Learn the discipline, pursue the art, and contribute ideas at www.ArchitectureJournal.net **Resources you can build on.**

THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

Journal 13

Software + Services

Microsoft®

Contents

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes

Journal 13

Founder
Arvindra Sehmi
Microsoft Corporation

Editor-in-Chief
Simon Guest
Microsoft Corporation

Microsoft Editorial Board
Gianpaolo Carraro
John deVadoss
Neil Hutson
Eugenio Pace
Javed Sikander
Philip Teale
Tim O'Brien

Publisher
Lisa Slouffman
Microsoft Corporation

Design, Print, and Distribution
CMP Technology – Contract Publishing
Chris Harding, Managing Director
Angela Duarte, Publication Manager
Kellie Ferris, Corporate Director of Creative Services
Jimmy Pizzo, Production Director

Foreword

Dear Architect,

As software architects, we often face technology decisions early on in the development process. Fat or thin client? Mobile or desktop? Local installation or Web application? Instead of choosing based on "or", what if we could pick something using "and"? Why do our choices have to be constrained to one particular technology, when the best solution is often a mix of both? That's the concept behind the theme of this issue of *The Architecture Journal:* Software + Services.

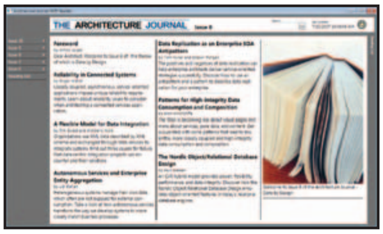
This issue's article lineup explores a new vision of Software + Services, as outlined by Microsoft Chief Architect, Ray Ozzie, in his keynote at MIX 07 this year.

We lead off this issue with Don Ferguson, who you may recall was our featured architect in Journal 11. Don shares his views on the architectural aspects of how to relay messages across the Internet using an ISB – or Internet Service Bus.

Following Don, and as part of our profile series, we are excited to share an interview with Ray Ozzie himself. In his interview, Ray shares some of the details of the Software + Services vision and what it's like to be Microsoft's chief software architect.

After Ray's interview, we get a technical overview of Project Astoria from Pablo Castro. Astoria is a new service that exposes data to Web clients within a corporate network and across the Internet. Kevin Sangwell follows with his thoughts on the implications of services consumption by enterprise IT, which lead on nicely into an article on mashups in the enterprise by Larry Clarkin and Josh Holmes.

To wrap up this issue, Chip Wilson and Alan Josephson explore the use of Microsoft Office as a platform for Software + Services. Finally, Gianpaolo Carraro's amusing analogy asks the question, *What if architecture was a planet?* In his piece, Gianpaolo takes an intra-world perspective to reveal more about the benefits of using Software + Services.


THE ARCHITECTURE JOURNAL
Foreword
Reliability in Connected Systems
A Flexible Model for Data Integration
Autonomous Services and Enterprise Entity Aggregation
Data Replication as an Enterprise SOA Antipattern
Patterns for High Integrity Data Consumption and Composition
The Nordic Object/Relational Database Design

Here at *The Architecture Journal,* we like to "practice what we preach." To help demonstrate this, we're proud to announce a new offline experience for the Journal, called the "Journal Reader."

Demonstrating many of the principles highlighted in this issue, this new reader is a locally installed application that enables you to take every issue of the Journal into a searchable, immersive, and easy-to-read experience. The application synchronizes with our content management services so that you'll automatically have access to the latest Journal issues without needing to download PDF files or checking online.

We receive a lot of feedback from you about how you read the magazine and hope that this new service offers a unique and useful way of reading the magazine. In early November, you'll be able to download the reader and get more details from our Web site, http://www.architecturejournal.net.

Simon

Simon Guest

The Internet Service Bus

by Donald F. Ferguson, Dennis Pilarinos, John Shewchuk

Summary
Web applications are an extremely common application model, and they will become increasingly dominant. Almost all applications in medium to large businesses offer a Web user interface. In this article, we will use the term enterprise to encompass medium to large enterprises, and the software vendor and integrator companies. Desktop and client/server applications are increasingly using the browser for the UI engine and making calls to data and services through Web protocols.

Software, application models and the Web itself are undergoing a revolutionary transformation. This transformation will have as large an effect on computing as the client/server model or the initial emergence of the Web itself. The Web will evolve from connecting users to applications provided by sites into a model in which:

- The application executes "in the Web."
- End-users develop their own applications for accessing the Web, transforming it into an end-user developed workspace for access to Web services.

This paper focuses on a small number of elements of the transformation. Other papers will expand the vision.

Several technologies and trends provide the impetus for the revolutionary transformation described above. Some examples are: multi-core processors; virtualization; application scenarios that federate many devices, such as mobile phones and tablets; Service-Oriented Architecture (SOA) and Web services; Web 2.0; and Software as a Service (SaaS)

We will discuss the effect of some of these trends, but our primary focus is SOA, Web 2.0, and Software as a Service (SaaS). These concepts and their relationships are not well understood. The technologies often appear to conflict. The paper describes the elements of a high-level reference architecture that brings the concepts together into a consistent whole.

Many of the preceding technology trends have broad acceptance and awareness. This paper discusses a third trend that is more controversial: *ubiquitous programming ability.* A large percentage of high school and college graduates have basic programming skills when entering the workforce; many students have developed simple PHP or Visual Basic applications and built Web sites. The professionals' primary job responsibilities may not include programming, but in many cases professionals will do simple programming if it makes them more productive. They may also develop simple applications because it is "cool." We will use the terms *end-user programming* for this concept.

End-user programming is an extreme case of *opportunistic development,* which occurs in departments and Lines of Business (LOBs) in enterprises. LOBs and teams often build simple, "quick and dirty" SharePoint or PHP applications that solve an immediate business problem by extending packaged applications or enterprise-wide core applications.

Opportunistic development contrasts with *systematic development*. Systematic development is model-driven, replete with requirements gathering, use cases and stakeholder interviews, an application development life cycle that includes quality and assurance, and so forth. Systematic development is the predominant model of the enterprise development team ("the CIO team"). Many packaged application developers (independent software vendors or ISVs) and Systems Integrators (SIs) also produce systematic solutions.

There is a tension in enterprises between opportunistic development and systematic development. This tension will increase if end-user programming becomes common. End users will not be content to wait for the systematic development teams to develop or modify solutions. The reference architecture we describe provides an approach for reconciling the extremes of opportunistic and systematic development.

We use a scenario to illustrate the reference architecture. A core element that emerges and underpins the architecture is an *Internet Service Bus (ISB)*. The reference architecture encompasses many elements, however, this paper provides detail only for the ISB. Other papers will describe other elements.

The Scenario and Opportunistic and Systematic Development

Dave travels frequently on business, and uses preferred hotels and airlines. He uses local town car services in the cities he visits and makes reservations at restaurants. Collaboration with friends, family and colleagues is also important. Dave uses the various travel providers' Web sites to make and change travel plans.

Dave's managing of trips involves a lot of manual tasks interacting with travel providers through their Web sites. He has to manually coordinate tasks that span sites and cut and paste data between fields. There is also some sequencing logic, for example booking a restaurant reservation and car service to get to the restaurant. Dave acts like a composite application or Enterprise Application Integration (EAI) solution. The manual work is tedious and error prone. Dave has basic programming skills and decides to write a little mashup. The mashup uses the travel providers' Web sites through client-side Web page scripting or simple HTML clipping (Figure 1(a)). The mashup makes Dave's life a little easier and makes Dave more productive at work because he spends less time managing his travel and more on his job. The mashup is also cool, which impresses his friends.

Mary and Ludwig like the application and get the code from Dave. They want to have different UIs but share code. So they improve the application by separating the UI from the Web site access, scripting, and caching by implementing a simple model-view-controller version (Figure 1(b)). The improvement also enables code reuse for access through other devices like PDAs or mobile phones (Figure 1(c)). Finally, they decide to move the model layer to a department Web server and implement a simple Web application. This enables multi-person access to information, for example for assistants.

The friends have opportunistically built a composite application, which is a simple pseudo-EAI solution. As programming-capable professionals enter the workforce, these ad hoc and just-in-time applications will become more common. There is the "cool factor" and the applications will simplify tedious tasks. Professionals may also build the applications "just in time" for short duration business problems, such as a convention. The applications are analogous to the role of spreadsheets when a user performs a business task by accessing existing databases and core enterprise applications.

Opportunistic situational applications will have a profound effect on enterprises' systematic application delivery. One effect will be on enterprise application development. The situational applications may "pound on" core enterprise applications, or use the core systems in unexpected ways. This will cause the IT organization to move some of the "model layer" to enterprise servers to improve performance and integrity. In essence, the situational applications have defined use cases that can drive systematic enterprise application transformation. The situational applications can replace simple mockups for documenting use cases, and can drive formal modeling.

Many pressures will move as much of the situational applications to enterprise servers as possible. There may be partners that need to use the application, which requires enterprise security. Some applications may be part of important business decisions like loan approval. Governance and compliance will require logging data access and input, and saving versions of the code for these applications. Moving the situational applications to the data center has profound implications. The data center will need to support hundreds or thousands of frequently changing applications in addition to the core business solutions. The data center needs to manage dozens of heavily loaded core application servers accessed by thousands of users, *and thousands of virtual servers infrequently access by small teams using ad hoc applications.*

There are many scenarios in which systematic solutions consume opportunistic applications. Dave will think it is very cool if IT uses one of his applications.

In summary, opportunistic applications drive systematic solutions and systematic solutions drive hardening of opportunistic applications (for example, Representational State Transfer (REST) -> Web services). These dynamics also drive software as a service, or more accurately, *software and services*. Software and services provides a platform for bringing together systematic and opportunistic application development and delivery.

Software and Services, and an Internet Service Bus

Enterprise Service Bus

Consider what happens if an airline cancels Dave's connecting flight from Dallas to San Francisco while Dave is traveling from New York to Dallas. Dave will not make it to San Francisco and he will need to stay overnight in the connecting airport city. It is necessary to change hotel, restaurant, and car service reservations. Dave could use his mashup to simplify making changes when he gets off the airplane. It would be better ("cooler") if rebooking could occur automatically while he is flying. Airlines and flight monitoring sites emit feeds with flight schedule updates. Ideally, Dave's application would be always executing somewhere "in the cloud." The application would monitor feeds and use simple logic to react to events and change the itinerary and plans.

It is unlikely that a simple end-user application could *always* repair the itinerary, but most repairs are often simple. Dave would only manually handle the complex cases, and could also approve the changes his application automatically made while he was flying.

The general application scenario of repairing the itinerary is a common type of problem within enterprises. Similar problems can occur for purchase orders and expense account approval, for example. The scenario is an example of a composite application that implements a Straight-Through-Processing (STP) pattern (see Resources). Enterprises implement a systematic approach to solving these problems. Figure 2 provides an overview of what the composite application might look like if the airline, hotel, restaurant, town car and other systems were within the enterprise firewall. A relatively long running orchestration process subscribes to events that the flight management system emits. The process sends messages to/from and calls the existing applications to cancel reservations, query availability, and make new reservations. Enterprises are heterogeneous and the existing applications have diverse message formats

Figure 1: Mashup Evolution

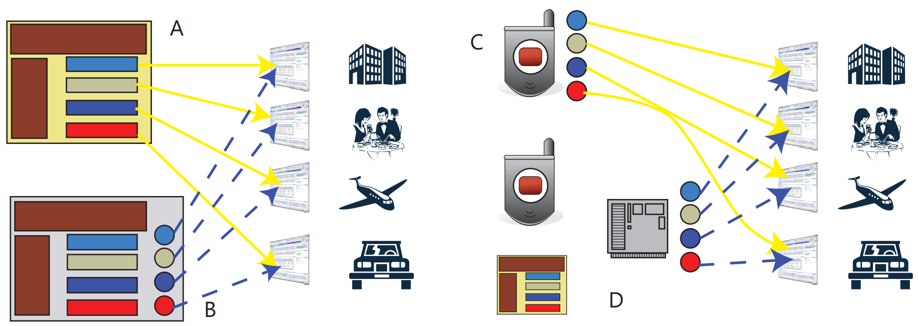
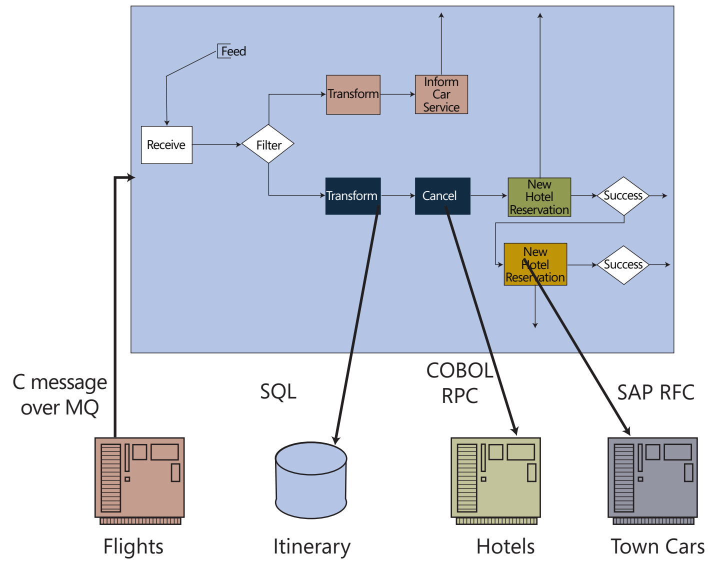


Figure 2: An Enterprise Business Process



(C, COBOL, and so on) and communication protocols (WebSphere MQ, SAP RFC, for example).

The repair process design shown in Figure 2 is fragile. The business process must change if another airline application is added, for example. The business process may also be coupled to specific message formats and languages of the existing applications. It would be difficult to add a general mechanism for logging messages that match certain conditions—for example, log all messages if the traveler is an executive. This fragility has led enterprise application architectures to evolve to an enterprise service bus (ESB).

Figure 3 provides an overview of an enterprise service bus. Application adaptors convert existing formats and protocols into standard Web services. This transforms the NxN protocol/format mapping problem of connecting anything to anything into N->1 mapping problem—that is, everything into a standard. The ESB provides additional functions that process messages flowing between services. Examples include message transformation, logging, and routing.

If the enterprise development and business units of Dave's company decided that implementing his travel application were important enough to fund, the enterprise team could implement an application similar to Figure 3. There are several problems with developing this systematic solution:

1. There is no guarantee that the enterprise would choose to fund the composite application. There may be other, more pressing business problems.
2. Systematic development involves use cases, some form of process modeling, interviewing stakeholders. These take time.
3. The airlines, hotels, and other applications are external to the enterprise. Enterprises are very deliberate and cautious about setting up business-to-business connections. Even if the business partner implements Web services, the enterprise will need to establish authorization rules and auditing for Web service interactions with partners. The enterprise will need to support user identity management, federation, and provisioning because an employee will have an intra-enterprise identity and identities in the airlines and hotels.
4. Customizing the solution for individual employees' preferences is complex. The employee lacks the ability to perform "do it myself" customization. The application resides on central enterprise servers. IT professionals define and modify the business process, not Dave.

It would be really cool if Dave could do a simple, personal version of the systematic solution. If we generalize the ESB and think of it as one type of service bus that is optimized for systematic enterprise development, we could also imagine a type of service bus that is optimized for opportunistic development. This is the Internet Service Bus (ISB). The ISB is more like a ubiquitous fabric. The ISB links devices to each others, devices to local servers, Web sites to Web sites, and ESBs to ESBs, and is itself an ESB. The ISB is a platform for "do-it-yourself" composite applications and business processes. The ISB is also an example of Software as a Service (SaaS).

Internet Service Bus

Figure 4 provides an overview of the Internet service bus concept. (An early example of an ISB is BizTalk Services; see Resources.) An ISB provider is analogous to a PHP Web site hosting company. Both provide an application platform in the "cloud." A PHP Web hosting site primarily provides a platform for developing dynamic Web sites and Web services that interact with a database. In contrast, the ISB provides a platform for creating and deploying composite applications that integrate services that other sites provide. ISBs, PHP Web hosting companies, and storage as a service such as Amazon's S3, are examples of application enablement infrastructure software as a service. This contrasts with Salesforce.com, which was initially packaged application software as a service.

The core ISB concept is built around a Uniform Resource Identifier (URI) space. Dave's team working on the application registers and "owns" the URI http://ISB.net/DaveAndTeam. URIs below this root represent application integration points, and are similar to destinations

Figure 3: An Enterprise Service Bus

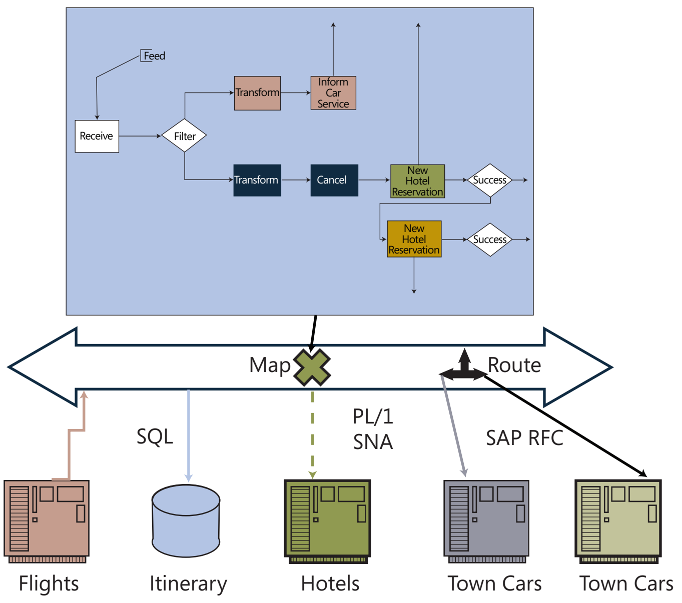
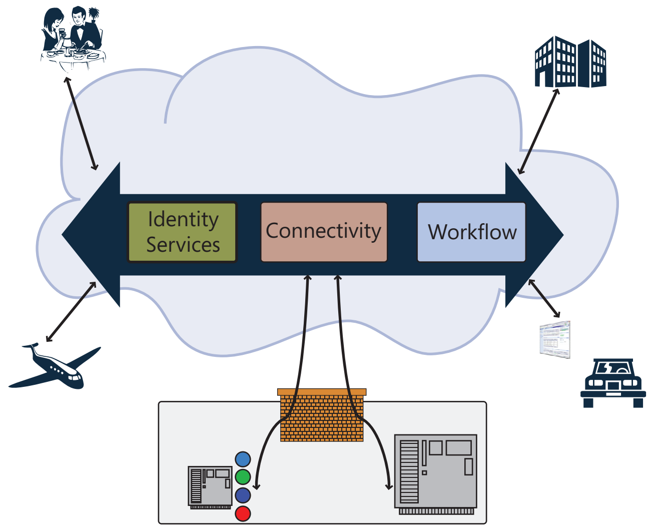


Figure 4: An Internet Service Bus



in the Java Messaging Service, queues in message-oriented middleware, or topics in publish/subscribe systems. The team develops an ISB application by associating policy and function with URIs. The composite application is a set of URIs, policies and functions.

The ISB provides an identity and access function for controlling which messages can be sent to a URI, and by whom. The identity and access function is an example of associating a policy with a URI.

For example, Dave may choose to maintain a wiki page on a public Web site that shows his travel reservations. Dave will want to control access to the wiki page. Setting up and maintaining authentication and authorization databases at his personal Web site can be tedious. The problem will become more complex if Dave has pages and data at several Web sites, for example:

- His personal database driven PHP site
- A family collaboration portal built using http://www.twiki.org/
- A presence on a spaces site like Windows Live Spaces (http://home.services.spaces.live.com/).

Dave's friend Don can register with the ISB's identity component and create a user ID don@foo.bar. Dave can use the identity component's Web UI to specify which of Dave's ISB URIs Don can access. Dave can also define groups and grant access to the groups. Don can access the URIs after having logged onto the ISB. The ISB simplifies Dave's security management because he can maintain a central database, and then authorize the "ISB" to access his wiki and other resources. The ISB protects the actual resources through access control for the ISB URIs that front them. The ISB benefit is that there is a single space for Dave to define and maintain identities, groups, resources and access policies for all of his "services" on the Web.

