDI support or plug-ins that allow developers to browse the services repository without having to leave their IDE environment. This ease of use inspires "integration" between the various parties involved in the SOA.

This article has only begun the discussion of reuse within SOAs. Many of you likely have unique insights and different perspectives that would turn this topic into a lively debate on the merits of reuse, how to cultivate it, and what it really means. Feel free to e-mail me your thoughts.

Finally, planning is a crucial part of success. I have encountered several situations where organizations were highly successful in their usage and adoption of Web services, but failed to prepare for success—to such an extent that their success created unique problems that hindered further growth and maturity within their SOA implementations. Above all, it is the proper planning, disciplined approach, and determined execution that lead to a successful and reusable SOA implementation. ■

GO ONLINE www.enterprise-architect.net

Read this article online:
Maximize Reuse of Services Within Your SOA

Read these related articles on FTPOnline:
Code Reuse in the Enterprise
Top 5 Asset Reuse Best Practices For SOAs

Scaling Over Time: The Version Problem

by Alex Krapf

See how to solve the problems surrounding management of a system's changes over time.

GO ONLINE www.enterprise-architect.net

Read this article online:
Scaling Over Time: The Version Problem
Read these related articles on FTPOnline:
Avoid Dead-End SOAs
Event Triggering XSLT Magic
Deal With Multiple Object Versions

Alexander Krapf *is president and cofounder of CodeMesh Inc. Krapf has more than 15 years of experience in software engineering, product development, and project management in the United States and Europe. Krapf has also worked for IBM, Thomson Financial Services, Hitachi, Veeder-Root, and Document Directions Inc.*

A lot of scientific papers and articles have been written about scalability. The focus has almost invariably been on the problem of scaling over resources or usage patterns, such as scaling over a number of processors or scaling over a number of requests. The problem of scaling over time, however, has largely been ignored.

By "scaling over time," I'm referring to managing a system's changes over time. Today, you usually scale over time by using a version control tool. You typically have the ability to revisit a snapshot of your entire codebase at a certain point in time. You can also look at a former version of any element in your codebase, normally at file granularity. Sophisticated version control systems might also allow you to deal with the elements of your system in terms of a change set, essentially combining changes to related elements into one conceptual change.

Version control systems are obviously an important part of your development infrastructure. Their features and capabilities have a large influence on your development process. Regardless of their feature set, they are all based on one unspoken premise: *Versioning is a concept external to your code.*

In today's development process, the idea of a version is introduced after you have written your code. The different versions of code that you use are essentially labeled snapshots in time. A later version of an element will not contain any information about the earlier version; all such change-related information exists only outside your code, as metadata in the version control system.

If you were to look at the evolution of a type *T* in a system, you might see something like Table 1. In version 1.0 of this product, I used the original version of type *T*. It remained unchanged until version 2.0, when I modified the type, and it became type *T'*. Then I immediately realized that I had to make another change, and it morphed into type *T"* with version 2.1. I didn't modify the type further throughout the remaining versions of the product.

Please note that in this example I have not made assumptions about the backward compatibility of any of the changes. The change from type *T* to *T'* might have been backward-compatible, while the change from type *T'* to *T"* might have been incompatible. The important thing to understand is this: Type *T* might undergo an evolution that is largely independent of the release numbers or version control labels. What's more, type *T* might not correspond with an element in the version control system. It might be a small part of a file, or on the other hand, it might span several files.

Release #	1.0	1.1	1.2	2.0	2.1	2.2	2.3	3.0	3.1
Type	T	T	T	T'	T"	T"	T"	T"	T"

Table 1. The evolution of type *T* over a number of releases

Traditionally, I have used the release number of a product to version an entire set of types, whether or not they had changed. I bundled a snapshot of the system into a deployment unit that might have a version number associated with it, such as a jar file or a shared library. I typically regard releases as black boxes and do not anticipate that implementation types of different product versions will coexist. Consequently, I often use the same, unchanged type name to represent different versions of a type. The assumption is they will never have to coexist in one context because they represent unrelated points in time.

Developers are increasingly running afoul of this core assumption. Yes, you might have discrete releases of software that never have to coexist, but type *T* might be a public API used by clients that have their own release schedule. It might be a persistent type, instances of which might be written out by version 1.0 of a product and (attempted to be) read by version 3.1. How does your version control system help you with these problems? The answer is: It does not help at all.

The Application Concept

Most IT professionals tend to equate "application" with the product's current version or maybe, if you're the unfortunate maintenance team member, the product's previous version. You do not usually consider way-back versions or not-yet-conceived future versions. You can typically get away with this misconception because customers suffer from the same misconception and run a single version of a product on their systems. They might not be aware of why they are doing this, but it has to do with the difficulties and costs associated with getting rid of the old version and deploying the new one.

Why is a product upgrade so hard? Why can't different versions of a product coexist peacefully? I would posit that it has to do with our broken notion of version management—a problem that's costing IT professionals and customers real money.

The difficulty stems from the fact that you don't treat the sum total of all versions of a product as the application. A new version will essentially be a different product that no longer includes the old version of the product. Any exposed interfaces that have changed could potentially break customers' applications; any changed implementation details that deal with persisted data could potentially break customers' applications. This is virtually no problem if you're dealing with a monolithic, self-contained application, but it becomes an increasingly bigger problem if you're dealing with published APIs, persistent data, third-party libraries, and so on. It is a huge problem when you're looking at service-oriented architectures (SOAs), which are essentially published application interfaces.

Consider applications that require you to store, maintain, and keep accessible some information for several decades. Such applications might, for example, be from problem domains such as pharmaceutical research, corporate governance (Sarbanes-Oxley), or intelligence. These problem domains will not allow you to punt on the issue of scalability over time.

What Can You Do?

It's not that developers and administrators are all incompetent; it's that the current tool set does not support the notion of change very well.

Let's look at a simple example. I'm using Java as an example because of its concise syntax, but this can easily be generalized to any other language or SOAs. I'll start with an interface *Foo* that publishes one method:

```
interface Foo
{
  public void  doSomething();
}
```

Over time, you realize that it would make sense to introduce an additional method in the interface:

```
interface Foo
{
  public void  doSomething();
  public void
doSomethingElse();
}
```

A lot of people would say that merely adding a method to an interface is a compatible change. Far from it! Imagine that you have many implementations of the interface. After your change, all implementations of the interface will have to be updated with the additional doSomethingElse() method. If all implementations are not updated, your application will not load anymore. Additionally, some of the existing implementations might not need the new method, but the compiler will force you to add the method implementation anyway.

You might counter that this was the wrong way to version your interface. An interface should never be modified once it has been created; instead, it should be extended through inheritance. So you might propose this design instead:

```
interface Foo2 extends Foo
{
  public void
```

```
doSomethingElse();
}
```

You certainly solved one problem: You don't have to modify any existing implementations of the interface. Implementations of *Foo* and *Foo2* can coexist peacefully. But what about existing users of the *Foo* interface? You might have a factory method that originally created an instance of a type implementing *Foo*:

```
public Foo createFoo( String
arg1, int arg2 );
```

Should this function return *Foo* instances or *Foo2* instances in the future, or should it return *Foo2* instances through a return type of *Foo*? Should you introduce another factory method, such as this one?

```
public Foo2 createFoo2( String
arg1, int arg2 );
```

Will methods that used to take a *Foo* as an argument now require a *Foo2*, or not? If a *Foo2* is required, should they enforce this through their parameter declaration?