We've just described explicit user actions through Web pages. Another very common approach will be to have the applications at endpoints participating in the composite application use Web service APIs to access the ISB.

The identity component will also support WS-Security's Security Token Service (STS) functions, and federate with other STSs. This allows Dave to manage access for identities not registered with the ISB. If foo.bar was a company that Dave trusts and implements an STS, Dave can define access policies for identities authenticated foo.bar.

Over time, ISBs will offer additional policies and implementations that can be attached to URIs. Examples may include WS-ReliableMessaging or implicit message logging. The concept is similar to associating quality-of-service policies with connections in message-oriented middleware.

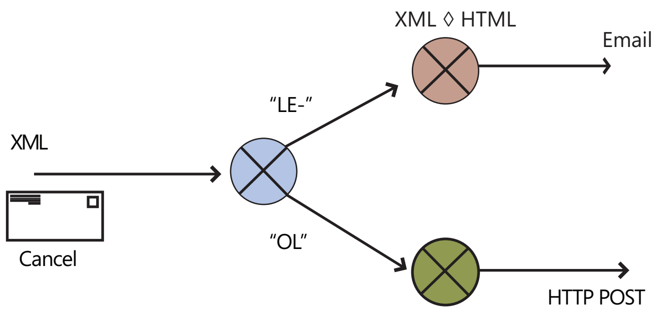
The ISB builds on the identity and access capabilities to provide "secure universal connectivity" for applications – even for applications located behind firewalls. This includes support for a wide range of connectivity patterns and protocols. Examples include REST-oriented HTTP, WS-*, and the event-driven patterns found in many enterprise applications. Specifically the ISB's connectivity component offers three core functions:

1. *Relay* enables communication between the ISB and applications behind firewalls. There are many techniques for accomplishing this capability (Biztalk Labs, see Resouces). The relay function eliminates the need to set up systematic cross-enterprise connections for simple scenarios.
2. *Protocol* provides a set of common protocols for exchanging messages, such as WS-* or REST. The ISB also provides protocol mapping for automatically connecting endpoints using different protocols. For example, it is possible to connect an RSS feed to a WS-* message connection without modifying either application.
3. *Functions* provide support for associating simple ESB-like functions to the URL. Examples could include multicast, WS-Eventing, persistent messaging, etc.

The connectivity layer operates at the infrastructure technology level. It simplifies solution development by removing complexity due to different "plumbing"—for example, REST versus WS-*. Projects that must implement infrastructure integration at this level incur significant cost and risk. The ISB eliminates these problems.

The connectivity layer is unaware of application level elements and message formats. Building a composite application requires adapting between different message formats that the connected services implement. An example of the ISB's functions would be converting the parameters in an HTTP GET into elements in an XML message. The ISB offers a simple workflow (service choreography) that provides support for application level mapping (Figure 5).

Figure 5: ISB Message Processing



The ISB offers a set of template "activities" for simple functions. A workflow is a graph composed of instantiating activity templates. Suppose that the airline emits flight status through an RSS feed and part of Dave's application expects to receive WS-Eventing notifications for the updates. The connectivity layer supports integrating RSS and WS-*. It is still necessary to convert the message payload from the RSS format to the XML event format that Dave's application expects. The ISB will typically offer a configurable, reusable activity template for RSS to XML mapping.

Another common activity template will be selection-based routing. Dave's application may emit a *cancel car reservation ID=1234* message. If one town car service's reservation codes begin with "LE-" and another's begin with "OL," Dave's application can send cancel events to a single ISB URI. The selector then processes the message and routes to the correct endpoint.

Combining activities for more complex message processing is useful and will be a common function of ISBs. As an example, Figure 6 shows the activities at the URL that Dave defines for receiving the cancel car reservation message:

1) Receives the cancel message in XML using WS-*.
2) Has an activity that extracts the reservation ID element and looks the prefix up in a table.
3) Transforms the message to the expected format of the town car service, which is
a) HTML email for one provider
b) HTTP POST for the second provider.

Building message processing functions can be quite simple. Many, many of the common application scenarios are simple instantiations of patterns and templates. An ISB provider will offer a simple Web-based application development tool that allows the developers to select

"TOTAL "SOFTWARE AS A SERVICE" IS A MYTH. ALL MEANINGFUL SAAS SOLUTIONS WILL EVENTUALLY INCLUDE SOME ON-PREMISE SOFTWARE – A HYBRID. ALMOST ALL SCENARIOS THAT USE AN ISB AND SAAS ARE ACTUALLY HYBRID OF ON-PREMISE AND OFF-PREMISE SOFTWARE. A TERM FOR THE HYBRID MODEL IS SOFTWARE + SERVICES."

activity templates and set the configuration parameters through a Web form. In the routing case, the Web form would allow the developer to specify the message field for routing and values in the routing table. Over time, ISBs will offer more powerful tools like the message processing tools in BizTalk.

Message processing (routing, transformation, and so on) is powerful enough for many application scenarios. Simple sequencing and flow of control is required for others, however. Consider the task of booking a hotel in Dallas when Dave is stranded. A simple description of the process could be:

1) Send a reservation request to hotel chain AAA
2) Receive a response.
3) If success then exit
4) Send a reservation request to hotel chain BBB
5) If success … …

Workflow activities extend message processing with activity templates for flow of control like while, if … then …, and so forth. ISBs will incrementally add support for simple workflow to extend the basic message processing.

Workflow can appear to be a complex concept, and systematic enterprise workflow solutions are powerful and complex. The vast majority of workflows for ad hoc, opportunistic applications are extremely simple, however. The structure is not significantly more complex than simple PowerPoint diagrams. There is a small set of "clip art" for connections and shapes, and the developer sets properties on the shapes to express the behavior of the activity.

Most workflow processes tend to have the structure of a nested list. This enables simple tools for building the workflows. A simple XSD can provide the structure for XML documents defining a nested list workflow. A visual tool allows the developer to specify the activities and their implements or connection to external services. A broad community of developers is familiar with this model because Web UI frameworks often provide

Figure 6: ISB Message Processing

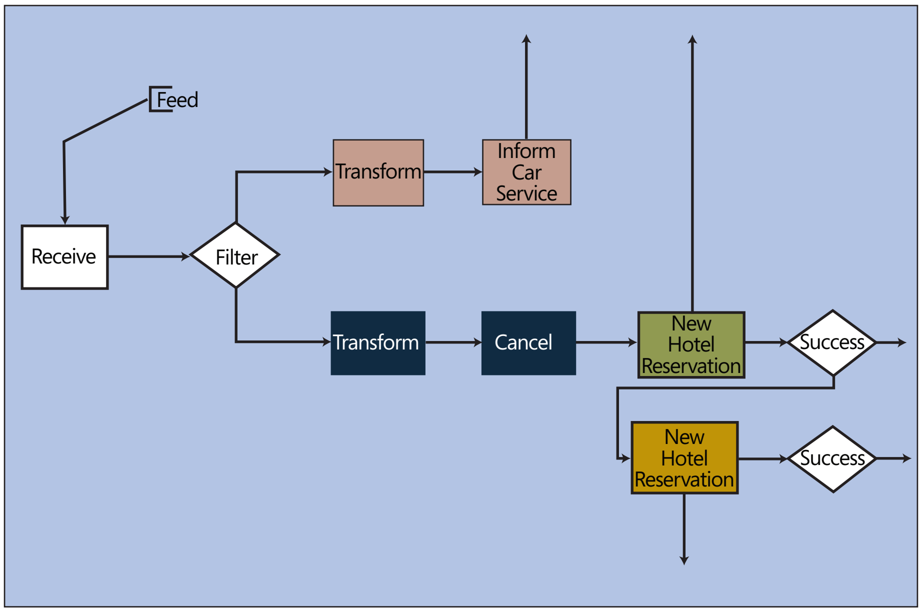
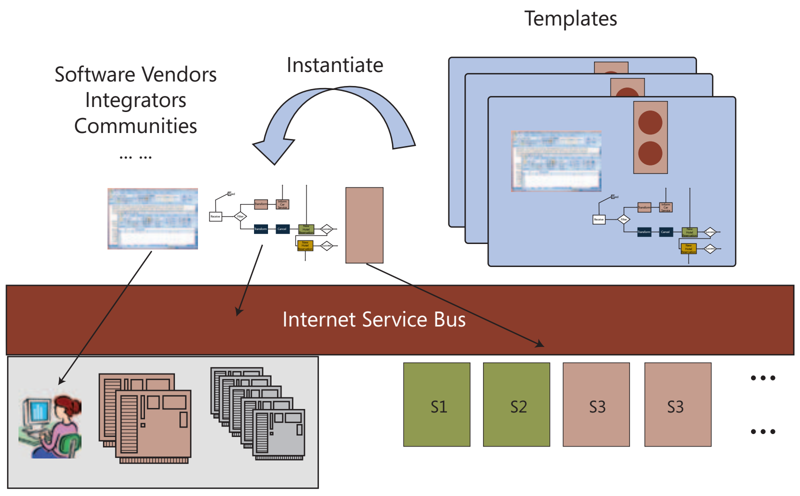


Figure 7: An Ecosystem and Business Model



a similar concept for page flows and transitions (Struts, for example).

Systematic workflow solutions are often complex because they are mission-critical and support applications that thousands of people use. The process modeling and engine must be able to express the full functions of the process and handle complex error conditions, approvals, and so forth. In contrast, for most opportunistic, ad hoc solutions, a small group of people uses the workflow, and the team is constantly tinkering with it to improve it.

Service Level Objectives

Enterprises deploying applications on the ISB will want to define Service Level Agreements (SLAs) specifying response time, throughput, availability, and so on. The SLA will determine the cost that the ISB provider charges. The general problem of achieving SLAs for arbitrary applications is daunting. The ISB's task is simpler, however, because it does not deploy arbitrary user code. Instantiating and configuring prebuilt templates for policy, publish/subscribe, workflow activities, for example, constrains the applications. This simplifies achieving SLAs, predictable cost, and integrity.

A Reference Architecture for Software and Services for Opportunistic Applications

Figure 7 provides an extremely high-level overview of how the strands in this paper come together. First, the Internet service bus is ubiquitous and connects all systems and servers. There will be many composite applications in which some elements are on the "ESB" and others are on the ISB. Multi-organization composite applications are one obvious example that would deploy elements onto an ISB in the cloud. Another possibility is single organization composite applications with a short duration. For example, a composite application that an enterprise uses to manage an internal conference. Reusing a preconfigured and installed "in the cloud" software platform may be more efficient than acquiring, installing, configuring, and supporting hardware and software to run the application "in house."

If the enterprise reuses the ad hoc application for several conferences, a systematic solution may emerge from the opportunistic solution. The opportunistic solution provides a concrete set of use cases for the systematic solution. It can also provide metrics for determining which aspects of the application are frequently used.

Third parties will connect value added services to the ISB. The first type of services will be infrastructure services, such as a more powerful workflow engine or an XML Query enabled database. Developers can include these services in their solutions by connecting them to URIs in their application. These infrastructure services are an example of how third parties can join the ecosystem by providing advanced infrastructure as a service.

The second type of service will be reusable business services—for example, a prebuilt service for maintaining product information and catalogs. Another example might be conference room scheduling for conventions. This is an example of a third party joining the ecosystem by adding an application "building block" service. ISB composite application can use the building block by connecting a URI in the composite application to the building block service.

Finally, system integrators and solution vendors will offer configurable, extensible solutions that are templates. A third party may offer a configurable solution that supports many of the conference/convention management functions. A packaged application vendor can offer the benefit of enabling "try and buy." Instead of shipping a CD that requires installation of the application and prerequisites, a potential customer can simply instantiate a version "in the cloud."

Community is an important aspect of Web 2.0. It may in fact be the most important aspect. Infrastructure services, basic application building blocks, and solution templates will also emerge through a community associated with ISBs that offer and share code. The community also provides a forum for self-service support and establishing the reputations of "software as a service" providers.

Software + Services

Total "Software as a Service" is a myth. All meaningful SaaS solutions will eventually include some on-premise software – a hybrid. Some elements of an instantiated solution will reside in the bus (workflows, for example), some will reside in services connected to the bus (such as an XML content management system), and some will "install" on premise. *Almost all scenarios that use an ISB and SaaS are actually a hybrid of on-premise and off-premise software.*

For another example, consider a data storage provider that Dave uses to store the itineraries in his application. Always using remote access to read/update the itinerary is fragile. The storage vendor will likely provide an on-premise and on-PC software package that optimizes data access through caching, replication, versioning, and so on. *A term for the hybrid model is software + services.*

Conclusion

Several trends are coming together to radically transform the Web application model. Currently, the Web primary enables connecting people to documents and applications. The fundamental transformation is thinking of the Internet and Web being the platform on which applications execute. Professionals with basic programming skills write personal applications that make their use of the Web far more efficient. They will share these applications with less computer literate friends and colleagues. Communities will emerge that provide another approach to spreading a personal solution "meme" through the community.

Inevitably, elements of the personal applications will "move into the cloud." A major source will be the broad use of "virtual" PCs that assemble themselves based on the user and nearby devices. Instead of using a notebook in a hotel room, the PC will assemble from the traveler's mobile phone and the TV, Internet connection and keyboard in the room. It is possible to assemble a Virtual Machine (VM) that includes just the software needed to implement a specific scenario. The VM also provides:

- Application isolation
- End-user administration by implementing a conceptual model similar to how the user manages personal computers
- Natural exploitation of scale-out processors based on multi-core.

The benefits to enterprise of the convergence of these trends include:
- Significant improvement in employee productivity and morale. Work is less tedious, more focused on business value tasks and potentially fun.

Increased agility and responsiveness because application development and modification can occur in hours instead of months.

A key enabling technology for these transformations will be an Internet service bus. SOA, Web services, and mashups enable rapid development of composite applications that integrate, customize and extend base application building blocks. Enabling these composites in the Web is the next major leap and a core aspect of Web 2.0. The critical element in achieving the promise is an Internet service bus. In addition to enabling flexible application development, an ISB enables an ecosystem of software providers. The capabilities of the ISB support the emerging populate of "programming literate" professionals entering the workforce, especially for bottom-up community development of long tail applications. The unifiying theory of the computing universe is software and services, and the ISB is the keystone of this new application model.

Resources

Biztalk Adapter
http://msdn2.microsoft.com/EN-US/library/aa744368.aspx

Biztalk Labs
http://labs.biztalk.net

Enterprise Application Integration (EAI)
http://en.wikipedia.org/wiki/Enterprise_application_integration

Enterprise Service Bus
http://en.wikipedia.org/wiki/Enterprise_service_bus

Mashup
http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

Model View Controller
http://en.wikipedia.org/wiki/Model_view_controller

OASIS Web Services Reliable Messaging (WSRM) TC
www.oasis-open.org/committees/wsrm/

SAP RFC
http://en.wikipedia.org/wiki/ABAP

Straight Through Processing
http://en.wikipedia.org/wiki/Straight_Through_Processing

Struts
http://struts.apache.org/

Use Cases
http://en.wikipedia.org/wiki/Use_cases

WS-Eventing
http://www.w3.org/Submission/WS-Eventing/

WS-Security Security Token Service
http://sts.labs.live.com/

Zorro's ISB Blog
http://zorroisb.spaces.live.com

About the Authors

Dr. Donald Ferguson is a Microsoft Technical Fellow in Platforms and Strategy in the Office of the CTO. Don focuses on both the evolutionary and revolutionary role of information technology in business. Prior to joining Microsoft, Don was an IBM Fellow and Chief Architect for IBM's Software Group (SWG) and chaired the SWG Architecture Board, which focused on product integration, cross-product initiatives and emerging technology, including Web services, patterns, Web 2.0 and business-driven development. Don's primary hobby is Kenpo Karate. He earned his black belt in December 2005.

Dennis Pilarinos is a senior technical lead in the Connected Systems Division at Microsoft. You can learn more about his work from his blog at www.dennispi.com.

John Shewchuk drives the Technology Strategy team for Microsoft's Connected Systems Division (CSD). In CSD, John has worked to develop Microsoft's application platform, including work on application messaging technologies like the Windows Communication Foundation, Web services interoperability specifications such as WS-Security, and identity and access technologies such as InfoCard. John co-founded the Indigo team and has been a key driver in cross-industry interoperability. With others on the Indigo team, John led development of the Web services architecture and specifications and managed technical negotiations with a broad range of industry partners.

Architecture Journal Profile: Ray Ozzie

Ray Ozzie is Microsoft's chief software architect, assuming the role from Bill Gates in June 2006. To coincide with the theme for this issue of the Architecture Journal, we caught up with Ray to ask him about his vision for Software + Services and some of his thoughts about becoming a software architect.

Microsoft Chief Software Architect Ray Ozzie outlines Microsoft's transformation to a world of live services at the company's annual gathering of financial analysts.

AJ: Many readers may have heard your keynote at MIX this year about Software + Services. Could you elaborate a little on the vision?

RO: There are fundamental changes that continuously occur within our industry related to the price of different types of components and the cost of communication. About every five years or so, we've found the need to re-evaluate the right architectures for systems based on changes that are occurring. Today, the confluence of cheap computing, cheap storage, and cheap communications is again causing us to re-evaluate where we put computing in order to deliver solutions and solve problems.

The Web initially was built for a low bandwidth world, utilizing a thin client or smart terminal, and assumed fairly low bandwidth with most of the computing power being applied at the service level. In the client-server era, we had a high bandwidth network within the enterprise that balanced processing on the client and the server. Given the changes in computing, storage, and communications, we as an industry, and we as a company, are reflecting upon the value we deliver to our customers and looking at the best way to balance what's on the client, what's on the enterprise server, and what's on the service in order to accomplish different scenarios.

We are undergoing a fairly dramatic shift in delivering solutions. I believe fundamentally the answer is never one extreme or the other unless you've got a really intense constraint like a thin pipe. There are some solutions that will be delivered as pure services. You'll just go to a browser; you'll complete your transaction, get your information, and do whatever you need to do. There are other situations where you're highly mobile and the connectivity to the Internet is less reliable from one spot to another. In these environments, the opposite extreme is true; you want to carry around as much data on your vast hard disk as possible, and have ready access to it.

For example, in the early days, the PC was mostly about documents. Now, of course, it's about media. People are taking digital photos, putting vast libraries of them on their hard disks, caching their music, etc. We also are seeing many camcorders going digital. Does this mean that you will have all your home movies on your PCs? Or replicated among your PCs? I believe there's a huge opportunity for our industry, in terms of looking, solution by solution, and asking what's the best way of balancing the use of client code and service.

AJ: It sounds like many of these solutions could fit within a spectrum of browser-based clients and rich interaction. Within that spectrum do you see different types of architectural patterns emerging?

RO: There are different patterns emerging, although we don't know exactly what they'll all be yet. The patterns for horizontal scale are the ones that I'm thinking are most challenging and most interesting at this point in time. What's very clear is that for any high-scale service you must have an image in your mind of a virtual machine that you scale accordingly to meet your needs. Refactoring your application so that it scales out as opposed to up, and also making it elastic in terms of very broad or very narrow scope is probably the most interesting right now.

There are certain design patterns like MapReduce that are clearly horizontally scalable design patterns, but these solve a relatively small set of problems compared to the large number of business applications that we have today. Over time, we'll ultimately find a middle ground and will end up with frameworks that think of your application in tiers. These tiers, as long as they fit within this pattern, should be horizontally scalable.

AJ: To the degree that these patterns emerge, or frameworks are provided, how do you think the Software + Services vision differs from consumers to enterprises?

RO: That's a really good question. I'm not so sure that the patterns are all that radically different, aside from the one thing in the enterprise that doesn't exist in the consumer space -- the enterprise server. If you're in the enterprise and you're building systems for your customers, it's going to be the same as if you were building systems for consumers. If you're building for employees there are more systems integration issues. You're probably going to want to build solutions that have some kind of affinity to another local server in the geography, and locating those based on data access patterns might be fairly significant. But over time, I do see enterprises embracing services more and more; initially for infrastructure services such as e-mail, communication, and other commodity-level services, before moving on to enterprise applications.

AJ: As readers think more about embracing a Software + Services model, where should they start today? And how do they know when they are successful?

RO: I'll start with the last one. If they are achieving business goals and objectives, then they are successful. As for where they should start, from my vantage point, I would recommend becoming extremely fluent in the technologies, both from a developer and a designer perspective. For the Microsoft platform, this means understanding Expression Studio and Visual Studio. Then continue, explore, and prototype things in WPF (Windows Presentation Foundation) on Vista; it's an amazingly powerful tool. As you know, I started programming back in the '60s – and in the early days there was a model of a program that you had in your mind. It began with *main* and you started writing. At a certain point, it transformed to *WinMain* and became more of an event-based model. The new model of programming is where you're thinking in a declarative model, starting with XAML and a canvas, and building your application from there.

This new model is different — until you start prototyping with it, you don't understand the power that you get, and how productive you can be once you've transitioned to this new mindset. Prototyping brings you to a point where you start thinking, "Oh wow! Now I understand how I could build something in Silverlight and deploy it out there very quickly to anybody via a browser." Following this approach is going to be much easier than figuring out how to write JavaScript for different types of browsers.

"REFACTORING YOUR APPLICATION SO THAT IT SCALES OUT AS OPPOSED TO UP, AND ALSO MAKING IT ELASTIC IN TERMS OF VERY BROAD OR VERY NARROW SCOPE IS PROBABLY THE MOST INTERESTING RIGHT NOW"

Ray Ozzie

Ray Ozzie

Chief Software Architect, Microsoft Corporation

Ray Ozzie, an industry visionary and pioneer in computer-supported cooperative work, is Microsoft's chief software architect. Ray assumed the chief software architect's role in June 2006, when Chairman Bill Gates announced his intent to relinquish his Microsoft day-to-day responsibilities in July 2008. In his role as CSA, Ray is responsible for oversight of the company's technical strategy and product architecture. Ray is also directing development of the company's next-generation software services platform.

Previously, Ray was chief technical officer from April 2005 to June 2006. He assumed that role in April 2005 after Microsoft acquired Groove Networks, a next-generation collaboration software company he formed in 1997.

Prior to Groove, Ray was a founder and president of Iris Associates, where he created and led the development of Lotus Notes. Before Iris, he contributed to the development of Lotus Symphony and Software Arts' TK!Solver and VisiCalc, and was involved in early distributed operating systems development at Data General Corp.

Ray earned a bachelor's degree in computer science from the University of Illinois, Urbana-Champaign, where he was first exposed to the nature and significance of collaborative systems and computer-supported cooperative work while working on the university's seminal PLATO project. This work significantly influenced his perspective on collaborative systems and the projects he has undertaken throughout his career. He's subsequently been honored as one of the school's distinguished alumnus.

Honored as one of seven "Windows Pioneers" by Microsoft, Ray was named "Person of the Year" in 1995 by PC Magazine, and has been inducted into the Computer Museum Industry Hall of Fame and the InfoWorld Hall of Fame. In November 2000, he received the Institute for Electrical and Electronics Engineers (IEEE) Computer Society's W. Wallace McDowell Award, and in 2001 he was honored as a World Economic Forum Technology Pioneer.

He has served as a member of the National Research Council's Computer Science and Telecommunications Board, and was a member of the NRC committee that produced the landmark CRISIS report on the societal impact of cryptography.

AJ: You mentioned a relationship between designers and developers. Historically, I would argue that these people have not communicated as well as they should. Do you see that changing with this new wave of technology?

RO: In any effective solution that has a design element to it, both had to find a way to get along. There's nothing Microsoft can do to change the DNA—meaning the type of person who is a designer or the type of a person who is a developer.

What we've observed is developers and designers stretching themselves in different ways. Some designers can go further in understanding the challenges developers face. And some developers can relate more effectively to the challenges designers face. Having tools where there's an overlap, where there isn't an absolute boundary between a design tool and a developer tool, is extremely important and helpful. The core of Microsoft Expression is really oriented toward a designer. The work that they do translates into XAML, which can be imported and used in Visual Studio by a developer. I'm excited and optimistic about how tools like this will bring developers and designers closer together, for the ultimate benefit of the user.

AJ: We have a number of Architecture Journal readers in developer roles today who aspire to be software architects. Given your responsbilities, what does a day in the life of Microsoft's chief software architect look like?

RO: From my vantage point, being an architect is really about pattern matching. It's about being exposed to enough tools and techniques of the trade that over time you start to develop a toolkit of different patterns that work in different situations. This is true of software architecture and probably other types of architecture as well. Whether you build bridges or design buildings, you're trying to apply design patterns to a given situation.

My role within Microsoft is an interesting one because there are very strong architects within the different product groups, within the different divisions of the company, and they're doing a terrific job on their products. My role is essentially a cross-cutting one. By that I mean understanding how customers are using multiple products together, and then asking myself what patterns I see. What's the smallest possible thing I could suggest to a product team that they could do to re-architect their product in order to minimize seams with other products? Or from a business perspective, what's the smallest possible thing I could overlay on these products to add value for our customers, and advantage our solutions in the market. At the very highest level, my advice to aspiring architects is, don't jump in too quickly. You need to do your time as a programmer to understand the different patterns that are out there, and recognize the attributes of well-architected systems, in order to raise yourself to the next level of abstraction in the solutions you're building.

AJ: It sounds like the ability to do pattern matching really comes down to experience?

RO: Absolutely. It especially pertains to the things that nobody at the architecture level likes to think about. For example, performance characteristics, IO characteristics, reliability characteristics… You might have had experience with a system that works well given a certain level of complexity, but if used in a more dynamic environment it could be too fragile. You only learn that through experience.

"AT THE HIGHEST LEVEL, MY ADVICE TO AN ASPIRING ARCHITECT IS TO FIND THE RIGHT BALANCE OF FOCUS ON INTERNAL AND EXTERNAL TRENDS THAT WILL GIVE YOU THE PERSPECTIVE YOU NEED."

AJ: You've been in the software industry for over 25 years, and clearly feel very passionate about technology and software. What keeps the motivation going? What gets you up in the morning?

RO: I like to solve problems. I enjoy complexity. I enjoy technology. I've been fortunate because very early in my career I had the opportunity to work on systems that were deployed at large scale. So I'm somewhat addicted to the notion of having a large impact in the things that you do. Each individual wakes in the morning with different motivations. For me, it's about being a builder. I like to solve problems that can positively impact people's lives.

AJ: I imagine that to do that you must have a wide array of knowledge across multiple technologies. Given that, and the product teams you interact with each day, how do you keep up to date?

RO: It's really an interesting combination. The easiest and most natural thing is to just talk to people in your sphere of influence, and gain exposure to different technologies as a part of your job. But in order to be successful long term, you need to stay in touch with the trends and what's going on externally, especially staying in touch with what customers or individual users are saying and doing.

I spend a fair amount of time reading blogs, tracking specific influencers that have very interesting voices both related to our products and completely unrelated to our products.

I go to a combination what I call 'head' and 'tail' conferences. I'll attend a few where the known industry influencers congregate. These allow me to track major competitive issues, or at least the outward presentation of what competitors might be saying and doing. But I also enjoy the tail conferences, where you can get a closer-to-the-ground perspective on what's really happening. I like to meet people who are just out of school, who have startups and understand the kinds of technology choices they're making and why. At the highest level, my advice to an aspiring architect is find the right balance of focus on internal and external trends that will give you the perspective you need.

AJ: That's definitely great advice. Related to this, what would you say are defining characteristics of an effective software architect?

RO: The most effective architects I've dealt with are the ones who've paid their dues. These are the architects who've spent time in the trenches building and debugging fairly complex systems. You can learn a lot about how things work by fixing other people's bugs. When something fails and the person has left the company, you can learn a lot by either reverse engineering or looking at the documentation. The more systems that you can learn from the inside out, the more you can develop an understanding for bad practice and good practice design patterns. As I mentioned earlier, it's this library of patterns in your mind that will define you as an architect.

Project Astoria

by Pablo Castro

Summary
Project Astoria delivers a set of patterns as well as a concrete infrastructure for creating and consuming data services using Web technologies. This article explores how modern data-centric Web applications and services are changing and the role Astoria can play in their new architectures.

Data is becoming increasingly available as a first-class element in the Web. The proliferation of new data-driven application types such as mashups clearly indicates that the broad availability of standalone data independent of any user interface is changing the way systems are built and the way data can be leveraged. Furthermore, technologies such as Asynchronous JavaScript and XML (AJAX) and Microsoft Silverlight are introducing the need for mechanisms to exchange data independently from presentation information in order to support highly interactive user experiences.

Project Astoria provides architects and developers with a set of patterns for interacting with data services over HTTP using simple formats such as POX (Plain Old XML) and JavaScript Object Notation (JSON). Closely following the HTTP protocol results in excellent integration with the existing Web infrastructure, from authentication to proxies to caching. In addition to the patterns and formats, we provide a concrete implementation that can automatically surface an Entity Data Model schema or other data sources through the HTTP interface.

While the software infrastructure provided by Astoria can be a useful piece of the puzzle when building new Web applications and services, it is just one piece. Other elements in these applications need to be organized in a way that enables interaction with data across the Web.

In this article, I will discuss the architectural aspects that are impacted by modern approaches for Web-enabled application development, focusing on how data services integrate into the picture.

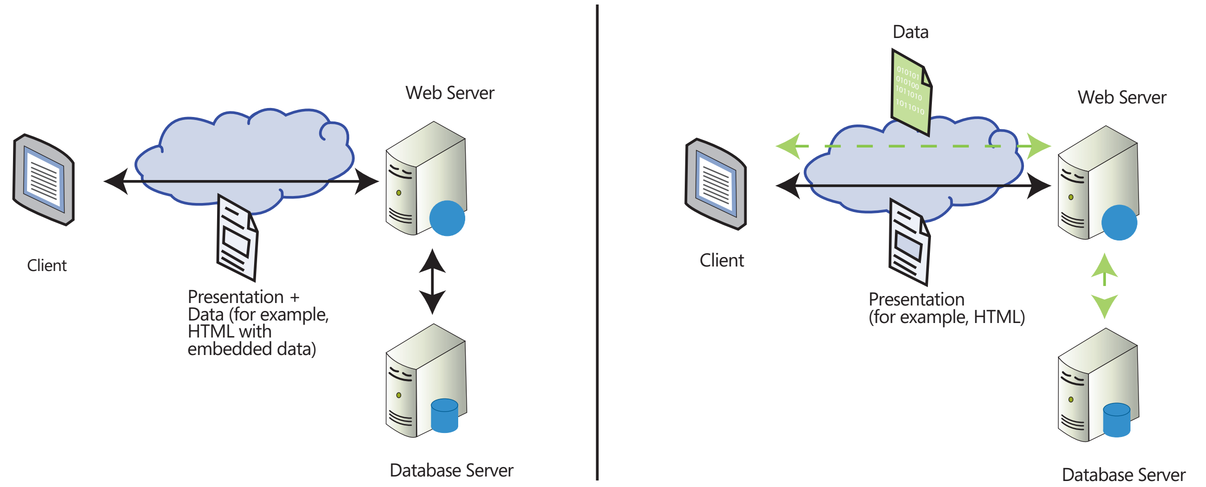
Separating Data from Presentation

The new generation of Web applications utilizes technologies such as AJAX, Microsoft Silverlight, or other rich-presentation infrastructures. One common characteristic of all these technologies is that they impose a change with respect to the way Web applications are built.

Figure 1 compares the flow of content in traditional and modern Web applications. Traditional data-driven Web applications typically consist of a set of server-side pages or scripts that run when a request arrives; during execution they execute a few queries against a database and then render HTML that contains both presentation information and data embedded in it.

An AJAX-based or Silverlight-based application does not follow that interaction model. Presentation information is shipped to the Web browser along with code to drive the user interface, but without actual data. That presentation information is typically a combination of HTML, Cascading Style Sheets (CSS), and JavaScript for AJAX

Figure 1: Traditional flow of content (left) and how it changes in modern Web applications (right)


Web Server
Client
Presentation + Data (for example, HTML with embedded data)
Database Server
Data
Web Server
Client
Presentation (for example, HTML)
Database Server

applications and Extensible Application Markup Language (XAML)/ DLLs for Silverlight. Once the code is up and running in the client, the initial user interface is presented and data is retrieved as the user interacts with the interface.

Interestingly, this new round of technologies is acting as a forcing function to push application organization toward a goal that is common in many application architectures: strict separation of data and presentation.

Serving user-interface elements is relatively straightforward from the server perspective. Most of the time these are simple file resources on the server, such as HTML or CSS files, media files. Serving data is another story. Until now, interaction with data was something that happened between the Web server and the database server; there was no need to expose entry points accessible from code running across the Web in a Web browser or some other software agent. This is where Project Astoria kicks in.

Flexible Data Interfaces

There are a number of ways to expose data to clients that will consume it from across the Web. One approach that is enabled by existing technologies is to use an approach similar to Remote Procedure Call (RPC), where "functions" are exposed through an interface such as Web services (for example, a SOAP-based interface or a simple URI-based convention for invoking methods and passing parameters). Microsoft Visual Studio has a mature set of tools that makes it straightforward to both create and consume interfaces built this way. The ASP.NET AJAX toolkit takes this to the next level by enabling Visual Studio-created Web services to work with AJAX clients.

The main issue with this approach is flexibility. If interaction with data only happens through fixed, predefined entry points then every new scenario or every variation of existing ones with slightly different data will typically require the creation of new entry points. While this level of control is occasionally desired, in many cases, more flexibility would increase development productivity and contribute to a more dynamic application.

Project Astoria introduces an alternative to the RPC approach that is based on the simple semantics of HTTP. Astoria takes a schema definition that describes each one of the entities that your application deals with, along with the associations between entities, and exposes them over an HTTP interface. Each entity is addressable with a Uniform Resource Identifier (URI) and a URI convention allows applications to traverse associations between entities, search entities, and perform other commonly needed operations on data.

The schema definition used by Astoria is an Entity Data Model (EDM) schema, which is supported directly by the ADO.NET Entity Framework. The Entity Framework also includes a powerful mapping engine that allows developers to map the EDM schema to a relational database for actual storage.

In order to show interaction with Astoria services, I will use an example based on the well-known Northwind sample database. A data service on top of Northwind can be set up by creating an ASP.NET application, importing the Northwind schema from the database into an EDM schema using the EDM wizard, and then creating an Astoria data service pointing to that EDM schema.

Detailed steps for creating Astoria data services, as well as extensive documentation on the Astoria URI and payload formats, can be found in the "Using Microsoft Codename Astoria" document available on the Astoria Web site at http://astoria.mslivelabs.com.

With the service up and running, URIs can be used to browse the resources exposed by the data interface. The following examples use the experimental Astoria online service that hosts a few read-only data services. For instance:

```
http://astoria.sandbox.live.com/northwind/northwind.
rse/Customers
```

would return all of the resources in the Customers resource container. In the default XML format, it would look like this:

```
<DataService xml:base="http://astoria.sandbox.live.
com/northwind/northwind.rse">
 <Customers>
  <Customer uri="Customers[ALFKI]">
   <CustomerID>ALFKI</CustomerID>
   <CompanyName>Alfreds Futterkiste</CompanyName>
   <ContactName>Maria Anders</ContactName>
   <ContactTitle>Sales Representative</ContactTitle>
   <Address>Obere Str. 57</Address>
   <City>Berlin</City>
   <Region />
   <PostalCode>12209</PostalCode>
   <Country>Germany</Country>
   <Phone>030-0074321</Phone>
   <Fax>030-0076545</Fax>
   <Orders href="Customers[ALFKI]/Orders" />
  </Customer>
  <Customer uri="Customers[ANATR]">
     ...properties...
  </Customer>
  ...more customer entries...
 </Customers>
</DataService>
```

It is also possible to point to a particular entity by using its keys:

```
http://astoria.sandbox.live.com/northwind/northwind.
rse/Customers[ALFKI]
```

would return a particular Customer resource; again, using the XML format in the example:

```
<DataService xml:base="http://astoria.sandbox.live.
com/northwind/northwind.rse">
 <Customers>
  <Customer uri="Customers[ALFKI]">
   <CustomerID>ALFKI</CustomerID>
   <CompanyName>Alfreds Futterkiste</CompanyName>
   <ContactName>Maria Anders</ContactName>
   <ContactTitle>Sales Representative</ContactTitle>
   <Address>Obere Str. 57</Address>
   <City>Berlin</City>
   <Region />
   <PostalCode>12209</PostalCode>
   <Country>Germany</Country>
   <Phone>030-0074321</Phone>
   <Fax>030-0076545</Fax>
   <Orders href="Customers[ALFKI]/Orders" />
  </Customer>
 </Customers>
</DataService>
```

When a URI points to a specific resource such as the one just discussed, it is possible not only to retrieve the resource using an HTTP GET verb, but also to update it, using HTTP PUT, or delete it, using HTTP DELETE.

Since the schema description provided to Astoria includes associations between entities, those can also be leveraged in the HTTP interface. Continuing with the example, if each Customer resource is associated with a set of Sales Order resources, then the following URI represents the set of Sales Orders related to a particular Customer:

```
http://astoria.sandbox.live.com/northwind/northwind.
rse/Customers[ALFKI]/Orders
```

In addition to being able to point to specific resources and list resources in a container, it is also possible to filter, sort, and page over data to facilitate the creation of user interfaces on top of the data service. For example, to list all the Customer resources in the city of London, sorted by contact name, an application can use this URI:

```
http://astoria.sandbox.live.com/northwind/northwind.
rse/Customers[City eq 'London']?$orderby=ContactName
```

The actual format of Astoria URIs is still subject to change, but the semantics and capabilities should remain relatively stable.

In all cases, data is exchanged in simple formats such as XML or JSON. The actual format can be controlled by the client-agent using regular HTTP content type negotiation. Data is encoded as resources that simply map properties in EDM entities into XML elements or JSON properties, and they are hyperlinked to other resources that they have associations with. For example, the URI from the previous example:

```
http://astoria.sandbox.live.com/northwind/northwind.
rse/Customers[ALFKI]
```

would result in the following response (in XML):

```
<DataService xml:base="http://astoria.sandbox.live.
com/northwind/northwind.rse">
 <Customers>
  <Customer uri="Customers[ALFKI]">
   <CustomerID>ALFKI</CustomerID>
   <CompanyName>Alfreds Futterkiste</CompanyName>
   <ContactName>Maria Anders</ContactName>
   <ContactTitle>Sales Representative</ContactTitle>
   <Address>Obere Str. 57</Address>
   <City>Berlin</City>
   <Region />
   <PostalCode>12209</PostalCode>
   <Country>Germany</Country>
   <Phone>030-0074321</Phone>
   <Fax>030-0076545</Fax>
   <Orders href="Customers[ALFKI]/Orders" />
  </Customer>
 </Customers>
</DataService>
```

You can see that the response includes simple properties and hyperlinks to other resources ("Orders" in the example). Every resource also includes a URI that represents the canonical location for it (in the URI attribute of the Customer element above).

Interacting with Data in the Presentation Layer

There are a number of options for interacting with data sources from the presentation layer. The first aspect that will scope the available options for a given scenario is the nature of the client (for example, browser versus rich client).

Since the Astoria interface is just plain HTTP, pretty much every environment with an HTTP client library can be used to consume data services. The interface has been specifically designed to be easy to use at the HTTP level; the URI patterns are simple and human-readable, and the payload formats use JSON or a subset of XML that keeps it straightforward.

For .NET applications, the Astoria toolkit includes a client library that runs in the .NET Framework environment and presents results coming from Astoria services as .NET objects; not only is that easier for developers to use within the codebase of the client application, but it also integrates well with components that already operate on top of regular .NET objects. The library provides rich services such as graph management, change tracking, and handling of updates. (See Example 1.)

The Astoria client library runs both on the .NET Framework and inside Microsoft Silverlight. This enables the creation of desktop applications and Silverlight-based Web applications using the same API, just by referencing the corresponding Astoria assembly for the target environment.