Also, it's silly that you have to use the inheritance mechanism to express a new version of the same concept. It has always bothered me that I have to give up the "perfect" name for a type, simply to version it. But beyond personal preferences, this issue also causes real and expensive problems in current applications. So far, I've just been talking about types in general. It becomes much worse when you consider the domain of application integration. The whole point of integration is the publishing and consuming of interfaces and data that is generated by another entity. How many versions of applications, APIs, and data objects do you think you'll encounter over 20 or 100 years?

The State of the Art

The state of the art is … not very artistic. Mostly, there are programming guidelines, naming policies, extensibility patterns, and best practices, but there is little or no support in the technologies in use today. Some programming languages include the concept of an API version; some architectures include the concept of a service version. Java has a serial version UID that can provide an indication of the compatibility of data or API types, but nothing on the market today offers:

- Versioning as a first-class language feature.
- Tool support for type versioning.
- Tool support for detecting and dealing with version problems.

This example is not a proposal for a Java language extension; I simply intended to illustrate what versioning support as a language feature might look like and what it has to offer:

```
versioned interface Foo:2
{
  public voiddoSomething():1-2;
  public void
doSomethingElse():2;
}
```

This interface declaration essentially tells us that interface *Foo* is a versioned interface and that it exists in two versions (*Foo:1* and *Foo:2*). It also tells us that the first method is present in both versions, whereas the second method is only present in version *Foo:2*. This method is already much nicer than having to use two different types because you keep the declarations together and have an immediate understanding of how the type evolved over time.

Now you can create implementations of both versions of the interface, for example:

```
public class FooImpl
implements Foo:1
{
  public void  doSomething()
{
Util.doSomething();
}
}

public class FooImpl2 extends
FooImpl implements Foo:2
{
  public void  doSomethingElse()
{
Util.doSomethingElse();
}
}
```

Notice that you probably don't want to version everything in the system. Here I chose to version the interface but not the implementing classes. So far, this approach is not much different from just using a naming policy, but imagine that you could overload methods based on the versions of their parameters. You don't have to be explicit in terms of version, but you can choose to be. You could do this to indicate that you will support any version in the implementation of the method:

```
public class FooUser
{
  public int calculate( Foo f )
  {
    versionswitch( f )
    {
      case 1:
        return 1;
      case 2:
        return 2;
      default:
        return 2;
    }
  }
}
```

Or you could overload the method based on the version of the argument:

```
public class FooUser
{
  public int calculate( Foo:1 f )
  {
    return 1;
  }

  public int calculate( Foo:2 f )
  {
    throw new
VersionNotSupportedException(
f );
  }
}
```

So what's the big deal here? Imagine that the compiler might perform these steps:

- Enforce that every declared version of a type is handled either through an overloaded method or through a version switch.
- Enforce that you either declare version incompatibility or provide a compatibility path between versioned APIs.
- Enforce that there are conversion operators between different versions of a versioned serializable type.
- Inform you of all the changes you need to make to your code if you were to version a certain type. All you would need to do is version the type and try to recompile.

Wouldn't it be nice to write this code, try to compile it, and receive a compile-time error because you are not handling the later version of the interface?

```
public class FooUser
{
  public int calculate( Foo:1 f
)
  {
    return 1;
  }
}
```

```
  Error: FooUser.java, line 1:
method calculate(Foo) does not
support Foo:2
```

Service-Oriented Architectures and Versioning

Please don't be distracted by the futuristic and unlikely extension to the Java language used in my examples. The same problems exist to an even greater extent in SOAs. In a service-oriented architecture, you will publish only interfaces, and you will use only serializable data types. Both categories of items will force you to deal with the versioning problem sooner or later.

It is a sad fact that versioning support is as poor in current SOAs as it is in programming languages, even though you need it even more. In a traditional programming language, you're creating an implementation that is usually tightly coupled internally. You control the types that are used, the versions of the third-party libraries that you bundle, the packaging, the deployment, and so on. In a true SOA, on the other hand, you are going to build applications that are stitched together from services that you might not have developed and that are not under your development or deployment control.

Imagine another group creating a new version of a service that your application is consuming. The new version might offer great new functionality, but it might also be totally or subtly incompatible with your application. The other group might be a good corporate citizen and simply add a new version of the service, but this raises other troublesome questions:

- The other group might prefer to have the new version have the "perfect" name. Now you will have to change all your applications.
- How long do they have to keep old service versions around? How can you know for sure that no one needs the old version anymore?
- How do you inform service consumers of newer versions?
- Can you afford to keep dozens of different service versions (each potentially backed by serious infrastructure) around forever?
- How can service consumers tell which versions are "compatible"?
- What does service version compatibility even mean?
- Where are the tools that enforce consistency in orchestrated service frameworks?
- How many different versions of serializable data types named "PatientData" do you plan to support? Concurrently? In one application?

In the past, few applications had to have a lifecycle that spanned decades. Tools are equipped to scale over a few years, maybe up to a decade, but eventually applications accumulate so much "cruftieness" through namespace pollution, unenforceable naming and versioning policies, and so on that a rewrite and the loss of compatibility are taken for granted.

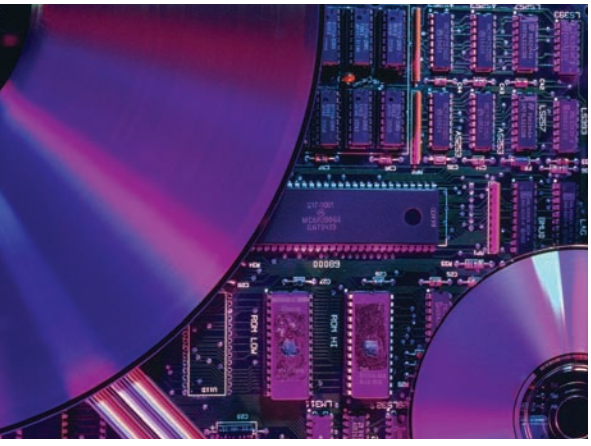
In the future, this is not going to be an option. The government has already proposed or enacted legislation that forces corporations to keep data available and usable forever. You will be in desperate need of formal versioning support in the tools you're using. The current best practices for writing maintainable software will not be good enough, simply because they don't scale over time.

Our technology vendors will have to step up and add versioning support to languages and technology specifications, otherwise developers are doomed to fail before they even get started with the second version of their product. ■

Combat Increasing IT Complexity

by Firdaus Bhathena

Achieve success in your architecture initiatives through the appropriate mix of people, process, and technology.



GO ONLINE www.enterprise-architect.net

Read this article online:
Combat Increasing IT Complexity

Read these related articles on FTPOnline:
Strategies for Operational Risk Management
Component Asset Management
Harness Technology Architecture

Firdaus Bhathena *cofounded Relicore in November of 2000 and served as the company's president and CEO for its first four years. In his current role, he is responsible for defining and driving Relicore's market, product, and technology strategies. Prior to Relicore, Firdaus cofounded WebLine Communications, a company responsible for introducing Web-based, enterprise-class customer interaction software to the international call center market.*

Over the last decade, the role of an enterprise architect (EA) has evolved from a focus on consolidation of technologies, to application and data integration, and most recently, to business alignment and support of business strategy. The latest iteration of the EA role calls for an increased focus on architecture planning to ensure that IT is in a position to effectively respond to ever-changing business needs.