In the case of AJAX-style Web applications, most AJAX frameworks include easy-to-use wrappers for HTTP access to external resources. Those

Example 1: Accessing an Astoria service from .NET code using the Astoria client library

```
WebDataContext ctx = new WebDataContext(
                  "http://astoria.sandbox.live.com/
Northwind/Northwind.rse");

WebDataQuery<Customer> customers = ctx.
CreateQuery<Customer>("/Customers?$orderby=CompanyNa
me");

foreach(Customer c in customers) {
    Console.WriteLine(c.CompanyName);
}
```

Example 2: Accessing an Astoria service from an AJAX application (example uses ASP.NET AJAX library)

```
function ListCustomers() {
    var req = new Sys.Net.WebRequest();
    req.set_url("Northwind.svc/Customers?$orderby=Co
mpanyName");
    req.get_headers()["Accept"] = "application/json";
    req.add_completed(onCustomersCompleted);
    req.invoke();
}

function onCustomersCompleted(response) {
    if(response.get_statusCode() != 200)
        alert("Error retrieving customers: " +
response.get_responseData());
    else {
        var res = response.get_object();

        var html = "<ul>";
        for(var i = 0; i < res.length; i++) {
            html += "<li>" + res[i].CompanyName + "</
li>";
        }
        html += "</ul>";

        // "Customers" is a DIV element defined in
the HTML document
        $get("Customers").innerHTML = html;
    }
}
```

wrappers even support materializing the response into JavaScript objects if you indicate that the response will be in JSON format. (See Example 2.)

Independent from the kind of application, they will typically run in environments with relatively high latency between the client and the data service, so the use of typical asynchronous execution techniques should be the norm. The client API has built-in support for asynchronous request execution. For the case of AJAX applications, the XMLHTTP interface, along with most wrappers, provides support for asynchronously submitting requests. (See Example 3.)

Introducing Business Logic

Astoria enables developers to simply point to a database or a pre-built EDM schema and it will automatically generate an HTTP view of it. While this is great for the initial iterations of an application, this wide-open interface to the data will often not be appropriate for production applications.

In most databases, there is a relatively clear split between two kinds of data (most typically, two kinds of tables): There is a part of the data that has enough implied semantics in itself; this is true for simpler concepts such as "product category." In those cases, business logic is usually thin or inexistent, and the direct interface to the data is good enough. The other part of the data only makes sense with some

Example 3: Using the asynchronous API in the Astoria client for .NET

```
WebDataContext ctx = new WebDataContext(
                         "http://astoria.sandbox.
live.com/Northwind/Northwind.rse");

WebDataQuery<Customer> q = ctx.
CreateQuery<Customer>("/Customers[City eq 'London']");

q.BeginExecute(
    delegate(IAsyncResult ar)
    {
        foreach (Customer c in q.EndExecute(ar)) {
            // process each customer
        }
    },
    null);
```

business logic around it; for example, the data that is shown needs to be restricted based on the context, or modifications need to pass external validations, or when a given value is changed, another side-effect needs to take place, and so on.

To address the requirement of being able to introduce business logic that is tightly bound to certain pieces of data, Astoria supports two customization mechanisms: service operations and interceptors.

A default Astoria service consists entirely of resource containers that are the entry points to the resource graph, such as /Customers or /Products. In addition to those, developers can define service operations that encapsulate both business logic and queries. For example, in a given application, it may not be desirable to list all customers; instead, the application could provide an entry point to list "my customers" and even then there could be a requirement where clients retrieve their customers for a given city at a time. The developer can define a "MyCustomersByCity" service operation that obtains the users' identity from the context (from ASP.NET's HttpContext.User property, for example) and can then formulate a query that factors in both the user identity and the city name that is passed as an argument. For example:

```
[WebGet]
public static IQueryable<Customer> CustomersByCity(No
rthwindEntities db, string city)
{
    if (city == null || city.Length < 3) throw new
Exception("bad city");

    var q = db.Customers.Where("it.City = @city",
                                new
ObjectParameter("city", city));

     // add user-based filter condition to q

    return q;
}
```

would then be callable with a URI following this pattern:

```
/MyCustomersByCity?city=Seattle
```

An interesting feature of Astoria is that service operations can opt for returning a query instead of the actual results, as shown in the previous example. When a query object is returned, the rest of the URI pattern can still be used; so for instance, client could still add an "orderby" option to the URI:

```
/MyCustomersByCity?city=Seattle&$orderby=CompanyName
```

The service operation can also contain code to perform validations, log activity, or any other need. This provides a good middle-ground between strict RPC, which makes building flexible UIs hard, and wide-open data interfaces that do not allow for control of the data that is flowing through the system.

For scenarios where preserving the resource-centric interface is desired, interceptors can be used. An interceptor is a method that is called whenever a certain action happens on a resource within a given resource container. For example, a developer could register an interceptor to be called whenever a change (POST/PUT/DELETE) is made to the "Products" resource container. The interceptor can perform validations, modify values, and even choose to abort the request.

Extensibility and Alternate Data Sources

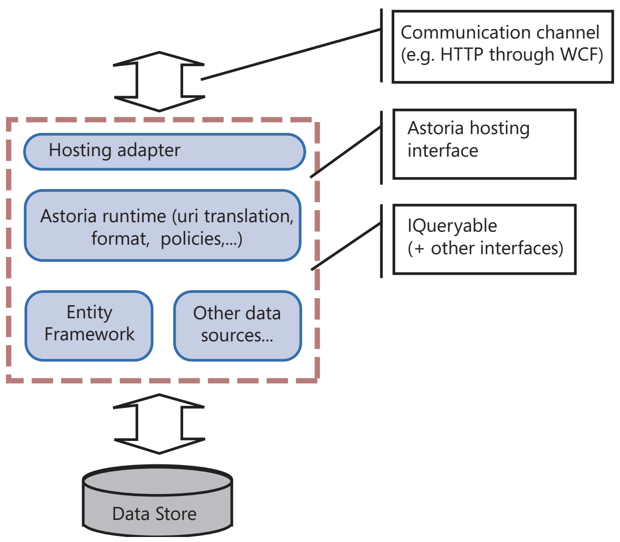
So far I have discussed Astoria in the context of EDM and the ADO.NET Entity Framework. When using Astoria for surfacing data in a database, this is most likely the best choice; however, not all data is in a database.

In the first public CTP of Astoria, we have only targeted the Entity Framework. As we iterate on the design of the product, we are changing that to provide more options. Specifically, in order to enable scenarios where you want to expose data sources that are not databases, we will support using any LINQ-enabled data source to be exposed through the HTTP interface.

LINQ defines a general interface called IQueryable that allows consumers to dynamically compose queries without having to know any details about the nature of the target for the query. It is up to the actual implementation of IQueryable in each source to interpret or translate queries appropriately. This allows the Astoria runtime to take a "base query" and compose it with operations such as sorting and paging. (See Figure 2.)

With this extensibility point in place, developers will be able to bring a broad set of data sources into the picture and expose them through the HTTP interface. This ranges from access to specialized data stores to using LINQ-based access libraries for online services (for example, there are informal implementations of LINQ to Amazon and LINQ to Flickr that provide limited query capabilities to those Internet sites).

Figure 2: Astoria architecture diagram illustrating IQueryable-based layering


Communication channel (e.g. HTTP through WCF)
Astoria hosting interface
IQueryable (+ other interfaces)
Hosting adapter
Astoria runtime (uri translation, format, policies,...)
Entity Framework
Other data sources...
Data Store

Deployment Scenarios: Applications and Services

A typical data-driven Web application today will have its own database in addition to one or more Web servers. For enterprise applications, these servers are part of the IT infrastructure and for applications hosted in ISPs, most ISPs provide database services.

Along the same lines, you can expect many applications that use Astoria as their data services and are built around AJAX or Silverlight to still have their own database. In those scenarios, the Astoria data services will be part of the Web application itself, and will be deployed together with the rest of the application components. This is one of the scenarios we target with Astoria, but it is not the only one we envision.

Another way of looking at Astoria is as a technology for building data services for other systems to consume. Data providers can set up Astoria servers that other applications can interact with, both consuming and updating data as required and as allowed by the security policies. The provider of the data could be either the same as the owner of the application that consumes the data or it can be a service for others to consume.

Yet another way of looking at Astoria is as a general-purpose service for data storage. To explore this idea we have set up an experimental online service that hosts several sample data sets, including the Northwind and AdventureWorks sample databases, a subset of the Microsoft Encarta articles and a snapshot of data that supports the Microsoft TagSpace social bookmarking site. Turning this into a real-world service requires technology well beyond just the HTTP interface and patterns, and that is a space outside of the scope of the Astoria project, but we still find the experimental service valuable as a learning tool, to see what applications developers would build on top of it.

Security

Exposing a Web-facing data interface requires careful thinking around securing access to make sure only the data that is meant to be accessible is effectively accessible over the HTTP interface. This involves both an authentication infrastructure and proper authorization policies.

While a lot of the design points in Astoria equally apply to Astoria as a component of an application and to Astoria as a service, authentication is one of the areas where this is not the case.

When using Astoria data services as part of a custom Web application, authentication typically applies to all resources within a given boundary, including access to data. Users would authenticate once with the Web

REST and Astoria

The term REST stands for Representational State Transfer and was initially introduced by Roy Fielding. It refers loosely to an "architectural style" where systems present a very simple, resource oriented abstraction for application state, and a uniform interface to act on those resources. There are also aspects such as layering and caching that fit very naturally in the model and enable the creation of large, highly-scalable systems. The World Wide Web is often cited as an example of a REST application and the level of scalability that it enables.

I think Astoria can be considered a good REST citizen. Astoria turns entities/records into resources, and those resources are addressable through the URI space that the server presents. Every resource can be obtained and manipulated through the HTTP uniform interface, and the system allows for simple layering and caching through the traditional methods that are used by the WWW. Like every other piece of systems infrastructure, some pragmatisms need to be considered; when needed, Astoria does allow developers to leave the resource-only world and to introduce some RPC-ish constructs, which are sometimes requires to build real-world applications.

site and the system needs to be able to apply the credentials to the data service as well as to the rest of the application. Astoria looks into the ASP.NET API to find out whether a user is authenticated and to find out further details, so that an application that uses any authentication scheme properly integrated with ASP.NET will automatically work with Astoria.

Integration with ASP.NET authentication means that in typical cases common authentication-over-HTTP mechanisms will work. This includes "forms authentication", integrated authentication (useful inside corporate networks), and custom authentication schemes; it is even straightforward to roll a custom implementation of HTTP "Basic" authentication, which may be good enough when used over SSL connections, depending on the nature of the application.

For online services this becomes a much bigger challenge. Besides the actual technological question of how authentication happens, there are higher level differences that need to be addressed first: If an application and the data service are from different sources, is the data in the data service owned by the user of the application? If it is owned by the user, then the application should not have access to the user's credentials to the data service. What is required in this scenario is a scheme where the user authenticates with the data service independent of the application (which may require authentication as well). We are exploring this space as part of the Astoria design effort, and although we have a reasonable understanding of the scenarios, we have not yet designed concrete technology to support them.

Once the authentication scheme is in place, a proper authorization model is required. The currently released version of Astoria (May 2007 CTP) has an over-simplistic implementation, where authentication policies can be set at the resource container level (/Customers, for example); on each one of them the configuration can indicate whether authentication is required to read the resources in the container, and to write to them. This is not flexible enough for most applications, so clearly a more sophisticated scheme needs to be provided.

Closing Notes: Scope and Plans for Project Astoria

The manner in which applications are being written is changing. One of the key traits of emerging Web applications is the new ways they interact with their data. There is a clear opportunity here to introduce base technology to help the development community take on this new space.

With Project Astoria, we aim at enabling the use of data as a first-class construct on the Web and across the application stack. We want to provide the infrastructure for creating Web data services, and also contribute to the creation of an ecosystem where service providers and service consumers use a uniform interface for data. We would like to see UI controls vendors, client library writers, and other players leverage the power for reuse of the data interfaces to build better tools for Web application creation.

We are starting with this vision at home. Within Microsoft we are closely collaborating with various groups to explore different aspects where Astoria can play.

On the services side, we are working together with the Windows Live organization, the Web3S folks in particular (they are responsible for the data interfaces for Live properties), to explore a world where every data interface is a Web3S/Astoria interface and can be consumed by the various tools and controls out there.

On the tools side, the ASP.NET, WCF, and Astoria teams, as well as the various folks involved in Silverlight, are working together to provide a solid end-to-end story for Web application development, which includes first-class tools and libraries for interacting with data from Web applications and services.

Project Astoria moves fast and focuses on solving real-world challenges around the Web and data. We plan to go through the design process in a transparent way, so everyone can see what we are up to. It is an exciting time to be working on this space; I would encourage anyone that has an interest in the topic to check out the Astoria Team blog, follow our design discussions, and jump in whenever you have an opinion.

Resources

Astoria Team Blog:
http://blogs.msdn.com/astoriateam

Pablo's Blog:
http://blogs.msdn.com/pablo

Project Astoria
http://astoria.mslivelabs.com

About the Author

Pablo Castro is a technical lead in the SQL Server team. He has contributed extensively to several areas of SQL Server and the .NET Framework including SQL-CLR integration, type-system extensibility, the TDS client-server protocol, and the ADO.NET API. Pablo is currently involved with the development of the ADO.NET Entity Framework and also leads the Astoria project, looking at how to bring data and Web technologies together. Before joining Microsoft, Pablo worked in various companies on a broad set of topics that range from distributed inference systems for credit scoring/risk analysis to collaboration and groupware applications.

Implications of Software + Services Consumption for Enterprise IT

by Kevin Sangwell

Summary

Many articles in this issue use the term Software + Services (S+S) when referring to client (desktop, browser and device) and server based applications which consume one or more Internet (cloud) services. While this model shares some characteristics with Software as a Service (SaaS) the differences are significant for Enterprise IT.

This paper contrasts the challenges of adopting S+S versus SaaS; it will become clear that consumption of a well-defined external service is less challenging for enterprises than the consumption of a finished service.

Today, the majority of applications delivered as a service over the Internet (that is, SaaS) are aimed at the consumer and small business markets. The business monetization model used, whether subscription- or advertising-funded, is largely that of the Long Tail; selling a little of something to many, many customers through a scalable distribution channel, as described by Chris Anderson (see Resources).

However, enterprise demands are significantly different from the demands of these consumer and small business segments, so certain assumptions supporting Long Tail economics and service delivery (and consumption) just do not apply in an enterprise context. For example, consumers don't have to worry about compliance and Enterprise Application Integration (EAI) and all that is implied by it is largely irrelevant to small businesses.

Thus considering software services from an enterprise perspective raises a number of questions. Who owns the data? What is the Service Level Agreement (SLA)? Can internal identities be extended outside the firewall to access cloud services? Are there regulatory implications?

Background

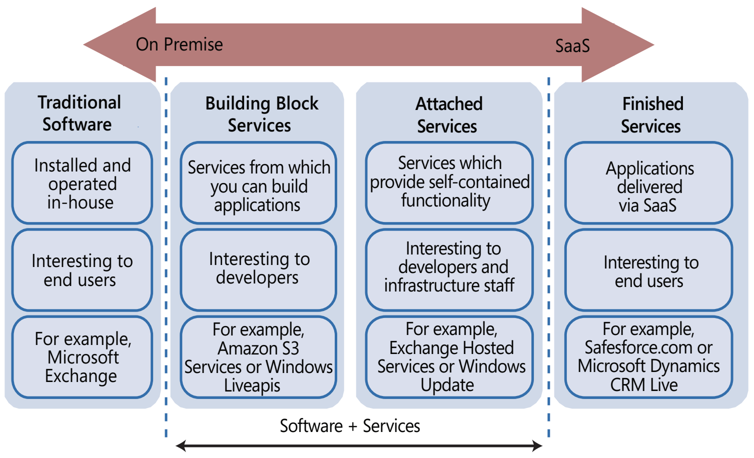
Roughly 70 percent of IT budgets go to maintain existing systems leaving around 30 percent for new solutions. While the cost of hardware and software is becoming less the cost of management and support is growing. Businesses are expecting more from IT than ever before, partly due to recovering confidence following the dot-com bust, as well as growing demand for Web 2.0-style capabilities inside the firewall.

At the same time, corporate IT is suffering a crisis of perception, as evidenced by the feedback over Nicholas Carr's 2003 essay, "IT Doesn't Matter" (see Resources). Business leaders are often frustrated that internal IT projects take months and significant investment to provide benefits that appear readily available on the Internet. Users experience search, collaboration, and publishing capabilities on the Internet that are far superior to many enterprises' internal capabilities. An increasing number of applications are being made available as services on the Internet, giving the business an alternative IT sourcing model.

In this paper I discuss the implications of consuming external software services on existing corporate IT infrastructure and operations and compare the challenges of the SaaS model with the S+S model when consuming any line-of-business application that has broad adoption across the company. I use the term "software services" to refer to the services in both SaaS and S+S models because we can consider software delivery as a continuum with traditional in-house hosted software, built or bought, at one extreme, and finished services delivered over the Internet (that is, SaaS) at the other. The hybrid, in-house software plus cloud services, spans the middle of the continuum (that is, S+S). Figure 1 illustrates this software delivery continuum. In this paper,

- *Traditional software* refers to applications installed in the infrastructure accessed exclusively by internal users.
- *Building block services* provide low-level capabilities that can be consumed by developers when building a composite application. These services exist in the cloud.

Figure 1: Software delivery continuum and software services taxonomy


On Premise
SaaS
Traditional Software
Installed and operated in-house
Interesting to end users
For example, Microsoft Exchange
Building Block Services
Services from which you can build applications
Interesting to developers
For example, Amazon S3 Services or Windows Liveapis
Attached Services
Services which provide self-contained functionality
Interesting to developers and infrastructure staff
For example, Exchange Hosted Services or Windows Update
Finished Services
Applications delivered via SaaS
Interesting to end users
For example, Safesforce.com or Microsoft Dynamics CRM Live
Software + Services

- *Attached services* provide a higher level of functionality compared with building block services. Applications leverage attached services to add functionality.
- *Finished services* are analogous to full-blown applications, delivered over the Internet using the SaaS model.
- *S+S* refers to the use of applications that consume attached services or one that is built with building block services.

When considering an IT infrastructure or application sourcing model, it is important to understand the business objectives. For example, outsourcing is driven by the need for cost efficiency, transferring the cost and risk of delivery of an existing, mature application to another party in exchange for contracted payments. In contrast, adoption of a software service satisfies a business need, such as more effective customer management (in the case of CRM). From a business manager's perspective, SaaS appears to be the best of both worlds: the business benefit is realised at a cost proportional to use (or even free), with no additional or up-front capital investment in IT resources. Although the business is best placed to determine how well the service solves the business problem, unless IT is included in the discussion, many of the wider implications for enterprise IT—hidden costs of software services adoption—will be missed.

Software Services: New Integration Challenges

When adopting software services, one set of challenges becomes the responsibility of the provider — service delivery and service support. However, IT will face an additional and therefore new set of challenges in adopting the new model (Figure 2).

Ignoring these new challenges is not an option. As we will see, there may be direct and indirect costs, resource implications, and compliance issues. In other words, the adoption of a software service means a hybrid procurement/integration project for internal IT. Integration needs to be considered in three broad areas:

- **Identity and Access Management**
- **Data**
- **Operations**

Regulations and legal obligations should also be considered.

Identity and Access Management

Identity and access management is a perennial problem that affects many aspects of IT, from help desk costs through user productivity to data security. It's also one area where enterprise IT should provide a better user experience compared to the Internet—yet most enterprises have dozens of user directories. Analyst organizations frequently state that on average password resets account for 30 percent of help desk calls.

The addition of a finished service, with its corresponding external directory, requires extensions to the provisioning and deprovisioning process, *even if this is a human process*. For example, when an employee leaves, many organizations struggle to deprovision or disable internal accounts in a timely manner; the risk of exposure in the case of an external application or service is significant because there is no corporate firewall preventing the user from accessing the application and its data.

Neither Active Directory (AD), nor metadirectory products such as Microsoft Identity Lifecycle Manager solve this particular problem. AD is proprietary and its trust model is not sufficiently granular, and metadirectories are not widely deployed and don't operate in real time. Something standards-based, such as Federation offers a set of capabilities that make it a particularly good fit for integration with an external application or service. It is loosely coupled yet operates in real time rather than via a schedule, simplifying provisioning and deprovisioning. Federation trust relationships have a high level of granularity, allowing the consumer organization to expose only a subset of their directory (based on rules); and real-time mapping of attributes to "claims," reducing the need for internal directory changes. (See Figure 3; for more on Federation and ADFS, see Resources.)

In contrast to SaaS, S+S applications may have a back-end service running inside the firewall, in which case a single enterprise or proxy identity could be passed to the service in the cloud. Identity integration is a common capability of many enterprise applications, so the back-end service may integrate with Active Directory or generic LDAP directories out of the box.

Access Control

Authorization and access management is another aspect that needs consideration. Many applications provide capabilities that differ according to the user. For example, an expenses application may allow a manager to authorize claims up to a value of $4,000; claims above $4,000 may need director approval. Today, many applications inside the corporate firewall set permissions against individual users; in effect, the application contains a mapping between the user and their

Figure 2: Software services consumption adds new challenges

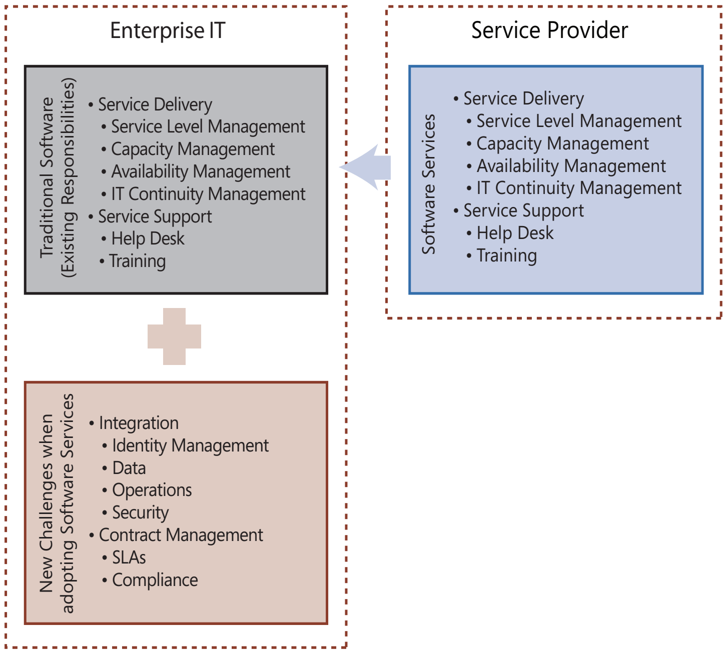
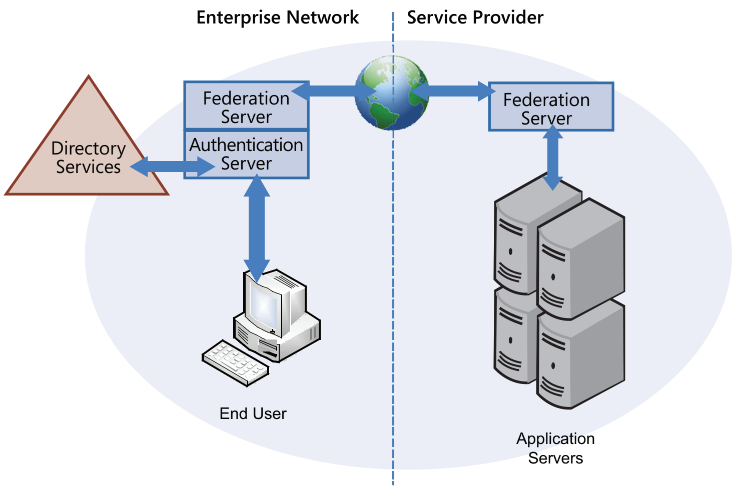


Figure 3: Federation providing identity integration



authorization. However, numerous organizations have started to move away from user-based authorization to role-based authorization. The benefits are clear: lower administration costs, consistent authorization within a role, and transparent compliance.

When a finished service has multiple levels of authorization, there are two options: Task someone inside your organization with manually mapping users to authorization levels; or extend the internal role-based model to the external application. The former often falls to the help desk; a hidden cost of adopting finished services. The latter, mapping an internal role to the external application, could be achieved through Federation; for example, membership of an Active Directory group could be mapped to a name/value pair in a cookie that is used by the external application.

Authorization in building block or attached services (the hybrid Software + Services) could follow the Federation model, and indeed there are advantages to doing so: Integrity is tightly managed at the service provider, which may help achieve compliance. The more flexible model of S+S means that authorization could be carried out by a local server before the request is made to the external service. Having local control over and enforcement of policy means the organization can react to business changes more quickly. It also simplifies integration; the enterprise can make changes to the configuration of the on-premise part of the vendor application to suit its environment.

Summary of Identity and Access Management Implications

- If the external service depends on user identity (highly likely for SaaS, possible for S+S), your provisioning and deprovisioning processes need to be extended. Integration could be via technology or a manual process, both of which have cost implications.
- Service provider user account policies need to be evaluated against your internal policies (for example, password complexity, lock-outs, and so on).

Data

A line-of-business application is unlikely to exist as an island, even when it is externally sourced. A good example is payroll. The payroll service provider needs raw data: employee name, pay amount and so on, to process payroll monthly. In this example, the data could be supplied via a simple extract of the relevant information from the internal HR system. Clearly, e-mailing an XLS or CSV file to the payroll provider is unlikely to provide the level of security/confidentiality needed, so another form of data exchange is required, and IT will be expected to provide this. In other words, IT faces an Enterprise Application Integration (EAI) project.

From an EAI perspective, building block and attached services should be straightforward to integrate; after all, they're designed to be consumed by developers as extensions of a local application. One example of an attached service is Exchange Hosted Services for Filtering (for more information on Exchange Hosted Services, see Resources). Integration in this situation is twofold:

1. DNS: MX record changed to point at the service provider (Microsoft in this case)
2. Firewall rules changed to allow inbound SMTP from only the service provider (which increases security).

As infrastructure is the focus of this article, I won't delve into further detail with respect to EAI. However, we will look at the infrastructure implications for data from a few other perspectives:

- Firewall rules and filters
- Encryption and signing
- User view

Firewall

To understand the firewall implications of consuming finished services, we need to investigate which internal applications will integrate with the software service and the form of integration. *Is it one internal application that needs to be integrated or several? What firewall rules need to be created to allow publishing and traffic flow? If the applications are exchanging XML, does this need to be validated at the firewall, and can the firewall provide this capability? Does the data need to integrate with some form of workflow, and if so, how does this workflow span the internal/external infrastructure?* If the integration occurs over HTTP (SOAP, for example), there may be few implications on the firewall beyond the creation of rules.

With building block and attached services, there's likely to be some local back-end infrastructure, which naturally becomes the focal point for data integration and security. Indeed, some SaaS vendors are realizing that enterprises will be more willing to subscribe when their data is stored inside the corporate firewall—so they're evolving their finished services to the S+S model by installing an appliance inside customer data centers.

Encryption and signing

The most effective way of exchanging encrypted data across the Internet is to adopt certificates from a public certificate authority. If the certificates get installed on clients, a Public Key Infrastructure (PKI) project is required, with all its implications, such as certificate life cycle management and publishing the certificate revocation list. Not a trivial undertaking, but once completed, it will enable other capabilities within the infrastructure, such as signed e-mail and Smart Card authentication.

Of course, with a local back-end infrastructure in S+S implementations, encryption and signing is likely to be far simpler (fewer end points).

User view
Another perspective on data is the view of the user. Rightly or wrongly, many business people want to continue working with the tools they are familiar with—the Microsoft Office applications. Consider the number of times a new application fails to reach its full potential because the business users insist on extracting the data and using Excel for day-to-day management. Rarely does this data get back into the application. A number of independent software vendors have started to build their application clients as Office Business Applications, essentially using Office as the application platform. This generally results in faster adoption and lower training overhead, but it does present IT with deployment and maintenance issues. Points to consider: *Does the software service provide all of the capabilities needed by the business, or will users need to extract the data to perform manipulation/analysis in a local tool/system? Will the adoption of the software service result in yet more departmental applications built in Access and Excel?*

Summary of Data Implications
- Analysis will be needed to determine data integration and ETL needs.
- Firewall rules may be needed to allow integration.
- The firewall may need updating to provide filtering for application data (for example, XML Schema validation).
- Purchase of certificates or implementation of PKI may be needed to support authentication, encryption and signing requirements.

Operations
The problem of operational integration when sourcing an application or service externally is somewhat incongruous: After all, a key benefit of being a consumer is that operations are the responsibility of the provider, yet internal operations and processes will be impacted by the external application or service in several areas, the most significant being help desk and user training.

Help desk
Many consumers have been frustrated by inefficient and confusing call centers. To avoid similar problems, internal help desk teams should be aware of new applications and services being integrated into IT operations. Enterprise users depend on numerous internal IT systems to access an external application: the network, DNS, proxy servers. It's the responsibility of the corporate help desk to support these, not the service provider. Many organizations now realize the importance of simplifying the support process for the business, providing, for example, a single phone number and intranet site which connects them to the appropriate first-line team.

Training
User training is an important aspect of introducing any new application, irrespective of where it is hosted. An advantage of finished services is that the provider typically makes on-demand training available. However, the enterprise has little control over the quality of such training, and poor training increases the cost of support and reduces business productivity. The introduction of upgrades as part of the service, one of the benefits of subscribing to finished services, is another potential problem: If the enterprise does not have the ability to delay the implementation of the upgrade until all staff have received training, the internal help desk may have to handle a spike in support calls. A related issue is whether individual features of the finished service can be selectively disabled: If the capability is already provided in-house, IT needs to ensure that users are all using the internal capability.

"THE GREATER THE IMPORTANCE OF THE APPLICATION TO THE BUSINESS, THE MORE IMPLICATIONS YOU NEED TO CONSIDER. A LINE-OF-BUSINESS APPLICATION DELIVERED AS A FINISHED SERVICE IS A SIGNIFICANT INTEGRATION UNDERTAKING."

Deployment
If the finished service uses the browser as its client, the normal compatibility concerns apply: browser version and security settings, plus installed plug-ins and their versions. With a traditional application, the enterprise can determine when to upgrade to the new version, which is critically important if the new version requires the latest browser. When the application is external, this option may not be available.

If the service's client is not browser-based, internal IT will be responsible for deployment and its implications (compatibility testing, deployment planning, rollout, and so on).

If the service is an attached or building block service, there will be a need for deployment of either the back-end infrastructure in the enterprise data center or the client, or both. For back-end deployment, there are common nonfunctional challenges: *What server, network, and storage capacity is needed to meet the load? Can shared services such as, existing SQL server or web server farms be used? How are resilience and disaster recovery provided? Can it run in multiple data centers?* The answers will be more dependent on the local components of the application than on the attached service in the cloud: In other words, it can be approached largely like a normal back-end deployment.

Provider operations / business continuity
It is tempting to focus purely on the contents of the SLA when it comes to selecting, monitoring, and evaluating a service provider. How the SLA will be achieved is an important consideration: An SLA which states that service will be restored within 24 hours of a disaster seems good on the surface; however, the definition of a disaster and form of restoration are critically important. To the service consumer, disaster may be the accidental deletion of records from the application, but the service provider will likely have a different view. Taking this example further, restoring application data is a complicated process for many business applications. For instance, restoring a single Exchange mailbox or message or a single Sharepoint site or document was a significant challenge until those applications matured. Now apply that challenge to a multitenant SaaS application—even if granular restoration is possible, the provider will be reluctant to do it due to operational costs.

Another concern several architects have expressed to me is the risk of the service provider going out of business, or withholding data to prevent migration to a competitor. Placing the application code in escrow is a step in the right direction, but isn't really sufficient. Assuming an enterprise consumer could get its data, rebuilding the application in their data center without install instructions or access to the developers may not be feasible. There is no solution to this yet; it's a question of trust that the provider will do the right thing if the worst happens, and confidence that they have good business and management skills.

Reporting
As with any SLA, business and IT groups should review reports on performance and investigate where SLAs have not been met.

Summary of Operations Implications

- Help desk procedures need to be updated to perform first-line troubleshooting for new application & escalation processes need to be defined with the service provider.
- Review internal help desk SLAs to ensure they can still be met when depending on the service provider for escalated support.

Regulations and Legal Obligations

Proving compliance with regulations and legal obligations will have an impact on the infrastructure, and ensuring compliance for a business process that spans internal systems and external services can be especially challenging. It also presents a potential solution with respect to compliance: If the application or service is industry-aligned, there is a good chance it will be compliant with the relevant regulations in major markets; this may not be the case where the service is generic. Even in situations where compliance is a selling point of the service, an enterprise's internal security policies or region-specific laws, such as the European Parliament Data Protection Laws, may be incompatible with the service provider policies.

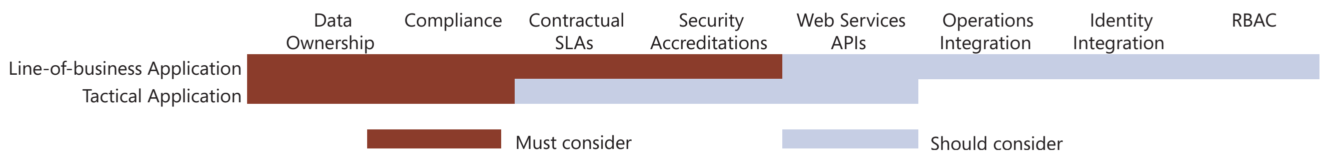
Data ownership
Clearly, an enterprise wants to retain ownership of its business data at all times; the contract should state this explicitly. Furthermore, it may be prudent to verify that data can be extracted on-demand.

Privacy
Enterprises should carefully evaluate a provider's privacy policies and terms of use to ensure that its data will be kept private and not used for marketing or sold to other parties, especially important where the finished services is supported by advertising. Privacy seal programs can be helpful in determining the trustworthiness of a provider; providers who are TRUSTe licensees, for example, will have a published policy and

Table 1: Summary of recommendations

Identity and Access Management	• Consider placing Federation support onto your provider selection criteria. • Investigate role-based administration as a way to reduce complexity and cost. • The business case for integration should be built on improved security and achieving compliance above cost reduction. • Start ramping up on identity technologies such as federation and CardSpace. • Produce an architecture to simplify identity integration of future software services.
Data Operations	• Include requirements for open standards (for example, WS-*) in the provider selection criteria. • Instigate lightweight monitoring of the service to support SLA measurement • If data is resident at the provider, determine whether brick-level restores are needed/possible. • Evaluate the use of formal operational frameworks (for example, MOF/ITIL) in the provider selection criteria. • Ensure you clearly understand the operational-related charging structure (for example, are there additional charges for restores, reports, and so on). • Evaluate training made available by the provider. Is it of good quality and available in all the spoken languages you need? How quickly is the material updated when a new feature is released? • If client deployment is needed, factor the provider release schedule (patches and new versions) into exiting desktop engineering processes and plans. • Evaluate deployment needs, and their dependencies (for example, desktop application compatibility testing, data center space, purchase of integration products). • Get your developers to evaluate the documentation made available by the provider. Is it of good quality and available in all the spoken languages you need? • Review provider resilience and disaster recovery policies/procedures.
Regulations / Legal	• Determine whether compliance extends to the provider and if so, work out what reports/policies/accreditations are needed to prove compliance. • Consider automating the creation of hybrid internal/provider compliance reports. This may mean the service provider exposing reporting APIs rather than providing a static report. • If data is resident at the service provider, review the contract to ensure you retain data ownership. • Extend your data Integration Architecture to support external service integration. Add security accreditations to the provider selection criteria.
General	• Create an Infrastructure Integration Architecture to provide a framework for software service integration.

Figure 4: Considerations heatmap



have been independently audited to ensure compliance with a set of privacy principles.

Summary of Legal Implications

- Compliance may extend to service provider, how do you still prove compliance?
- Compliance reports may have a cost associated with them.

Conclusion

The SaaS market is currently dominated with offerings aimed at consumers and small businesses, as this market segment benefits from the delivery model without the need for integration. Consuming a line-of-business application from these providers is risky for any enterprise, as many of the integration points discussed in this paper will not be addressed.

The greater the importance of the application to the business, the more implications you need to consider. A line-of-business application delivered as a finished service is a significant integration undertaking compared to the low effort associated with a tactical application where the primary concern is contractual issues such as data ownership. This relationship is represented in the heat map shown in Figure 4.

Today, enterprise IT departments are far more experienced and confident of consuming building block or attached services than full applications. Be it a data feed for a rich application like Reuters 3000, or an infrastructure service such as spam-filtering, this model is well understood.

As the SaaS delivery model matures and gains more widespread adoption, it is natural for more enterprise demands such as integration to be catered for, and the range of capabilities enterprises are prepared to source as services will increase. The result will be software plus services applications; a natural balance of on-premise software and cloud services.

As Gianpaolo Carraro and Fred Chong state in their article "SaaS: An Enterprise Perspective," SaaS and S+S are additional tools that savvy CIOs can use to provide better value to the business. Rather than feel threatened, IT managers should view SaaS and S+S for what they are: alternative sourcing models for business benefit, and a different architectural approach to building solutions.

Handled correctly, software services will help change the business' perception of IT.

Resources

European Parliament Data Protection Laws
http://ec.europa.eu/justice_home/fsj/privacy/index_en.htm

Exchange Hosted Services
http://www.microsoft.com/exchange/services/default.mspx

"IT Doesn't Matter," Nicholas G. Carr, Harvard Business Review, May 2003
http://harvardbusinessonline.hbsp.harvard.edu/b01/en/common/item_detail.jhtml?id=R0305B

The Long Tail: Why the Future of Business Is Selling Less of More, Chris Anderson (Hyperion, 2006)

Microsoft Privacy Guidelines for Developing Software Products and Services
http://www.microsoft.com/downloads/details.aspx?FamilyID=c48cf80f-6e87-48f5-83ec-a18d1ad2fc1f&displaylang=en

Microsoft Regulatory Compliance Planning Guide (although this guide is focused on internal IT, it can also be useful when evaluating an external provider)
http://www.microsoft.com/technet/security/guidance/complianceandpolicies/compliance/rcguide/default.mspx?mfr=true

WS-Federation is a draft OASIS standard that has been adopted by several vendors including Microsoft, IBM, RSA, BEA, and VeriSign. Microsoft's WS-Federation support takes the form of Active Directory Federation Services (ADFS), a component of Windows Server 2003 R2.

- OASIS
  http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsfed
- Active Directory Federation Services (ADFS)
  http://www.microsoft.com/WindowsServer2003/R2/Identity_Management/ADFSwhitepaper.mspx

About the Author

Kevin Sangwell is an infrastructure architect in the Microsoft Developer and Platform Group. He has held a number of technical and leadership roles in the IT industry for more than 16 years, including five years as a principal consultant in Microsoft Consulting Services. Kevin has lead the architecture and design for Enterprise and eCommerce infrastructures in the U.K. public and private sectors, including the distributed Microsoft infrastructure for a 120,000 user organization and an extranet application platform for 1.2 million educational users. As infrastructure architect, he provides advice and consulting to enterprise customers and presents at international events.

Enterprise Mashups

by Larry Clarkin and Josh Holmes

Summary

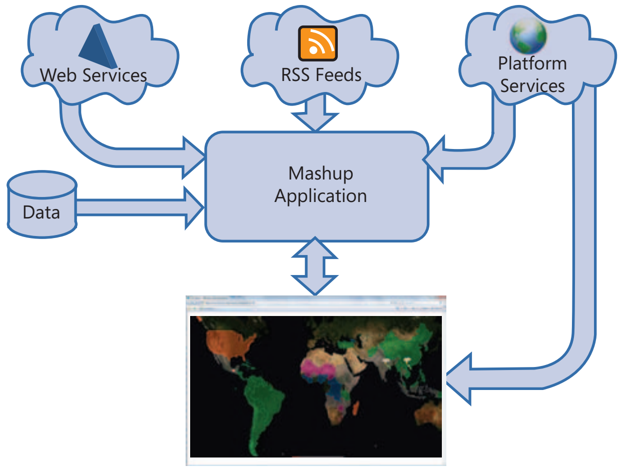
A mashup is a technique for building applications that combine data from multiple sources to create an integrated experience. Many mashups available today are hosted as sites on the Internet, providing visual representations of publically available data. This article describes the history and architecture of mashups, and explores how you can create mashups for use in your enterprise. We also impart some wisdom gained from projects with customers and systems integrators who have implemented mashups for the enterprise.