As most of us have learned, the best architecture planning efforts are tied to the baseline of what you have in your environment today. Further, the foundation of any successful re-architecture initiative is a fully accurate picture of the systems, applications, and other infrastructure components in your environment, and how they collectively function to deliver IT services to the enterprise. When these enterprise linkages and interactions are documented and widely understood, the EA performs a powerful role in business enablement.

This sounds relatively straightforward, but several factors can make it difficult to get an accurate view of this information when it's needed and to leverage it to support important business decisions.

The first factor is increased complexity. As IT organizations continue to move from monolithic, legacy applications toward distributed business applications, strategic and operational teams alike are struggling with infrastructure that is increasing in scope, scale, and complexity (see Figure 1). Nothing runs on "a box" anymore; distributed application infrastructures are characterized by an integrated and customized collection of many smaller software components, with dependencies on common "building blocks" such as databases, Web servers, and application servers. Complexity is further compounded by mergers and acquisitions and decades of layered technology purchases.

IT complexity has a major impact on availability, manageability, and operations costs. Complex IT environments are inherently expensive to operate, more difficult to manage, and can be unpredictable when change is introduced. However, complexity cannot be eliminated in a dynamic and innovative environment, so the goal needs to be to manage it through making appropriate investments in architecture, internal culture, organization, and technology.

The good news is that investment in enterprise architecture has emerged as a core strategy for leading organizations seeking to rein in complexity within their IT environments and create a common language across management disciplines.

IT Complexity and EA Success

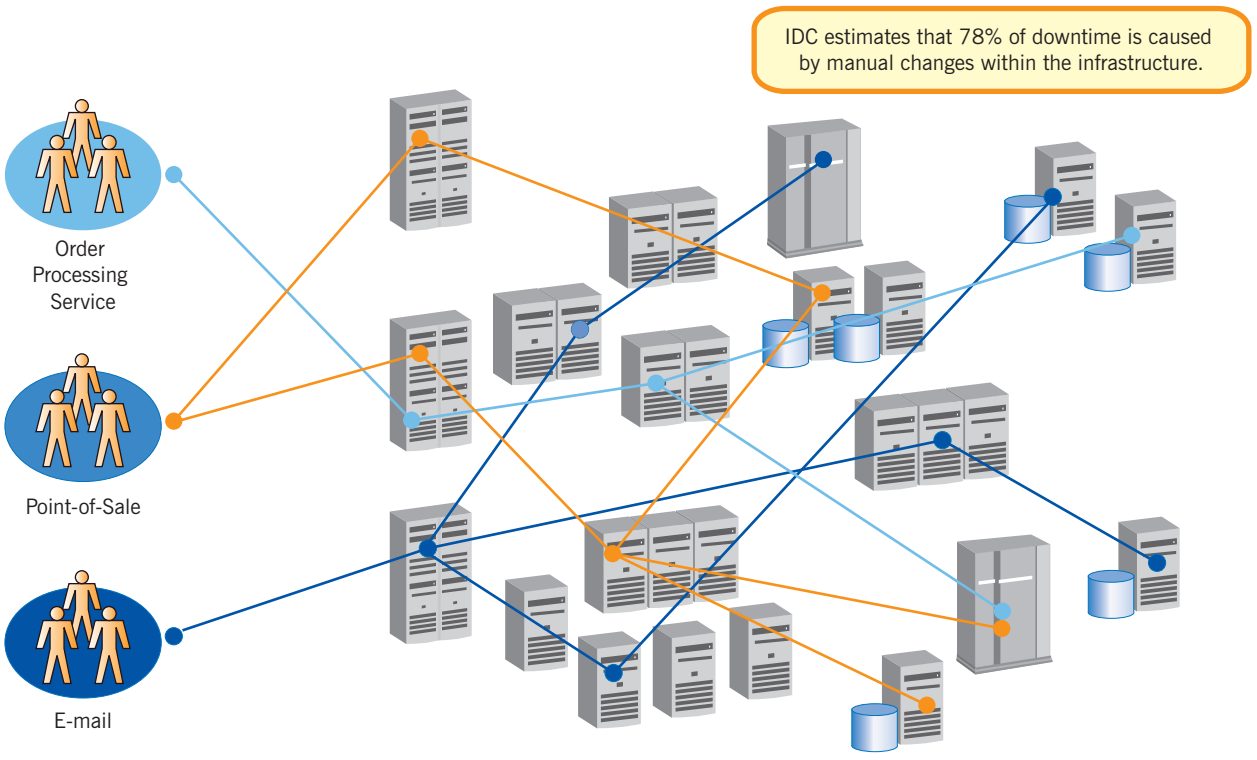


Figure 1. Increasing IT complexity can hinder EA success.

Beware the Rate of Change

The second factor that can make it difficult to get an accurate view of your environment is the rate of change. Operational changes to infrastructure—new application rollouts and upgrades, configuration changes, hardware changes, hot fixes, security patches, and so on—occur continuously. Larger change projects such as data center migrations, server consolidations, and infrastructure assimilation of acquired companies are also routine activities for leading companies today. Change is a well-known complexity multiplier, and as you're all aware, the pace is on the increase, with no signs of decrease in sight. Without an understanding of what's in the environment and how elements are interdependent upon one another, any change to the architecture is fraught with risk.

Scattered information is a third factor complicating a clear view of your environment. IT architecture and management information tends to be scattered throughout a company and to reside on whiteboards, in notebooks, and in the heads of enterprise architects, system and network administrators, and other critical IT personnel. On the positive front, EA tools are emerging to combat this challenge by storing relevant information in a repository and providing capabilities to assemble and present the data in a variety of ways.

A fourth and final factor to account for: technology advancements such as virtualization and provisioning. As every aspect of enterprise architecture and computing has grown more complex, the flexibility and intelligence that virtualization and provisioning add to the management mix has made these technologies increasingly attractive. Virtualization reduces technology limitations and provisioning reduces capacity constraints. Both virtualization and provisioning increase the rate of change, which contributes to increased complexity.

Given these challenges, how can you get a fully accurate picture of the systems, applications, and other infrastructure components in the environment? You also need to know how they support the delivery of IT services to the enterprise—knowledge that is central to nearly every EA initiative. Here are some suggestions.