History of Mashups

Mashups have gained popularity within the last few years, swept along with the momentum around Web 2.0. Early mashups took data from sources such as Craigslist (http://www.craigslist.org) and combined them with mapping services or photo services to create visualizations of the data (for example, http://housingmaps.com). Many of these early mashups were consumer-focused, although recently there has started to be both interest and acceptance of mashups in the enterprise. Organizations are starting to realize that they can put their well-defined services that do discrete bits of business logic together with other existing services, internal or external to the organization, to provide new and interesting views on the data.

As techniques for creating mashups have matured, we are starting to see companies build business models around mashups. For the real estate market in the United States, both Redfin (http://www.redfin.com) and Zillow (http://www.zillow.com) use large amounts of public and private real estate data (from sources such as county record offices and the Multiple Listing Service), combined with internal "value added" services, the result of which is displayed to the user on a map (using Microsoft's Virtual Earth and Google Maps, respectively). There are many possible types of information that you could add to a real estate site; other similar listings, information on local schools, local hospitals, recent crime rates, classifieds for job postings and more.

Figure 1: The architecture of a typical mashup application



Client Application

Architecture of a Prototypical Mashup

Although there is a great variation in the user interface and the sources of data for many mashups, we can still derive common architectural patterns that they all share. For example, all mashups are RESTful in nature (they conform to the Representational State Transfer principles). Figure 1 shows an architectural rendering of a typical mashup.

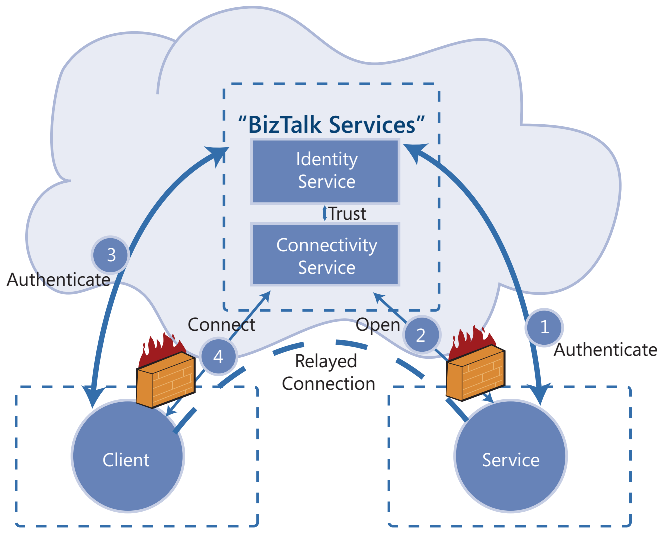
Data

The core element of any mashup is the data being aggregated and presented to the user. Although the above diagram depicts the source of the data as a database, the concept of a mashup does not require a database that is local to the mashup software or the client. The data can strictly come from Web services where data is serialized to XML or JSON (this is the most common pattern in Internet-based mashups). There are architectural trade-offs to be made from storing the primary data in a local data store and accessing the data with every request. As mashups move from being Internet-based applications to internal to the enterprise, they tend to depend less on external data stores.

RSS feeds

Use of RSS (Really Simple Syndication) feeds is a common source of primary or supplemental data for mashups. RSS feeds are easy to consume as they are XML documents, and many libraries exist to manipulate the feeds. The format and specification for RSS is well documented and understood with only a few variations from version to version. The extensibility of RSS is also well known, as demonstrated by the number of extensions in use today, such as adding attachments to the feeds, creative commons licensing information and location information.

Figure 2: Using BizTalk Services as a platform service to relay information



Web services
It is also common to include calls to Web services within mashups. It is common to see both WSDL-based Web services and REST-based Web services, with some services exposing both styles. Web services can be used to provide additional data or used to transform the data being mashed up. For a map-based mashup, the data may only contain street addresses and a call to a WSDL or REST-based Web service may be required to translate the street address to a Longitude / Latitude coordinate for the map.

Platform services
Figure 2 depicts a special class of services that are used to create mashups. We are calling these platform services because they provide functionality beyond the typical request/response model of traditional Web services. A typical example of this is the mapping service provided by Virtual Earth. Virtual Earth includes an entire array of server-side and client-side processing capabilities, as well as the "services in the cloud." We see the emergence of cloud-based building block services that begin to create value. For example, the Amazon S3 service offers storage "in the cloud"; this makes it easier to expose any static data by uploading it to a hosted storage provider. Microsoft's BizTalk Services is a platform service that provides a different capability – the ability to relay communications from the Internet across a corporate firewall, thus exposing internal services for consumption by business partners or third parties building their own enterprise mashups. Relayed communication as provided by BizTalk Services is also useful even within a single enterprise, with many business units or with numerous network segments. An Internet-based communications relay can eliminate physical network topology as a communications obstacle.

Mashup applications
Thus far we have identified many of the types of services that can be used to create mashups, but we have not addressed the importance of the software that creates and delivers the mashup experience. Think of the mashup application as a combination of middle-tier services and some lightweight business logic. For Internet-based mashups, the software is usually written using Web technologies (like PHP or ASP.NET), but we are starting to see the line between server processing and client application blur with the emergence of Rich Internet Applications (RIAs). RIAs are applications that run inside the browser with rich functionality similar to that of many desktop applications. These typically do not require a client side installation beyond a generic plug-in such as Adobe Flash or Microsoft Silverlight.

Client application
The client application is how the mashup is delivered and presented to the user. For public Internet mashups the most common client application is a Web browser that receives HTML and JavaScript from delivered from a Web server over HTTP. However, we have started to see mashups being delivered with RIA platforms as well. In this model, the client can provide more visual richness and can even provide some of the mashup processing on the client side.

Future Direction of Mashups
With early versions of mashups much of the implementation was very tedious and time consuming. Many of these used server-side processing (often with PHP or PERL) and tedious client side scripting in the form of JavaScript in order to create the mashup experience. It was common for the person creating the mashups to create custom code to parse the XML return sets that they received from their data sources.

As time has passed and the development process has matured, a lot of the tedious coding has been replaced by frameworks and better codification of standards. Custom scripts on the server side are starting to be replaced by standardized libraries that will automatically generate the required client-side script. We are also seeing standardization in the message formats. An example of this is the GeoRSS extension to the RSS standard that allows you to specify the Longitude and Latitude that is related to the items in the feed. All three major mapping service providers (Google, Microsoft, and Yahoo) support GeoRSS, which means mashups that use this RSS extension require almost no coding.

The creation of mashups was once only the domain of the developer, but there is a movement to put the ability to create mashups directly into the hands of the end customer. As the frameworks to create mashups are becoming simpler to use and the message formats are becoming more standardized, the next logical step is to build tools that can create mashups. Some of these tools will be targeted at the end consumer of the mashups. Pipes by Yahoo and Popfly by Microsoft are examples of frameworks and tools for allowing users to create their own mashups.

We are seeing an increase in the importance of common schema and metadata in the development of mashups. We described in the previous section how the common schema of RSS feeds makes it easy to incorporate them into mashups. The same principles will need to be applied to other types of data as well (as robust as RSS is, we cannot model all of our data into that format). We are already seeing the emergence of other standard schemas, such as KML (Keyhole Markup Language) to describe geospatial data. Even more interesting will be Microformats, a promising framework for delivering semantic meaning which can easily be read by software such as mashups.

Mashups in the Enterprise

A great way of exploring mashups in the enterprise is with an example. Let's imagine that you are an application architect for a call center system that receives calls about warranty and parts service. Using the phone number of the caller, we could display the records for that user, including a purchase history. This interesting application has already been implemented today in most call centers. But what if, in addition to looking up the customer information, we plotted the phone number on a map using a publically available service and also displayed a list of local service centers or parts suppliers for our products overlaid on the map? With this data in hand, we might be able to answer the customer's questions in seconds. What if we also looked up the current weather conditions in that area or the local sports teams and their recent games for a conversation starter or filler on long running calls?

Using this example, Figure 3 shows a mock customer service mashup that could be used by a representative when they receive a call. Combining data from public services (such as weather and news) with internal sources of data (a customer service alerts blog and a database of service locations), this application combines the mashup elements with the accessibility of a portal.

As shown in Figure 3, the embedded mashup puts a tremendous amount of ready information at the call service agent's finger tips – ranging from the top service items so that they can answer questions quickly to reverse lookups on the phone number to get conversation starters. This type of mashup is strictly internal but provides tremendous value on the initial call with any client.

There are several key elements to making this mashup successful. First and foremost, it is contextual to the task at hand. Second, it prioritizes the information in the order in which is most likely to be useful. You have to know who you are talking to, what products they have registered, what the top service items are for those products, and then start trying to answer questions that they may have such as where the local service centers are for the customer's location. Third, once we have the customer information and locale, each of the panes of content are standalone and do not require interaction with the other parts. This allows gathering and aggregation of the data for each of the parts to be performed asynchronously.

Realizing Return on Investment (ROI) is a common concern for Service-Oriented Architecture (SOA) deployments. Many organizations find it difficult to justify the upfront investment for creating services for different functionality when it would be easier to create a single application. Enterprise mashups are a great example of how an investment in SOA can provide great value. (See the sidebar "Mashups deliver ROI for IDV Solutions.")

Leverage the services you have built

Immediate return for a SOA can be realized when organizations start mixing and matching these services for new and exciting purposes. It can be exciting to start leveraging services or applications in ways that couldn't have been imagined when they were written. (See the sidebar, "Quicken Loans leverages mashups for fast results")

Leverage the services others have built

It's important to realize that you can't own all of the information in the world; and there's a fairly high return on investment when you can simply leverage someone else's hard work instead of us inventing that particular wheel.

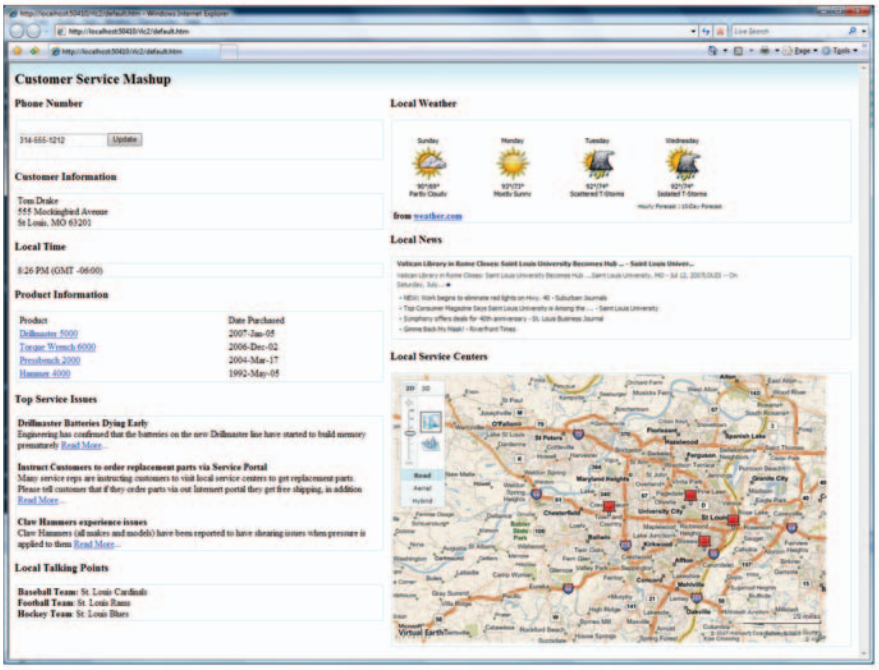
Build services that others may leverage

Another opportunity for enterprises, as mashups become more and more mainstream, is to build services that can easily be consumed by mashup applications. Going back to the example we cited, the customer service representative can make her customer happy by providing nearby store locations. But imagine if the stores themselves exposed their inventory and product availability to the mashup. Now the service representative can provide even more detailed and valuable information to the customer on the phone. That kind of service would be valuable for the customer, the call center, as well as the store itself.

Platform agile

As discussed earlier, many mashups have been created for delivery on the standards-based Web platform (HTML and JavaScript). This is not a limitation of mashups themselves, but merely the standard way of delivering applications on the Internet. As we see mashups move into the enterprise, we will see a growing number of mashups built on RIA platforms (Like Adobe's Flash and Microsoft's Silverlight) and even the emergence of full rich desktop mashups built on Windows Presentation Foundation. Full 3-D rendering on a rich client platform can increase the visual appeal of the mashup. Enterprise mashups can take full advantage of these richer platforms as many will have greater control of the desktops.

Figure 3: An example mashup for our call center



Customer Service Mashup

Phone Number

314-555-1212 Update

Customer Information

Tom Drake
555 Mockingbird Avenue
St Louis, MO 63201

Local Time

8:26 PM (GMT -06:00)

Product Information

Product	Date Purchased
Drillmaster 5000	2007-Jan-05
Torque Wrench 6000	2006-Dec-02
Pressbench 2000	2004-Mar-17
Hammer 4000	1992-May-05

Top Service Issues

Drillmaster Batteries Dying Early
Engineering has confirmed that the batteries on the new Drillmaster line have started to build memory prematurely Read More...

Instruct Customers to order replacement parts via Service Portal
Many service reps are instructing customers to visit local service centers to get replacement parts. Please tell customer that if they order parts via out Internet portal they get free shipping, in addition Read More...

Claw Hammers experience issues
Claw Hammers (all makes and models) have been reported to have shearing issues when pressure is applied to them Read More...

Local Talking Points

Baseball Team: St. Louis Cardinals
Football Team: St. Louis Rams
Hockey Team: St. Louis Blues

Local Weather

Sunday	Monday	Tuesday	Wednesday
90°/69° Partly Cloudy	93°/73° Mostly Sunny	92°/74° Scattered T-Storms	92°/74° Isolated T-Storms

from weather.com

Local News

Vatican Library in Rome Closes: Saint Louis University Becomes Hub ... - Saint Louis Univer...
Vatican Library in Rome Closes: Saint Louis University Becomes Hub ...Saint Louis University, MO - Jul 12, 2007 -- On Saturday, July ...
- NEW: Work begins to eliminate red lights on Hwy. 40 - Suburban Journals
- Top Consumer Magazine Says Saint Louis University is Among the ... - Saint Louis University
- Symphony offers deals for 40th anniversary - St. Louis Business Journal
- Gimme Back My Mask! - Riverfront Times

Local Service Centers

Quicken Loans leverages mashups for fast results

While writing this article, we spent some time with Keith Elder from Quicken Loans. Keith is the architect for an internal application that uses mashups and existing services to quickly deliver value to their customers. These services have been built over the last several years as part of the company's SOA initiative.

"We pull in information and composite our screens from all types of services. Our end users have no idea we are leveraging services throughout the enterprise to mine data, check address validation, view call activity, send faxes, and much more. To our team members the application is just one happy application that provides them functionality they need when they need."

He adds that Quicken Loans had many discussions early on about the granularity of the services they built. There are trade-offs to be made between single-purpose services and multipurpose services. They chose to go with single-purpose services that delivered very precise functionality. The trade-off that they made was that they might have to call multiple services as part of a single mashup.

Quick turnaround
If you are leveraging existing Web services and robust platform services, the development time of mashups can be measured in hours or days, rather than weeks or months. The quick turnaround time for mashups can change the way that IT departments interact with the users in their organizations. Joint development and multiple iterations of the applications being delivered in a short time can become more realistic goals.

Rich visualization of the data for your users
Perhaps the greatest benefit achieved from mashups is the visualization that they create for the users. Data that is visual in nature is far easier to understand and can create greater meaning for the customer. This is not limited to the mapping examples that we have discussed so far. Other techniques such as heat maps and tree maps can also be created as data visualizations in mashups.

Key Success Factors

Look for utilitarian services to consume (simple before complex)
The more things that a given service does, the harder it is to mash it in with other services. The right services do one thing and do that one thing well. For example, a service that returns people data with all of their addresses, loans, cars, trains, planes and automobiles, may be offering too much data. You might want to create a subset service that returns just name and address to mash with, as it's a waste of bandwidth and processing to pull data that you are not going to use.

Keep mashups read-only
Mashups are agile views into the data that they present. They are not an all-powerful editing surface capable of editing any and all data thrown at them. If someone wants to edit that data, they should go back to the application that created the data to do the editing. The mechanism for doing this should be obvious as well. This will dramatically reduce the complexity (and improve the agility) of your mashup applications.

Data freshness matters
Business decisions will be made based on enterprise mashups. This is dangerous when pulling data from multiple sources as the data might be stale. As with any report that you want to make decisions on, it's important to have the timestamp for when the data was last refreshed. That will greatly improve the clarity and usefulness of your mashed data, and the confidence with which the decisions can be made.

Don't try to solve enterprise data problems
As mashups move into the enterprise, it is inevitable that you will run into data fragmentation and normalization issues that are a result of years of legacy application development. For instance, to mashup sales by geographical area might seem like a logical mashup to create for your general sales manager. But what if the data is stored in three different systems by product line and each has different data structures and business rules governing them? Our advice is to find a better candidate for mashup technologies and let a more traditional project (such as a systems consolidation effort) resolve the more complex problems.

Understand authentication and authorization issues
Authentication and authorization can be huge impediments to doing a mashup of several different services if they all require authentication with different schemes. For example, issues can arise if you have one service that requires username and password and another that requires an X509 certificate. This is not insurmountable, but can be a large roadblock that has to be overcome. There are several strategies to attacking this problem. You can avoid services that require authentication or that require authentication via methods that you don't support. While you can get started this way, the reality is that there are services that you need to leverage that you can't get to without authentication. You could try to force others, or pay them, to offer the type of authentication that you are comfortable with. The reality, however, is that your application will most likely have to support service authentication using many different mechanisms, which all need to be taken into consideration.

Risks

When implementing mashups, four areas of risk should be considered:

Dependency on services
One of the major risks in creating enterprise mashups is when you create a dependency on services that are external to your company (such as the "services in the cloud"). The terms of the service agreements should be investigated before the dependency is created. For example, some services require that the software using the service be a public facing Internet site; this might occur when the service has an ad-based revenue model. The terms of service may also be subject to change in some cases in ways that could be detrimental to your use of that service. To mitigate this concern, look for service providers that have a model that fits your usage.

Loss of data fidelity
Loss of fidelity in the data being displayed is another key risk. As data is visualized, there is a tendency to make the data fit the confines of the presentation surface. There will be a natural tendency to not visualize small amounts of data or to group data into larger collections in order

Mashups deliver ROI for IDV Solutions

We recently spoke to Ian Clemens from IDV Solutions, a systems integrator who develops mashup applications for enterprise customers, about the speed with which mashups can be developed. "Our customers are seeing significant ROI from mashups, due not only to decreased implementation costs over traditional, custom software development, but also due to richer, more user-friendly applications that greatly simplify business workflows," said Ian.

to conserve space on the presentation surface. This has the potential to "warp" the end user's view of the data.

Politics

Politics can also be a hurdle when creating mashups. If you didn't create the service, then it might or might not do exactly what you wanted and it takes too long to get the originator of the service to change it to your needs. This Not-Invented-Here (NIH) mentality is fatal to mashups. This also manifests itself in trust. If you don't trust the provider of the service, then you will not rely on that service in your mission-critical application.

Uncontrolled consumerization

Consumer technologies are increasingly being used inside the enterprise without the awareness or governance or corporate IT, according to a recent report from Gartner, Inc. Consumer-facing tools are tightly focused on the creation of a mashup and the visualization, so it can be very easy to create the initial mashup, but longer term maintenance of the mashup is not taken into account. Also consider how dangerous would it be to have an end user upload corporate data into a public mashup tool like Pipes or Popfly.

There are several ways to mitigate these risks. First, for internal or external services, put into place a Service Level Agreement (SLA) that clearly describes responsibilities of both parties, response time for change requests, uptime requirements, bandwidth restrictions, and all other relevant details. Second, lay out the possible fall back requirements in your application when there is a failure calling a given service. For some services, it might be acceptable to just not show that data; with other services, you could cache data that you can fall back on anytime there's a failure. You might need a secondary service lined up as a backup to call in case something goes horribly wrong.

Finally, you will need to address the consumerization issues and the politics. Unlike increasing the reliability and redundancy of services, this requires a governance process. If your organization has a mature process for governing the use of services, then you should leverage that process for mashup creation and consumption as well.

Conclusion

A January 2007 survey by McKinsey asked corporate customers about their adoption of Web 2.0 technologies. As you would expect, a great many of them had either invested or were planning on investing in one or more Web 2.0 technologies. The surprising part of the survey to us was that mashups were only in use or under consideration by 21 percent of the respondents and a majority of respondents, 54 percent, were not considering their adoption at all.

We think that the low response to mashups in the enterprise is due to the relative newness of the technology compared to other ones that were included in the survey (like Web services, podcasts and RSS feeds). The fact that tools to build mashups are just starting to emerge is a factor as well (many of the ones mentioned in this article are in beta and alpha stage as we write this article). We are certain that the techniques will become more common and the tools will mature. Concepts such as the Internet Service Bus (page 2) should make build these enterprise mashups both easier and more useful. We feel that mashups have a place in the enterprise and we encourage you to investigate their adoption.

Resources

"Consumerization Gains Momentum: The IT Civil War," Gartner Special Report, 2007 (summary)
http://www.gartner.com/it/products/research/consumerization_it/consumerization.jsp

"How Businesses are Using Web 2.0: A McKinsey Global Survey," *The McKinsey Quarterly,* August 2007
http://www.mckinseyquarterly.com/article_abstract_visitor.aspx?ar=1913

Mashup (Web application hybrid), Wikipedia
http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29

Redfin
http://redfin.com

"Web 2.0 in the Enterprise," Michael Platt, *The Architecture Journal,* Journal 12

Zillow
http://zillow.com

About the Authors

Larry Clarkin is an architect evangelist with Microsoft. Larry has over 15 years of experience in the design and construction of applications in a variety of technologies and industries. He specializes in integrating Web technologies with Legacy and ERP systems, which he has been doing for over 10 years. When not working with customers, you will find Larry talking technology at local User Groups. You can contact Larry through his blog at http://larryclarkin.com.

Josh Holmes is an architect evangelist with Microsoft. Prior to joining Microsoft last October, Josh was a consultant working with a variety of clients ranging from large Fortune 500 firms to smaller sized companies. Josh is a frequent speaker and lead panelist at national and international software development conferences focusing on emerging technologies, software design and development with an emphasis on mobility and RIA (Rich Internet Applications). Community focused, Josh has founded and/or run many technology organizations from the Great Lakes Area .NET Users Group to the Ann Arbor Computer Society and was on the forming committee for CodeMash. You can contact Josh through his blog at http://www.joshholmes.com.

Microsoft Office as a Platform for Software + Services

by Chip Wilson and Alan Josephson

Summary

The vision of Software + Services (S+S) is to create application architectures that maximally leverage edge resources—the resources of client devices—to provide end users with rich, intuitive experiences. This vision represents an incremental step toward ubiquitous computing, where technology will no longer intrude on the thought processes of people and force humans to think like machines in order to accomplish their tasks. Microsoft Office with its well-understood interface, rich service set, and established business user base, is a natural foundation for creating S+S business applications.

Although the resources of the client computing device must be leveraged to provide a rich user interface, they are not appropriate for executing business logic and processes critical to the enterprise. The system resources ideally suited to hosting mission-critical business processes and data stores are typically centralized in secure, managed environments under the watchful eyes of the IT department's operations team. Service interfaces are the key to leveraging these core business capabilities from the myriad client devices that provide rich user interfaces.

Many client technologies and platforms make up the universe of edge computing, ranging from simple Web 1.0 browser interfaces to full-blown .NET applications running on desktops or laptops. Much has been made of recent announcements from Microsoft regarding Silverlight and Google touting Gears—new entries into the client computing space. Although they add exciting new capabilities to many platforms, there are more proven technologies that are better understood by the existing user base. A compelling case can be made for enabling these existing client platforms to be the "Software" in the Software + Services architectural pattern.

Virtually every knowledge worker in every enterprise is familiar with the same suite of productivity applications—Microsoft Office. The simple fact that so many people have learned how to use it makes it a natural foundation for creating business applications.

Recent releases have morphed Office into much more than a set of standalone productivity applications. Office is now a full-fledged application development platform that provides a rich set of services on which to build enterprise applications. Since so many knowledge workers already use these applications to implement mission-critical business processes, there is tremendous momentum behind an emerging class of solutions known as Office Business Applications (OBA).

Beginning with Microsoft Office 2003, it has been possible to build full-fledged applications on top of Office client applications. This capability goes far beyond what was possible with Visual Basic for Applications (VBA). Although mechanisms for integrating .NET assemblies into Office applications are well documented, the particular design patterns necessary to fully leverage and enhance the user interface provided are not well understood in the industry.

Once the techniques for marrying the Office UI to .NET assemblies are mastered (allowing the code to move information in and out of the documents managed by Office), connecting these documents to back-end systems via services is not only logical but also relatively simple, assuming that the work to expose the business logic as a service has already been done. This overcomes one of the biggest drawbacks to implementing significant business logic within Office—the side effect of creating islands of data, or so-called "Excel Hell." Once the documents are connected to, and acting on, live data in systems of record, they become real-time tools for automating business processes.

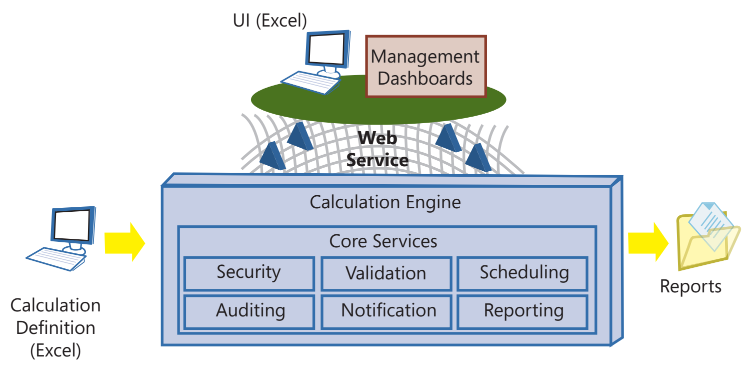
Connecting these documents directly to enterprise systems has the additional benefit of enabling source information to be accessed via services, freeing the end user from the tedious chore of manually entering data from enterprise systems and reports into Office documents. Once the process steps automated by Office are completed, the resulting information can be stored back into enterprise systems, again via services, eliminating additional data entry tasks and ensuring the accuracy and timeliness of the data.

A Common Business Problem

For businesses to thrive and keep their competitive edge, financial modeling and analyses must be done on real-time information without requiring users to access multiple data stores. OBAs expedite access to real-time information by eliminating manual processes for acquiring data and obtaining calculation definitions. These solutions enable Microsoft Excel users to perform complex analyses with data sourced directly from central data warehouses, Enterprise Resource Planning (ERP) applications, or external systems via the Web.

Some limitations of the disconnected Office model stem from the islands of data stored on individual hard drives and network shares. First, there is typically no audit trail for the data. This can be especially true when documents are e-mailed around and changes are made by multiple people, creating parallel versions of the document. Second, because the data is not aggregated, it is difficult or even impossible to

Figure 1: How Excel plays to the Software + Services vision



perform any kind of trending analysis. Last, searching many documents for specific information can be extremely difficult and time consuming.

Real-world examples

To better understand the characteristics of the business problem, it is useful to examine some real-world examples. We present three scenarios where the architectural patterns were used to solve similar business problems in very different contexts and industries.

"Company A" provides employee benefit plans to small U.S. banks and credit unions that want to attract and retain the best executive talent possible. To distinguish itself from the competition, Company A delivers highly customized plans—requiring an intensive administrative process to ensure the highest level of customer service.

Providing that kind of customized service, however, creates challenges. Benefits such as life insurance plans are typically recorded as assets of the banks that are purchasing them for their employees. Therefore, the plans must be handled like other important financial assets that reflect the overall financial health of the institution and that can be affected by external variables such as changing interest rates. Company A must monitor the plans and financing tools continually and provide regular updates to its customers. To do that, they employ trained administrators who can closely manage policy and benefit plan details such as changes in beneficiaries, interest rate fluctuations, retirement calculations, and other factors.

Historically, these plan administrators worked with a number of systems and software, including Microsoft Access and SQL Server databases, Microsoft Office programs such as Excel, and an array of other document processes, including paper forms. (See Figure 1.)

"Company B" is a full-service real estate investment management and support services company, managing investments totaling more than $20 billion in North America, Asia, and Europe. The risk management team focuses on maximizing the potential of the company's overall portfolio by continually monitoring its owned properties and financing instruments. The group maintains a comprehensive asset management database and relies heavily on Microsoft Excel to run analyses on its numerous and varied properties. The team wanted to be able to perform complex risk management analyses with data sourced directly from enterprise systems. The OBA solution enables risk management users to seamlessly retrieve information from the enterprise systems of record and pull it into a familiar, Excel-based application. Real-time data enables better decision-making by providing more accurate and immediate access to information.

"Company C" is a commercial bank with a comprehensive Capacity Management initiative. The purpose of the initiative is to bring product management and sales together with bank operations and IT and, as a unified cross-functional team, identify new incremental revenue opportunities for the bank that take advantage of measured and available unused capacity.

The cross-functional teams, organized according to a business line management structure, operate in a continuous business performance management cycle, where joint projections and performance targets are set and committed to each quarter, and then reviewed with the capacity management steering committee along with new projections for the subsequent quarter or cycle.

For each product, the capacity management team built Microsoft Excel models for the monthly calculations and generation of results for management information dashboards or scorecards. Connecting these models directly to the enterprise systems that contain the cost and production data required to perform the capacity calculations not only eliminated manual entry of the data into the spreadsheets, but allowed the analysts to perform the capacity analysis in a real-time fashion, providing feedback via the dashboards on an ongoing basis rather than monthly or quarterly.

Solution

Generalizing these three solutions yields the Software + Services architectural pattern, an OBA solution utilizing Excel as a client connected to distributed enterprise systems using Web services. Web services retrieve current information immediately into the user's worksheet. Authorized users can make and apply changes to the enterprise systems, ensuring accurate information is immediately available across the entire enterprise.

All charts, graphs, tables, and other reporting tools that reference the changed data are automatically updated to reflect the current, correct information. This solution enables users to seamlessly retrieve information from enterprise systems and pull it into a familiar, Excel-based application. Real-time information enables better decision-making by providing more accurate and immediate access to information. Administration and maintenance of the OBA application are simplified by deploying changes from a central document library.

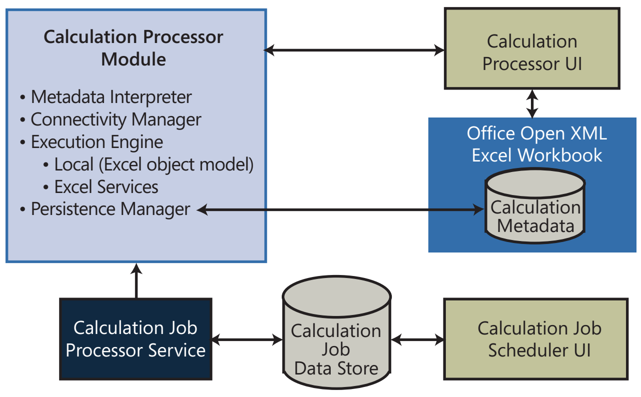
Real-world benefits

Let's look at how this general solution provided concrete business value in the three aforementioned scenarios.

Company A streamlined core business processes by better organizing and centralizing its documents. Web services technologies created flexibility in the company's IT systems, helping managers add and modify plan features for administrators without straining internal resources. Most importantly, the company is continuing its steady annual growth with minimal increases in staff.

The time saved by the solution is equally apparent in other business processes, including those that directly affect Company A's regular communications with banks. If a bank calls Company A and wants to run a "what-if" scenario on a policy—for example, the bank wants to examine the costs associated with accelerating an executive's retirement

Figure 2: Business logic framework components


Calculation Processor Module
• Metadata Interpreter
• Connectivity Manager
• Execution Engine
• Local (Excel object model)
• Excel Services
• Persistence Manager
Calculation Processor UI
Office Open XML Excel Workbook
Calculation Metadata
Calculation Job Processor Service
Calculation Job Data Store
Calculation Job Scheduler UI

date—it now takes two minutes or less because all data is aggregated and easily accessible. In the past, such a process could have taken up to three days because an administrator would have to search for the relevant spreadsheets and extract the appropriate data.

The analysts at Company B continue using the familiar interface of Microsoft Excel to model their real estate portfolio. Using Excel as a client in an OBA provided them with a rich and powerful environment for interacting with enterprise systems. It enhanced and simplified the analysis processes for the users, giving them the data they need in the way in which they are already accustomed to working.

Web services provide a seamless way to exchange large amounts of information over the network, simplifying how client applications generate requests to the enterprise systems and enabling developers to build intelligence into the application.

Company C leveraged Excel to provide a rich client interface for business analysts to build utilization models founded on the principles of activity-based costing. By enabling these models to pull live resource and cost data from back-end systems, Company C was able to automate the process of determining where unused capacity could generate additional revenue without affecting the fixed costs associated with the product, thereby increasing margins. By providing the calculated utilization information to enterprise systems, management dashboards could present not only the current and periodic capacity utilization information, but could perform trending analysis that helps management make decisions about where to focus future marketing and sales resources.

Technical solution

Having solved three different problems with the same architectural approach, a number of commonalities became apparent and several implemented frameworks are general enough to be used on future projects.

A versatile *business logic framework* allows workbook designers (skilled Excel users who understand a workbook's data relationships and their presentation) to change the system's business rules without bringing in a developer to recompile and redeploy the code. Role-based access to workbook metadata provides a special user interface for these designers, enabling them to perform such tasks as changing workbook behavior, altering menu items, controlling how data from enterprise systems is mapped through Web services to workbook fields, and hiding or showing worksheets or individual rows or columns based on data.

A *security framework* makes use of the Code Access Security feature of .NET. Using code access security reduces the likelihood that the application could be misused by malicious or error-filled code. Another benefit is its ability to enable system administrators to specify the resources that an application can use and restrict user access to specific workbooks or servers based on their role.

To address the issue of simultaneously maintaining multiple versions of the Excel workbook across a company, a general-purpose *versioning framework* was created that allows developers to update the assemblies associated with the documents and the data contained therein. Before the results can be submitted to the enterprise systems, the client component of the OBA determines whether a newer version of the document exists by making an inquiry through a Web service. If so, .NET assemblies are downloaded, which upgrade the workbook's data, business logic, and user interface. If the workbook data satisfies its upgraded validation logic, the upload proceeds with the correctly versioned data.

To execute the business rules and data mappings specified in the business logic framework, an *execution framework* reads the business logic, stored as metadata, from the workbook. It then interprets and executes the instructions based on a simple, extensible instruction set. This interpreted approach lets developers easily add new instruction types as needed and provide them to non-developer workbook designers through user-friendly wizards.

Capacity Optimization Application

In order to better understand the solution architecture, we examine Company C's capacity management solution in greater detail. The focus here will be on how a workbook developer leverages the business logic framework to connect workbooks to enterprise systems through Web services and "execute" the calculation tasks created for the workbook in both client and server scenarios. The general solution comprises four components (see Figure 2):

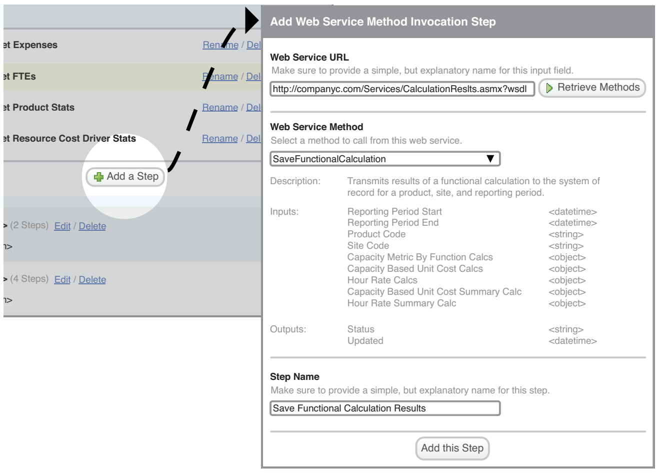
- a *Calculation Processor Module* for configuring, saving, and executing a calculation job (used in both client and server scenarios)
- a *Calculation Processor UI* that implements the *Calculation Processor Module*'s user interface
- a *Calculation Job Processor Service* for running calculations on the server using Excel Services invoked from a Windows service
- a *Calculation Job Scheduler* for configuring the time-based execution of calculation jobs.

Creating a Calculation Task

The *Calculation Processor Module* is responsible for modeling and sequentially executing the steps that comprise a calculation task. A *Calculation Task* is a named execution unit consisting of a simple set of *Calculation Steps* that describe how to:

- invoke Web service methods to move data into and out of the workbook
- move data within the workbook itself
- wire up Excel's UI events to react to user interactions (only executed when the workbook is opened on the client).

Figure 3: Creating a calculation step from a Web service method



et Expenses — Rename / Del
et FTEs — Rename / Del
et Product Stats — Rename / Del
et Resource Cost Driver Stats — Rename / Del
Add a Step
(2 Steps) Edit / Delete
(4 Steps) Edit / Delete

Add Web Service Method Invocation Step

Web Service URL
Make sure to provide a simple, but explanatory name for this input field.
http://companyc.com/Services/CalculationResltls.asmx?wsdl — Retrieve Methods

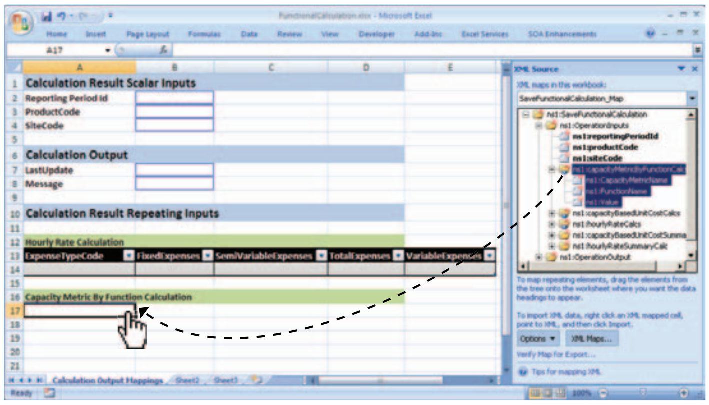
Web Service Method
Select a method to call from this web service.
SaveFunctionalCalculation

Description:	Transmits results of a functional calculation to the system of record for a product, site, and reporting period.	
Inputs:	Reporting Period Start	<datetime>
	Reporting Period End	<datetime>
	Product Code	<string>
	Site Code	<string>
	Capacity Metric By Function Calcs	<object>
	Capacity Based Unit Cost Calcs	<object>
	Hour Rate Calcs	<object>
	Capacity Based Unit Cost Summary Calc	<object>
	Hour Rate Summary Calc	<object>
Outputs:	Status	<string>
	Updated	<datetime>

Step Name
Make sure to provide a simple, but explanatory name for this step.
Save Functional Calculation Results

Add this Step

Figure 4: Mapping a generated Web service method's schema



	A	B	C	D	E
1	Calculation Result Scalar Inputs				
2	Reporting Period Id				
3	ProductCode				
4	SiteCode				
5					
6	Calculation Output				
7	LastUpdate				
8	Message				
9					
10	Calculation Result Repeating Inputs				
11					
12	Hourly Rate Calculation				
13	ExpenseTypeCode	FixedExpenses	SemiVariableExpenses	TotalExpenses	VariableExpenses
14					
15					
16	Capacity Metric By Function Calculation				
17					
18					
19					
20					
21					

XML Source
XML maps in this workbook:
SaveFunctionalCalculation_Map

```
ns1:SaveFunctionalCalculation
  ns1:OperationInputs
    ns1:reportingPeriodId
    ns1:productCode
    ns1:siteCode
    ns1:capacityMetricByFunctionCalc
      ns1:CapacityMetricName
      ns1:FunctionName
      ns1:Value
    ns1:capacityBasedUnitCostCalcs
    ns1:hourlyRateCalcs
    ns1:capacityBasedUnitCostSumma
    ns1:hourlyRateSummaryCalc
  ns1:OperationOutput
```

To map repeating elements, drag the elements from the tree onto the worksheet where you want the data headings to appear.

To import XML data, right click an XML mapped cell, point to XML, and then click Import.