How to Get the Big Picture

First, foster and promote collaboration. The EA's goal is to select and

Enabling a CMDB

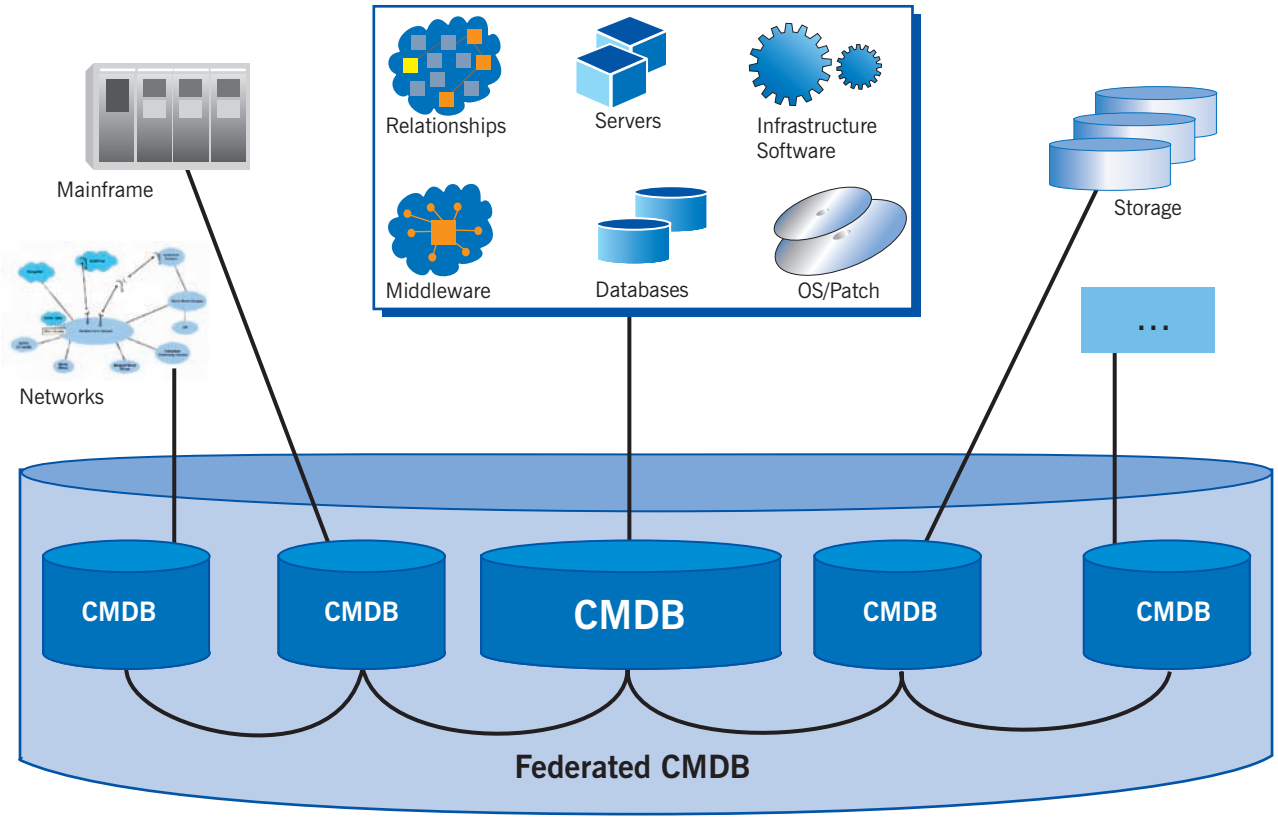


Figure 2. Learn how you can enable a CMDB.

implement the right investments in standards, procedures, and technologies to support the organization's business goals. This requires teamwork and collaboration with all groups within IT as well as key business personnel to ensure that the EA has a clear understanding of business needs, and that the business respects the role of the EA. Armed with this knowledge, for example, the EA can investigate the appropriate technologies to support business needs, gain buy-in on technology standardization, and work with the appropriate groups to resolve or rationalize exceptions to standards and maintain insight into the big picture of their architecture environment.

Second, implement structure through framework and process adoption. Complexity can be reduced (although not eliminated) with structure. A structured environment is built around standards and conventions for all components of the infrastructure, and many frameworks and process methodologies exist for providing structure. While there are no common definitions, standards, processes, or tools for managing enterprise architecture, EA frameworks that can provide structure include the Zachman Framework, the Open Group Architecture Framework, and the US Federal Enterprise Architecture Framework.

Widely used process methodologies include Control Objectives for Information and Related Technology (COBIT), which provides a reference framework for management, users, and IT audit, control, and security practitioners and covers all IT activities; Six Sigma, which is a broadly applied, disciplined, data-driven approach and methodology for eliminating defects; and IT Infrastructure Library (ITIL), which is a widely accepted and cohesive set of IT best practices focused on service management that continues to gain traction within companies seeking to improve their change management efforts.

Third, leverage technologies that give you the big-picture view of your environment. To manage the complexity of IT, organizations must begin by understanding their architecture and the interrelationships and dependencies that exist within the environment. In the past, many of us have attempted to create a map, or blueprint, of elements in our IT infrastructure by maintaining multiple rudimentary data stores such as spreadsheets, Visio diagrams, or Microsoft Access databases containing data gathered through manual efforts. This is an ef-

fort begging for automation—a manual approach simply cannot provide sufficient information due to the size, complexity, and amount of changes occurring within IT, or deliver fully accurate information at any given point in time.

One company reported that manually mapping out just a single critical business application took five staff members several weeks, and due to the dynamic nature of their environment, the data was out-of-date before the project was completed. This was a source of great frustration because they needed this information to support a host of strategic initiatives and operational tasks.

You can address this challenge with technologies such as automated application and server-dependency mapping solutions that provide information about hierarchical and peer-to-peer relationships existing among infrastructure components. The best tools in this category completely eliminate the manual effort traditionally associated with this process by automatically discovering infrastructure components, dynamically mapping their dependencies, and tracking changes in real time as they occur. The result is an automatically generated, dynamically updated picture of the complex server and applications within the IT infrastructure.

First and foremost, promote collaboration.

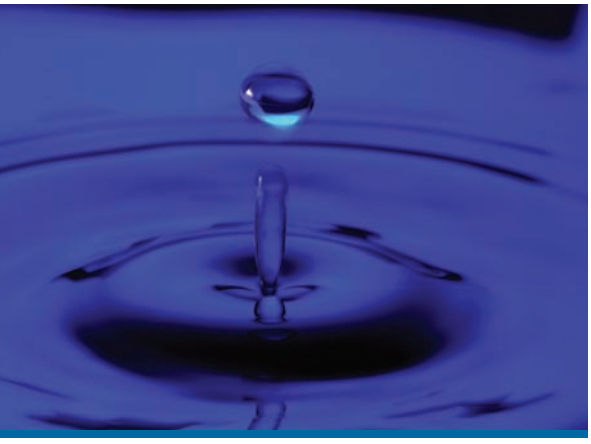
The information from these tools provides the foundation for strategic initiatives such as audit and compliance, disaster recovery, business continuity, and data center migrations, plus operational activities such as problem resolution and change impact analysis that require realtime information. Additionally, these tools can provide a critical feed of realtime, fully accurate application and server information into an enterprise configuration management database (CMDB) and maintain synchronization between live configurations and records stored in the CMDB (see Figure 2), or serve as a feeder into other EA tools used to create blueprints of business, systems, and technical architecture.

At the end of the day, an accurate understanding of your systems, applications, and other infrastructure components and how they function collectively to deliver enterprise services is essential to the EA's ability to realize business benefits such as cost reduction and technology standardization, process improvement, and strategic differentiation. ■

Incremental Architecture:
Principles for the Real World

by Chad Riland and Josh Paterson

Learn how you can establish meaningful enterprise architecture, cut costs, and deliver solutions more rapidly.



GO ONLINE www.enterprise-architect.net

Read this article online:
Incremental Architecture: Principles for the Real World

Read these related articles:
Lightweight Enterprise Architectures
Changing the Rules of Systems Architecture
Bridge the CIO/CEO Communication Gap
The Mobile Enterprise Challenge

Chad Riland *is the Manager of Architecture and Design and* ***Josh Paterson*** *is an Application Architect for Calpine Corporation. Calpine Corporation was recently ranked by* Information Week *as the most innovative users of information technology among energy and utility companies and fifth across all categories of companies. You can e-mail Chad at chadriland@gmail.com and Josh at Josh_Paterson@hotmail.com.*

How can enterprise architect teams successfully establish meaningful enterprise architecture while meeting management demands to cut costs and deliver solutions more rapidly?

As solution architects, we are continuously faced with the business drivers to cut costs, deliver solutions more rapidly, and respond to customer needs with more flexibility. In light of this fact, we must develop a practical, focused strategy that will allow us to meet goals toward furthering enterprise architecture (EA) initiatives (i.e., business alignment, standardization, reuse of existing IT assets, and sharing common system development methods), as well as meeting the enduring demands of business units and upper management.

The ultimate goal is to design more agile and responsive enterprise systems that provide the value our business partners demand, in addition to the integration flexibility for which architects and developers have been striving. To reach this goal, we must avoid two common pitfalls of EA endeavors: not showing clear business value and being perceived by the development organization as overhead or bureaucracy.

We can remedy those concerns and achieve this vision by applying standards and practices incrementally over time. Show value project by project and expand the EA influence to one solution, one service, and one person at a time. In a rapidly changing business environment of tactical decision making, it is often easier to justify small EA efforts that show immediate value than to get buy-in and funding for large-scale EA projects that many perceive as the "ivory tower." Project-based initiatives can be used as the starting point for any company's long-term EA vision, and form the basis for reusable standards and best practices.

Why Enterprise Architectures Fail

Many attempted EAs fail because they wait to resolve too many details before setting a specific direction. They try to take the Big Bang approach, or assume that there will be less risk and better long-term results if all of the details are gathered up front. However, this doesn't work in today's tactically driven enterprise, where the time and resources needed for an all-encompassing approach are difficult to obtain. In many cases, an immediate decision is better. Otherwise, there may be no decision at all.

In fact, an architect must frequently present a solution prior to knowing all of the details of the problem. This formula is often diffi-

cult for technical or analytical thinkers to follow. The career path of an architect is typically one where he or she has moved up through the technical ranks—first developing some basic software systems; graduating to more complex systems where he or she had to understand and apply design methodologies; and after showing successful results, moving into an architecture position. This progression can perpetuate the view that EA is all about technology (i.e., technical expertise, standards, and picking the best technologies). If this attitude is left unchecked, it can result in disputes within different factions of the organization over specific technologies.

Another issue may reside in individuals within the organization who come from a strictly technology background because individuals with this type of background tend to lack leadership and management skills. These individuals may attempt to control others with their rank and authority; or, on the other hand, they may not attempt to manage at all and simply continue to focus on technology. They may attempt to enforce rules without getting agreement from the development team and business sponsors. In these cases, others will hesitate to include architects on project teams.

EA failure also has to do with how the organization perceives the architecture team. After half a decade in the '90s of the double-digit IT budget expansion, corporate IT growth has dramatically slowed down. IT is being asked to deliver the same amount of new capabilities with decreasing budgets. If an EA team (and IT in general) is not conscientious about the influence of budget constraints on their environment, and not developing new ways to thrive in that environment, a perception will evolve in the business units that EA teams may not meet needs.

In addition, an EA team cannot expect to obtain funding for strictly architectural projects like models, frameworks, or methodology. Nor can they wait for upper management support and exposure for those activities. It is very difficult to quantify value for these types of initiatives. They often meet with resistance from executives that have to sign off on the budget because many times these executives must justify the projects with positive customer response.

Next on the list is overzealous evangelism. You've all heard the evangelists touting lofty objectives, abstract benefits, and future value without a clear plan for progressing from where you are right now to the utopia of an established EA. Advocating "the corporate good" is a great way to get people jazzed about strategy, but without a clearly defined execution plan, it amounts to nothing. The other side of this coin is the "build it and they will come" mentality. It may work for a successful sports franchise to build a new stadium and attract a myriad of fans, but it will not work for the establishment of an EA.

Architecture teams will not gain allies by simply building models and providing pages of documentation to other teams. The consumers of this information have project timelines to meet, code to write, business meetings to attend, and so on. Without direct participation and buy-in from these "consuming" teams, the response is frequently apathy: "Sure, just put it on the pile, and I'll read it when I get to it."

The Incremental (Results-Oriented) Approach

Incremental architecture is focused on results and value. The value of a common architecture is realized in the tactical application of strategic technologies. It is evolutionary progress instead of revolutionary progress. Architects need to implement architectural strategy through existing tactical projects. They should devote a substantial amount of their time to project teams that are well aligned with the business. This enables the expansion of architectural progress one solution, one service, and one person at a time.

The keys to success can be broken down into three elements: leadership, communication, and standardization. Incremental architecture is a concept that is valuable beyond corporate Information Services (IS) shops. For example, this article was created and finalized based on the principles of incremental architecture. It started with a vision and continued to evolve that vision internally at Calpine and externally at publication venues. It then progressed into a framework for the article's content, approach, and message standardization for relevance. Ultimately, the original vision was accomplished with tactical progress over time.

Leadership

Good architecture is a people challenge, not a technology challenge. Frameworks, methodology, and modeling are not productive in and of themselves—architecture must be action-oriented. The focus must be on creating value, changing behavior, processes, and executing the larger strategy.

People want to have a clear picture and roadmap—a plan that has a defined path of execution. Establishing a clearly defined and reproducible System Development Life Cycle (SDLC) process is a key step in the roadmap. This process provides our architecture and design team with important collaboration touch points among the majority of IS initiatives in the portfolio. The SDLC gives the team an opportunity to review and revise an individual project

plan, ensuring that it satisfies long-term and strategic goals. The SDLC also involves the team in the day-to-day operational activities of the business—no more ivory tower!

Over 50 percent of architects' time should be allocated to project-based work. This will allow architects to share the overall corporate vision, interact with multiple team members (business analysts, developers, and so on), and prove their worth on tactical efforts. Many strategic objectives as well as various EA initiatives (modeling, framework development, pattern documentation, and so on) can be achieved incrementally by showing that they apply to a particular project.

For example, architecture team involvement in projects with various reporting needs across numerous business units call for a near-term, common approach to reporting. On the architectural vision side, enterprise reporting and other business intelligence (BI) strategies align with strategic business goals. The tactical project requirements are a perfect match. They drive momentum for increasing reporting and BI maturity for competitive advantage, while also meeting individual project goals.

Communication

The ability to build relationships and establish alliances across the organization should not be understated. Great care and effort must be put forth to reach individuals outside the typical technological arena of an architecture team. To be successful at increasing EA awareness, you have to find creative ways to streamline the communication process.

There are a few main factors for successful communication in the architectural arena that go beyond the usual communication theory course. Remembering these principles will help you make your point:

- Relevance.
- Practicality.
- Addressing key stakeholders.
- Delivering the right message.
- Collaboration.
- Trial and error.

Information overload is commonplace within today's workforce. Two creative ways that Calpine has had success in communicating vision and strategies are through the communication mediums we call "whiteboards" and "roundtables." Our whiteboards are short video-based presentations that explain specific areas of technology, standards, projects, or vision. Podcasting is becoming increasingly popular, and whiteboard presentations are a spin on the podcast concept.

It is much easier for someone to digest information from a video presentation than it is to pore over pages of documentation. It is also more efficient for the architect to communicate thoughts in this medium than to write them down on paper. You can take the whiteboard concept a step further if you require certain individuals to watch the video by incorporating it into a learning management system's courseware. After watching the video, these individuals are more likely to be successful at their jobs.

Also, you should never be above marketing your team's agenda and goals. A creative example of this is placing a one-minute, team marketing section at the beginning of communication videos, similar to the way online news Web sites create their videos. Remember, in a whiteboard communication strategy, communication is one-way, not collaborative. Allow for necessary collaboration beforehand on any initiative described or defined in a whiteboard, and describe the parties involved in helping the work progress.

Roundtables are gatherings of key stakeholders in a particular technology or project. They can be either technology-focused with a goal of extracting project ideas or project-focused with a goal of extracting technology implementation ideas. Numerous positive things come out of roundtables. Innovation is sparked by encouraging moderated communication about a topic between individuals who don't normally talk to each other. Architectural strategies, new project ideas, system improvements, and especially cross-team communication are other byproducts.

These events don't take much time, and the return on investment for time spent is enormous. These are true collaborative gatherings. Individual idea exchange and contribution should be encouraged and be part of the rules that are established up front. Collaborative decisions do not make the IS community feel that they are receiving directions from above. Instead, they engage creative IT talent in developing the resulting strategies.

An architect has an exceptionally challenging job. He or she must align with the business and understand objectives to have a clear grounding in the conceptual underpinnings of many different technologies. Fundamentally, an architect must be a leader and communicator who is capable of bridging the gap between technologies, customers, and business goals by turning strategy into tactical reality.

The incremental architecture approach adds additional value to individual projects because they now are aligned with strategic technical goals instead of purely tactical goals for the business. Leadership and communication are marked as cornerstone components for the success of the incremental architecture approach. ■

Put some Insight into your Software Architecture

Software Architecture insight

As a software architect, both business needs and technology demands affect your decisions. You have to make strategic architecture decisions based on what's achievable today—with an eye to future growth and change.

That's where FTP Online's *Software Architecture Insight* helps you. Twice a month, this must-have e-mail newsletter gives you both technical perspective and actionable advice for building applications and enterprise solutions. You'll learn about important topics like:

- real-world SOA
- proven middle-tier strategies
- best practices
- modeling business processes
- architecting for scalability
- migration strategies
- much more!

Free newsletter:
Sign up today!

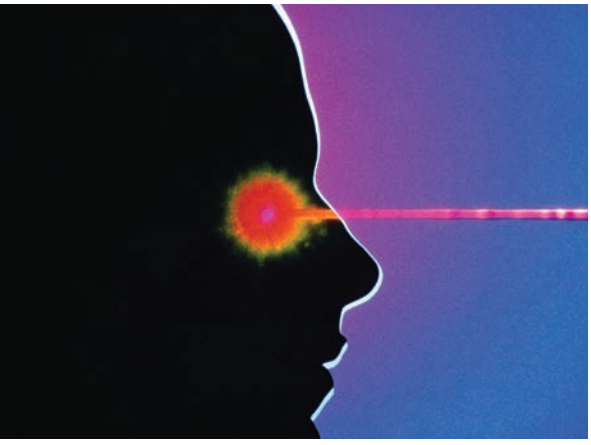
www.ftponline.com/channels/arch/

FTP FAWCETTE TECHNICAL PUBLICATIONS

If SOA Looks Hard, You're Looking at it Wrong

by John Sadd

Deconstruct the way you think about a service in service-oriented architecture.



GO ONLINE www.enterprise-architect.net

Read this article online:

If SOA Looks Hard, You're Looking at it Wrong

Read these related articles on FTPOnline:

Stories From the Field: Adoption of SOA and Web Services

How SOA Can Benefit From Active EA Models

Get Acquainted With SOA and Indigo

John Sadd *has been with Progress Software for 18 years, working in many capacities in product development and the company's Applied Technology group. Now an Engineering Fellow and Evangelist for the Progress OpenEdge product, he has written a half dozen books on OpenEdge products and given many presentations around the world.*

The playwright Moliere's creation Monsieur Jourdan was astonished one day to come to the realization that he had been speaking prose all his life without realizing it. Similarly, people in the software industry often attach weighty meanings to what should be simple words and concepts. In the process, the simplicity is lost and the words intimidate you rather than support you.

Today IT professionals are laboring under what seems to be a requirement to express all business applications using a service-oriented architecture (SOA). When you hear the word "service," all the simple meanings seem to disappear (such as, something that *serves* you), and only the weight and burden remain. To say "service" somehow automatically means using Web services for the communication between all the parts of your applications, let alone with all the other applications that it must talk to in order to survive and prosper.

This assumption brings with it a list of groan-inducing three- and four-letter acronyms—SOAP, WSDL, SSL, SAML—along with a host of burgeoning standards that have to be lumped together as WS-*—read "WS splat," as you're splattered by the implications of juggling Security, Routing, Reliability, Eventing, Addressing, and more without dropping anything on the floor.

Services were supposed to make it easier for you to build successful applications that handle business needs. But do you even *remember* the business needs after you're done struggling with all the technology requirements?

I'd like to suggest that, as with Moliere's ironic observation that a highfalutin word can really be applied to commonplace things, you should think about the services of a service-oriented architecture as, well, *serving* you in your business needs, not making your life more complicated. And, like all the mundane everyday statements that can be given a greater cachet by labeling them as *prose*, the right expression of the solutions to your business problems can turn into services and even fit into a true SOA if you think about the simplicity of the business task at hand and let the technology serve you without getting in the way. Here are some examples of what I mean.

The first step that's been recommended for years—decades, maybe, at this point—is to turn the spaghetti code of a typical older application into something more flexible and forward-looking. You can do that by separating your user interface from your business logic. Take

the logic that captures your business rules and remove it from the code that puts up the old green-screen interface or somebody's proprietary client/server GUI. Now you're on the path to keeping the value of the business rules intact when the UI fashions change. What's the alternative to nicely separated duties between the logic and the UI? Well, as people kept saying with a quiet smile at an architecture conference I recently attended, "Thank the Lord for screen-scraping tools."

With a proper separation of roles, you won't have to scrap your core business value and code your application again during the next industry "revolution." You'll be set up to access your data and domain logic from a variety of interfaces. You'll even be able to let parts of your application that don't have a UI retrieve data by using the same API calls your nicely separated code now uses. You'll also be on the way to letting other applications approach your data by publishing the API for them, which will make your code look a lot like—gasp, could it be?—a *service*.

Here's another example. What's one of the most common elements of a bloated application, one that has grown and grown over the years without much coding discipline? What do harried programmers do whenever they need a function that resembles or duplicates something already in the application?

The Dangers of Copy-and-Paste

They copy and paste somebody else's code—or maybe their own code—into a new procedure and make the necessary changes to adhere to the spec. Of course, they could factor out all the code parts that apply to both cases, or a growing number of cases, taking care not to break the code they started with. Then they could parameterize the variations. But most developers don't have time to do that.

You know where this scenario leads. Many applications have so much repeated and copied code that the similarity among resulting procedures is the closest thing these applications have to an architecture. A lead developer on a large ERP application once admitted to me that she had identified 157 different places in an application where a price calculation is performed.

Think about the services of an SOA as serving you in your business needs, not making your life more complicated.

What's the well-architected solution to this kind of problem? You must discourage the copying of code into a new procedure because it's the handiest solution. Instead, you need an architecture that gathers in one place all the logic that supports a related set of data. So whenever programmers need to make a change or an addition to the logic, they can go to one place to do it. Also, any other part of the application that needs to use or update the data knows where to get it. And if you provide a sufficient set of calls into that one procedure, then it can satisfy all requests in a consistent way.

When you develop in this way, the reduction in the amount of your application code can be staggering. Even *other* applications that will someday need to access data from your application can use a similar interface to retrieve it. And then your application is perilously close to becoming… well, you're getting the picture now.

Single-Machine Days Long Gone

Make sure all code that belongs together under all circumstances runs as a unit, and that all other code that might want to talk to it runs in some independently-defined unit, without any dependencies between the two units. If you don't, you'll be sorry when you discover that for a configuration you never considered, the two code units have to run on two different machines.

Separating out the UI is only the beginning. If you make sure every business object that manages a rational chunk of data can run independently of every other business object, you're on the right track. And when someone tells you that one of those chunks contains information necessary to somebody on some other corner of the planet, you can respond with a confident smile, and not a shiver of dread.

Here's another example: One of the big hot buttons in application development is "workflow." Developers, especially those designing and building applications for a larger audience, realize that requirements differ and change depending on how all the steps in a business process need to take place. Just when you thought you were done coding, you are informed that Susie—and Susie has clout—insists on entering the stock code before the customer ID, rather than the other way around, as it was originally spec'd.

You might panic at first. Entering the stock code requires running a whole business object that validates the stock code for the country of origin, assesses its availability, and so on. Entering the customer ID

requires accessing another similarly complex object. But wait! You coded each of those business objects so that you didn't need to concern yourself with where they run, if they're both in the same place, or whether the caller is a UI or a batch program. So it turns out you can easily accommodate Susie's wishes.

In a more serious scenario, your highly complex multistep ordering and pricing algorithm—remember the 157 price calculations you replaced with one—now has to be applied in a country you didn't consider during development, but one where your company must now do business to survive. It won't work to tell your prospective new customers, "Well, that's not how we do things in our country." All your pricing work needs an overhaul so that step five can execute in front of step four and step seven can be replaced by a new step that reflects the tax requirements of the new customers.

But because you coded the business object that manages each step independently, you won't have data dependencies or ordering dependencies between objects. With your nicely behaved code, you can reverse step four and step five and replace step seven without the rest of your application being the wiser, or having anything to complain about. It's the most natural thing in the world for you to express all your business objects as, well, you know… services.

Armed with such civilized code, SOAP and WSDL can hold no terrors for you. You want the output as XML? Nothing simpler. It just becomes another call in a well-defined API. Your boss wants to be able to resell the pricing algorithm without the rest of the ordering code? No problem. They'll never miss each other.

Jason Bloomberg at ZapThink reminds us that "you can't get SOA from software, because SOA consists of best practices." Bingo. You have to create an architecture that sets you up for success when the world changes, regardless of what technologies they throw at you. New technologies should just be new wrinkles in an already familiar and comfortable suit, no more dis-

You have to create an architecture that sets you up for success when the world changes.

tressing than Susie's special data-entry requirements. You can't let technology dictate how you think about the basic job you're working to solve—solving your company's business problems and letting yourself and your company succeed in a changing world.

Does this mean that SOA is simple? No, not automatically. After all, there's been enough sloppy code written to demonstrate that writing poorly architected applications is easier than writing good ones. But with the right motivations in place, architects and developers should have what is needed to build applications with a solid architecture, which makes everybody's job easier in the long run. Anyone who's had to code that 158th pricing algorithm variation will know exactly what I mean.

So a good "architecture" becomes its own reward. And as for the "service-oriented" part of it? Well, everybody thinks of their business logic as a set of services, don't they? It just makes life so much easier. ■

Down With Downtime

by Dan McCall

By implementing automated business application processing, you can realize significant increases in efficiency and productivity.

From mergers and acquisitions to new compliance mandates to service-oriented initiatives, the evolution of business is adding a new layer of complexity to the modern work environment, but it also promises rewards of increased efficiency and simplicity. Business application processing has become the new factory floor, and innovators within IT departments are poised to become the Henry Fords of today's enterprise environment.

Maintaining competitiveness requires the agility to respond immediately to change. End users' expectations of business processes are approaching that of the telephone, with users demanding always-on, quality service. Too often, however, IT departments are stuck reacting to existing business demands instead of proactively pioneering solutions for future needs. There are many excuses for reactive IT departments, but business application processing no longer needs to be one of them.

By implementing automated business application processing as part of the core IT framework, management can achieve true "straight-through processing" and realize significant increases in efficiency and productivity. Gains are recognized through:

- Quicker application deployments.
- Fast, reliable integration for multiple applications.
- Accelerated delivery of information to decision makers.
- Higher application service levels.
- Flexible workload balancing.
- Scalable and repeatable business processes.
- Reduced manual intervention and processing latency.
- Lower production and maintenance costs.
- Automated distribution of critical reports and data.

Business Resides in the Batch

According to Milind Govekar, research vice president of Gartner Inc., "Batch integration forms 70 percent of a company's integration requirements. To drive business, batch processing must progress from simple date- and time-based processing to event-based processing across a variety of business applications and operating systems."

Tools such as batch schedulers have been superseded by sophisticated automation tools that can drive business processes enterprise-wide. IT staff can design and define the business process at a simple object level, which can then be assembled into intricate process flows that in-

GO ONLINE www.enterprise-architect.net

Read this article online:
Down With Downtime

Read these related articles on FTPOnline:
Windows Server 2003 Maintenance Made Easy
Achieve 99.999% Uptime
Unifying Data, Documents, and Processes

Dan McCall *has served as vice president, COO, and director of AppWorx Corporation since 1994. McCall also has management experience in the petroleum, real estate, and banking industries.*

corporate if-then logic and dynamically supplied parameters to facilitate straight-through processing. These job streams mirror the way business analysts and corporate governors have mapped the business process. Business services are initiated automatically, executed across multiple applications and over disparate platforms, with no manual intervention.

When considered as an essential part of the infrastructure and implemented in the development phase, automated business application processing allows CIOs and IT managers to focus staff on proactive tasks instead of reactionary pain points. Much of the manual scripting and maintenance that was required to complete a business process has been eliminated. Now, instead of hundreds—or even thousands—of custom scripts, individual processes are defined at the object level. If a new application comes online or workflows change, you can assemble new process chains by adding new objects to your repository using a Java-based, drag-and-drop interface. Should existing objects need to be changed, IT staff can edit a master object definition to make the change cascade through all instances, instead of manually recoding hundreds of scripts.

The adage "we're only human" accounts for the fact that people sometimes make mistakes. Sure, it's natural, but that doesn't mean it needs to impede the speed of your business operations. Recovering from human error introduced during business application processing can be catastrophic in the near-realtime model of modern business. With little time available for rollbacks and recovery processes, it's necessary for organizations to seek further automation and integration of business applications.

Automation delivers assurance that your business processes are being executed exactly as you have defined them, on time, every time. Centralized automation tools monitor each process, recording vital information and reporting any issues to operations staff. This watchful eye over your applications relieves IT staff from the babysitting role that once was required to ensure successful execution.

Above and Beyond

In recognizing the need for a centralized, enterprise-wide application processing solution, IT architects add a layer of enterprise software over the entire application landscape. These tools provide communication, control, and management of job processing between and among each and every application, even homegrown or legacy applications.

As a noninvasive solution, the software allows for quick deployment and integration into any system. Best-of-breed automation tools can access live application data and use that ever-changing information to initiate batch processes, pass dynamic parameters, check for job processing conditions, and perform automatic recovery routines custom made to fit your particular business requirements.

From a single hub, business processes throughout the organization can be assigned priorities, and sophisticated process monitoring will shift job executions based on incoming needs and data. This flexibility in your processing environment assists with workload balancing and helps meet and exceed application Service-Level Agreements (SLAs).

A central software solution also offers the advantage of detailed log files and custom report generation. Each business process run is logged into a single repository, creating an easily audited account of your business activities. Reports can be generated based on your needs and automated for delivery in a number of formats, speeding up distribution of mission-critical information throughout the organization. Advanced reporting capabilities can assist with compliance efforts necessitated by legislation, such as the Sarbanes-Oxley Act. These reports show what processes were run and when, as well as who ran them.

Modern enterprise is always evolving. New technologies and shifts in architectural practices are ongoing challenges for all IT managers. Faster performance, impeccable accuracy, and steadfast reliability—on a limited budget—are their goals. In an environment of uncertainty, one thing remains absolute: Bulletproof business application processing is elemental to the attainment of corporate objectives.

Incorporating an enterprise automation methodology into your infrastructure positions your business to react to change proactively instead of succumbing to reactive, point-solution changes that require large investments of IT time to solve. Imagine complex processes executed using live data and requiring no manual intervention or custom scripting. Imagine flexible workload balancing, comprehensive reporting, and consistently exceeded SLAs, while reducing maintenance and overhead costs.

Plenty of talk circulates about aligning IT with business, and new industry buzzwords appear every day. Batch scheduling has long been the silent workhorse of the IT enterprise, but today's solutions have evolved to become hybrid providers of enterprise application integration and business process management. Maybe it's time to prepare your factory floor for an agile future by reexamining the execution and efficiency of business application processing in your organization. Go on, be a visionary. ■

Governance:
The Elephant in the Room

by Mike Dunham

The often unspoken EA issue: Is a new streamlined definition or governance necessary for effective management discipline?

For enterprise architecture (EA) to be effective as a management discipline, it must be agile and flexible in addressing a multitude of issues. It must be able to identify and develop enterprise solutions within the confines of external institutional boundaries. It must harmonize and integrate the artifacts it develops with those of related management disciplines, such as capital planning and investment control, strategic planning, performance management, and security. At appropriate times, it needs to be a strategy tool and, at others, a tactical device.

However, the most important component of EA is frequently overlooked—the importance of stakeholder relationships and ensuring that the stakeholders are integral players in the design and implementation of enterprise solutions. EA boils down to the management of stakeholder relationships in order to effectively advance enterprise solutions.

The essential component and the one that is most often neglected is governance—establishing the rules of engagement to "lubricate" these processes. Governance is the "elephant in the room," or the obvious ingredient that no one wants to discuss, much less take on. It is essential to the successful transformation of theoretical EA concepts into measurable results. All too often, EA does not transcend the conceptual level, and as a result, practitioners are unable to translate the concepts into day-to-day requirements and deliverables. Including key stakeholders in the process—from concept to tangible result—is crucial to the success of EA.

In the absence of a universally accepted definition, business managers often define EA in terms of solutions to their current tactical needs. Like any other management discipline, they view EA as a tool to solve their business needs and characterize it in terms of the management disciplines with which they are most familiar. The result is multiple interpretations of the meaning and purpose of EA. Until EA matures, multiple definitions will persist.

The absence of a universal definition hinders the discipline from advancing and potential breakthrough performance from occurring. The key to defining EA is to develop a definition that is broad enough to incorporate governance and theoretical EA concepts, while focusing on delivery of tangible, measurable results. Hence, I propose the following: *Enterprise architecture is about relationship management.*

EA is about managing two types of relationships: artifact and stakeholder relationships. In order to identify and develop effective en-

GO ONLINE www.enterprise-architect.net

Read this article online:
Governance: The Elephant in the Room

Read these related articles on FTPOnline:
Align Java Technologies With Business Results
Bridge the CIO/CEO Communication Gap
Risk Management—From Adversity to Advantage

Mike Dunham *is Chief Enterprise Architect at Thomas & Herbert Consulting LLC.*

terprise solutions, it is important to understand the relationship between those artifacts generated by the technical architecture and those produced by the business architecture. This is the traditional concept of EA embraced by most practitioners. This component of EA comes in many flavors and shades, from the technical architecture through data and service architectures. Artifact relationships, however, represent only 25 percent of EA.

Not surprisingly, the owners of the technical and business artifacts are among the stakeholders. These key stakeholder relationships are where the "rubber meets the road," and they represent 75 percent of the discipline. In addition, for an enterprise initiative to be successful, the business managers who have a stake in the new initiative must become fully involved in developing the rules of engagement that will govern implementation of the initiative. All of these activities fall under the general term "governance."

Strong stakeholder involvement and effective management of the stakeholder relationship will tend to eliminate traditional barriers to successful implementation of enterprise solutions, such as:

- Failure to develop strong business cases that provide incentives for stakeholders to participate.
- A history of major systems failure in connection with implementation of past enterprise solutions.
- Poorly documented artifacts that hinder stakeholder implementation.
- Resistance to new initiatives that are perceived as disruptive to standard operating procedures.
- Risks involved in incorporating a new enterprise solution that might expose the organization to failure in achieving its mission.
- Managers who feel that they are losing control of their operations or fear being dependent upon other organizational entities to provide mission-critical services through a new enterprise solution.

Faced with such issues, managers will inevitably find ways to avoid implementing a new enterprise solution. Governance processes provide the lubricant to help key stakeholders address the friction points and assist them in successfully managing the new initiatives. Again, it is the elephant in the room. It is the issue that many EA practitioners pass along for someone else to handle, or they assume the issues will somehow take care of themselves.

Based on this definition of EA, initiatives become more than the identification of potential solutions. Once identified, an enterprise governance body must be developed and nurtured to ensure that rules for data and resource exchanges are appropriately managed. EA governance must ensure that unambiguous dispute resolution procedures are in place to handle the inevitable disagreements that will arise:

- How are services delivered?
- Who delivers them?
- Who pays for them?

Rather than trying to define EA in all its complexity, it may help to start with this simple definition and let individual practitioners leverage it to their own needs. Perhaps by using this simpler definition, practitioners can avoid arguments as to whose view of EA is more relevant and stop the ongoing debate on whether EA is about configuration management, portfolio management, data normalization, or none of the above. ■

FTPOnline

One Source for All Your Technical Information

Newly Expanded, Easily Accessible

1

CHANNELS
To better serve your needs, FTPOnline has been restructured around seven channels: Architecture, Java, .NET Development, Windows IT, ASP.NET, Database and Security. More channels to come!

2

SPECIAL REPORTS
Get comprehensive information on subjects critical to all IT professionals, such as Security, Service-Oriented Architecture, and Operations Management.

3

WEBCASTS
Watch and listen to industry experts discuss hot IT topics.

4

WHITE PAPERS
Download white papers that examine evolving technologies.

5

RSS FEED
Get quick updates on the latest blogs and articles published at FTPOnline.

6

MAGAZINES
Filled with downloadable code, interviews with industry visionaries, in-depth tutorials, overviews of implementation and management strategies, article archives, and more!

7

NEWSLETTERS
Free e-mail newsletters in your area of interest, delivered right to your inbox.

Go there today:

www.ftponline.com

FTPOnline