Options — XML Maps...
Verify Map for Export...
Tips for mapping XML

In addition, a calculation task also has a set of user-supplied inputs to kick off task execution.

A workbook designer uses the *Connection Wizard* to add specific Web service method invocations to a calculation task (see Figure 3). Upon specifying the Uniform Resource Identifier (URI) to a Web service's WSDL (Web Service Definition Language), its methods and their signatures are retrieved and presented to the workbook designer who then adds a method of interest to the calculation task as a calculation step.

Once added, the calculation processor dynamically generates an XSD (XML Schema Definition) corresponding to the Web service method's inputs and outputs and adds it to the workbook as an Excel XML map. The workbook designer then maps individual elements of the XML map to worksheets using drag and drop from the XML Source Task Pane (see Figure 4). These will either be scalar values mapped to individual cells or collections of repeating elements mapped to Excel 2007 tables, formerly known in Excel 2003 as Excel lists. While XML maps preserve the hierarchical structure of Web service proxy objects, they become de-normalized when mapping to Excel tables on worksheets.

When service-enabling existing workbooks, it is frequently desirable to preserve the presentation of data in the workbook. Many times, similar data is not presented in the tabular form dictated by the mapping of repeating elements in XML maps. In addition, a limitation of Excel's XML maps is that their cell mappings cannot overlap those of other XML maps. In these cases, it is sometimes necessary to move data from one range of worksheet cells to another to get the service inputs and outputs to "line up." A Copy Data calculation step type lets the workbook designer specify how to duplicate data to other locations in a workbook.

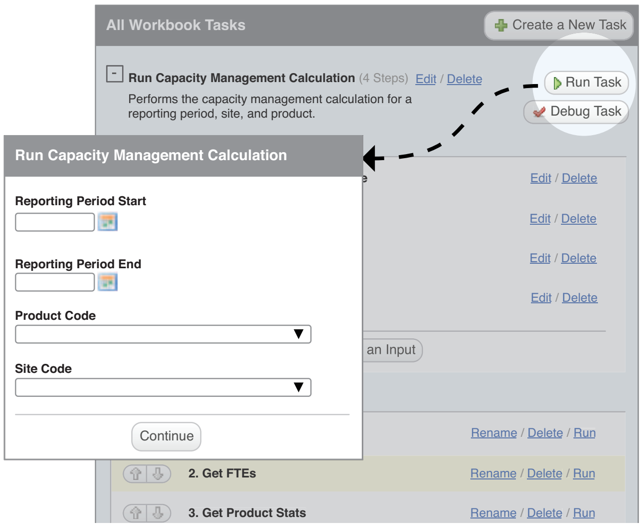
In scenarios where Excel is used on the client to edit and update data, sometimes business logic on retrieved data dictates dynamic changes to the Excel user interface. One such workbook data manipulation step might, for example, hide and show data on worksheets based on specific data values. A *Modify UI* step type lets the workbook designer wire up such interactions.

In addition to creating calculation steps, the workbook designer also specifies well-typed inputs for setting up a calculation task using a *Calculation Task Input Wizard* to specify input names and their data types. The calculation processor dynamically generates an XSD corresponding to the inputs, which is then added to the workbook as an Excel XML map and mapped to worksheet cells, as previously shown in Figure 4. When the calculation task is run in the Excel client, the user is presented with a generated form prompting for input values (see Figure 5). When run by the calculation job processor on the server, input values that were specified using the calculation job scheduler (stored in the calculation job data store) are used.

Executing a Calculation Task

The *Execution Engine* is the heart of the calculation processor module. It is responsible for interpreting the Web service method invocation and copy data steps and executing them in the order specified by the workbook designer in a calculation task. When Excel is run on a client, the execution engine directly accesses the Excel object model, using the XML maps to bind data to/from worksheets. When run on the server, it uses Excel Services to perform the data binding, calculation, and retrieval.

Figure 5: Running a calculation task



All Workbook Tasks — Create a New Task

Run Capacity Management Calculation (4 Steps) Edit / Delete — Run Task — Debug Task
Performs the capacity management calculation for a reporting period, site, and product.

Edit / Delete
Edit / Delete
Edit / Delete
Edit / Delete
an Input

Rename / Delete / Run
2. Get FTEs — Rename / Delete / Run
3. Get Product Stats — Rename / Delete / Run

Run Capacity Management Calculation

Reporting Period Start

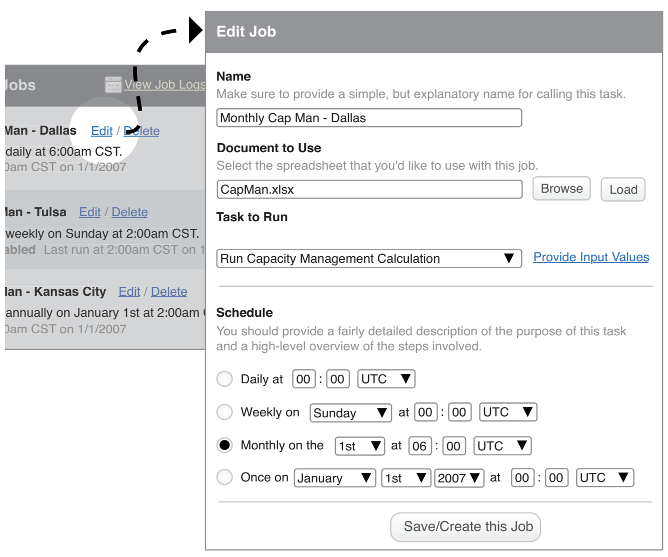
Reporting Period End

Product Code

Site Code

Continue

Figure 6: Scheduling a calculation job



Jobs View Job Logs

Man - Dallas Edit / Delete
daily at 6:00am CST.
0am CST on 1/1/2007

lan - Tulsa Edit / Delete
weekly on Sunday at 2:00am CST.
abled Last run at 2:00am CST on 1

lan - Kansas City Edit / Delete
annually on January 1st at 2:00am
0am CST on 1/1/2007

Edit Job

Name
Make sure to provide a simple, but explanatory name for calling this task.
Monthly Cap Man - Dallas

Document to Use
Select the spreadsheet that you'd like to use with this job.
CapMan.xlsx Browse Load

Task to Run
Run Capacity Management Calculation Provide Input Values

Schedule
You should provide a fairly detailed description of the purpose of this task and a high-level overview of the steps involved.

Daily at 00 : 00 UTC
Weekly on Sunday at 00 : 00 UTC
● Monthly on the 1st at 06 : 00 UTC
Once on January 1st 2007 at 00 : 00 UTC

Save/Create this Job

Current limitations to Excel Services prohibit workbooks with XML maps from being opened on the server. To get around this, the calculation processor module extracts the binding information stored in the XML maps from a workbook, removes the XML maps, and presents a "clean" copy to Excel Services. Web service method invocation steps use this binding information to determine how to set and get cell ranges within the workbook through the Excel Services API. This approach is akin to a "push" model where data is fed to the document using the explicit control structure specified in the calculation task. An alternate approach that leverages Excel's User Defined Functions (UDFs) to wrap Web service calls is possible but has several limitations, including:

- reliance on the spreadsheet's dependency graph for the sequencing of service method invocations (no explicit control)
- only a predetermined amount of data can be returned from UDFs since the array return value types must be bound to fixed Excel ranges
- additional maintenance is required to deploy and trust UDFs under Excel Services.

The *Persistence Manager* abstracts storage of the calculation processor metadata. The serialized metadata is stored as a Custom XML Part (new to Excel 2007) within the Office Open XML package that represents the workbook. In this way, all information about the calculation job is stored within the workbook itself, available to the calculation job processor service.

The *Calculation Client's* user interface can be implemented either as an Excel add-in or as an Excel customization that adds menu items to the Excel user interface for accessing a set of Windows forms. These forms extend the Excel user interface through a set of wizards so that a non-developer workbook designer can configure, save, expose a calculation task for execution, directly execute it, or step through its calculation steps. A different class of user, unable to modify the calculation, might be granted restricted access to only the execution interface.

Scheduling and Running a Calculation Job

The *Calculation Job Scheduler* is an application through which an administrative user configures a workbook and its calculation job for timed execution (see Figure 6). The information supplied is:

- a path to the workbook
- identification of which calculation task from the workbook is to be run
- its schedule for execution (recurring or a single execution date)
- initial inputs to the calculation job as determined by the calculation task's metadata
- a path to the folder where audit snapshots of the workbook are to be stored.

The *Calculation Job Processor Service* runs as a Windows service and invokes the calculation processor module when a calculation job's scheduling criteria are satisfied. Initial inputs that were supplied when the job was scheduled are passed to the calculation and the workbook's calculation job is executed using Excel Services. Optionally, a snapshot of the workbook (after the calculation task has been completed) can be saved to a specified location to provide an audit trail of the calculation job.

Conclusion

This architectural approach to solving an extremely common business problem has been utilized multiple times now. It has proved to be robust and reliable, providing tremendous business value. In one case, this solution completely revolutionized the business model of the company by allowing it to offer services that no competitor could even approach.

The Software + Services vision of rich user interfaces on client devices invoking services in the cloud is one well-suited to the Microsoft Office system, especially since the user interface provided by Office is so ubiquitous and well understood.

References

Denise Partlow, EMC Global Services (formerly Geniant, LLC). "Improving Data Integrity in Financial Analyses." April, 2006.

Denise Partlow, EMC Global Services (formerly Geniant, LLC). "Real Estate Firm Simplifies Risk Analyses with Web Services and Excel Smart Clients." April, 2006.

Microsoft Office Business Applications
http://office.microsoft.com/en-us/products/FX102204261033.aspx

About the Authors

Chip Wilson is an Enterprise Architect with EMC Global Services. His recently published book, Transparent IT: Building Blocks for an Agile Enterprise (www.TransparentIT.com), describes a detailed framework and roadmap for adopting a service-oriented architecture and creating an agile enterprise.

Dr. Alan Josephson is a Senior Practice Consultant with EMC Global Services, specializing in Microsoft Office system development. He has created custom Smart Client and Office Automation solutions for clients in the financial, insurance, and energy sectors.

A Planet Ruled by Software Architectures

by Gianpaolo Carraro

The world of software architecture took a "giant" leap forward when it moved from three letters acronyms (SOA and ESB) to four letter acronyms (AJAX and SaaS). As you've witnessed in this issue, we now have a new acronym that includes a non-alphanumeric character: S+S (Software plus Services). Two questions are now in all our minds:

1. Will analysts feel threatened by this new breed of acronym and up the ante by introducing acronyms requiring Unicode encoding?
2. How can I effectively communicate S+S within my organization?

As for the first question, time will only tell! In an attempt to answer the second however, I wanted to explore an analogy of a planet with imaginary epochs, where each epoch is dominated by a specific civilization, metaphorically representing an architecture style. Make sense? I hope so. Let's begin...

The Dark Ages

Architectopia has not always been the happy place it is today. Not so long ago, most of its land was dominated by the harsh Big-Iron Empire. Although not malevolent, the Big-Iron Empire imposed very strict rules on the local populations; in those days, freedom of expression was very limited. During the Dark Ages, there was no other option than to respectfully comply with the central authority of Emperor Mainframe the First.

The Renaissance

The Renaissance was a time of rapid innovation which saw the emergence of new political systems, the beginning of modern science, and geographical exploration. During this time, Architectopians were taken by the strong desire for emancipation which led to some decline of the Big-Iron Empire and the creation of several powerful states, most notably: the Republic of Desktopistan and the Great Duchy of Enterprisia.

The Republic of Desktopistan

The Republic of Desktopistan was founded on the principles of individuality and independence. Gone were the rules limiting expression. Everybody could now have access to literature, mathematics, and art; to make sure of that, a book, an abacus, and canvas could be found on every desk in every home.

No single house in the republic is the same, with custom layout, homemade wallpaper, and distinctive extensions, such as extra storage, voice-activated lighting, and large screen TVs, being the norm. In Desktopistan, it is expected that craftsmen will produce art that fits in these customized surroundings and extensions. Many loathe cookie cutter, common denominator approaches.

The strong individuality of the culture comes at a price however; it is a very lonely society. Social gathering and sharing of experience are not part of their traditions. Table 1 compares the pros and cons of living in Desktopistan.

Great Duchy of Enterprisia

A strong ally of the Big-Iron Empire during the Dark Ages, the Great Duchy of Enterprisia re-invented itself during the Renaissance. The dependence on the Empire faded away given the increased influence from Desktopistan. The motivations were not philosophical but instead grounded in the economical benefit of such change. This is not surprising as the Great Duchy has always been run as an "econimicracy," meaning that all political decisions must result in an increased value to the total Great Duchy assets. Under these rules, some discontent in the population is acceptable, especially if happiness is deemed too expensive.

Even though in principle, the Great Duchy is open to embracing new models for improved overall benefits, it takes a very conservative stance in the adoption of such models, tools, and equipment. This slow adoption, used as a protection mechanism, is overseen by the potent 27-member-strong "pragmatic committee of new adoptions." According to the Duchy's rulers, slow adoption is a small price to pay for the rigor and control enabled by disciplined dissemination of new technology. With this said, it can be observed that many village chiefs don't comply with the slow adoption process and adopt new models locally without the authority of the Great Duke. It is usually too late for the Great Duke to change anything by the time he hears about it.

On the positive side, high standards of quality are prevalent in all disciplines. The Duchy's infrastructure such as roads and electrical power, although old-fashioned, is very dependable and suffers rare outage. The legal system is also well-developed and contractual service agreements are generally well-respected.

Lately, as part of its five year growth plan, the Duke himself chartered the Society of Order and Abundance (SOA) to reform many of the Duchy's modus operandi. It is not clear yet whether large benefits will be obtained without reforming more

Table 1: Pros and cons of life in Desktopistan

Pro	Con
• Rich experiences • Full use of local resources • Extensibility	• Isolated experience

Table 2: Pros and cons of life in the Great Duchy of Enterprisia

Pro	Con
• High standards of quality • Contracts and Service level agreement based economy • Governance	• Slow to adopt, slow to agree • Lack of agility • Too much legacy

fundamentally the Great Duchy, but the hope is high. Table 2 shows the pros and cons of living in the Great Duchy of Enterprisia.

The Eruption of Mount Web

The People's Republic of the Online World is a young nation founded upon a new continent created by the violent eruptions of Mount Web. In many aspects, the People's Republic embraces the opposite values of the Great Duchy of Enterprisia. As much Enterprisia is system-centric and process-driven, the People's Republic is fast-moving and people-centric. Instead of relying on established, well-proven practices, citizens of the People's Republic favor trial and error. Another point of divergence is around craftsmanship. It is clear that the People's Republic preferred mode of craftsmanship is lightweight and functional rather than elegant or built to last. Even time seems to differ, whereas the rest of Architectopia follows long orbital years, in the People's Republic, time is counted in much shorter southern moon days.

A common saying in the People's Republic (which is considered heretical in the Enterprisia) is "let's try, we'll see what happens." Explosions are frequent in the People's Republic laboratories. This is not surprising to many, as most of the scientists are teenage aspiring alchemists who refuse to attend the Great Duchy of Enterprisia's best universities. Instead they dedicate their inexhaustible energy, looking for the philosopher's stone, which in the local dialect is called "IPO." From a sociological perspective, it is interesting to note that many of these experimentations are sponsored by a secret society known only by two letters: V.C. Not all experiments end in explosions, however; the few that succeed usually change the face of Architectopia and become models for all the other countries, albeit adapted to the local cultures.

A fundamental value of their culture is their focus on people-centric ways of work, enabled by a highly advanced communication infrastructure. In the People's Republic, little seems to be enjoyable if it is not done together. Work is done in group, massive social gathering are common, and reaching out to everybody is a core pillar of their philosophy.

A final aspect of cultural interest in the People's Republic is the notion that every purchase should be free, according to its citizens.

Table 3: Pros and cons of life in the People's Republic of the Online World

Pro	Con
• Reach • Collectiveness • Rapid innovation (both technical and in their business models)	• No contracts, all transactions are "I'll do my best" • Tend to cater to a common denominator experience.

Table 4: Pros and cons of life in Deviceland

Pro	Con
• Mobility • Access everywhere • Form factor	• Form factor can degrade the overall experience

The only request is to spend time at the end of the shopping experience in front of a large screen where propaganda (ironically, from the Great Duchy of Enterprisia) is shown. The time you are requested to watch is directly proportional to the estimated value of your shopping experience. Table 3 shows the pros and cons of living in the People's Republic of the Online World.

Deviceland

Kermit-TTY, the first ruler of Deviceland, arrived late to the Dividing Treaty of Architectopia, which allocated the various continents to the different early civilizations (allegedly because he "didn't get the memo"). Since then, Deviceland inhabitants are obsessed with having access to information anytime, anywhere.

Composed of nomadic tribes of road warriors, Deviceland residents have fully adapted their habits to accommodate their high speed, always-on-the-run lifestyle. The ingenuity of the tribes is not to be underestimated; it is common practice for them to look at what is happening in the other countries and adapt the advances they find to their nomadic life—usually however, at the expense of the richness of the experience.

In Deviceland, tribes live happily in isolated territories, but in constant contact with each other and the capital city. The most likely place to encounter tribe members is at specific locations known as Hot Spots. Hot Spots are very popular in Deviceland as they provide, "the juice," a necessary energy source for all, as well as "strong signal" which is a quasimystical electromagnetic field providing a deep sense of peace to the people. The Hot Spots are the central theme of the Deviceland anthem: "Can You Hear Me Now?" Table 4 lists pros and cons of living of in Deviceland.

The New World

The recent colonization of Servicetopia, the previously hostile territories of Architectopia, triggered something new. Instead of living in isolated city-states as was prevalent in their homelands,

Table 5: The central attributes of the United Federation of S+S, and the originating influence.

Attribute	Influenced by
High Standards and SLAs	Great Duchy of Enterprisia
Individuality and richness of experience	Republic of Desktopistan
Reachness, collectiveness and rapid innovation	People's Republic of the Online World
Optimized hardware form factors and mobility	Deviceland

Figure 1: The flag of the United Federation of S+S

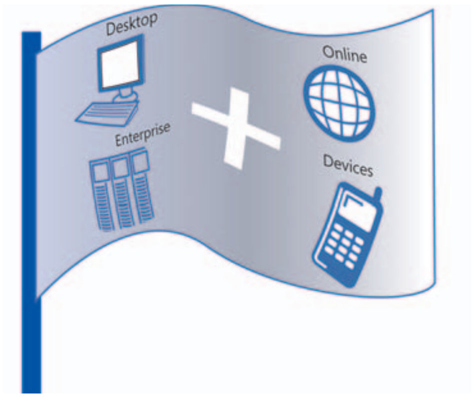
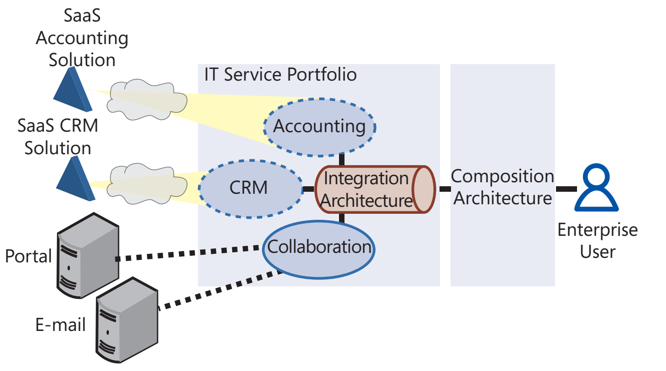


Figure 2: A possible S+S pattern in the enterprise



settlers from the old world; namely the Republic of Desktopistan, Great Duchy of Enterprisia, People's Republic of the Online world, and Deviceland inter-married and embraced each other values.

Diversity and respect for choice became the core ethos of Servicetopia. It is therefore not surprising that when Servicetopia became the United Federation of Software + Services (UF S+S), diversity and choice became central elements in the constitution. In fact, as represented by their flag, one can think of the United Federation of S+S as a powerful blend of what each old-world culture had to offer (Figure 1).

In its essence, UF S+S is a society that has banished the tyranny of having to choose one way of life over another, and has instead embraced the power of fusion. Both the high standards of Enterprisia and the freedom of experimentation of the People's Republic are sought; both the rich experiences native of Desktopistan and the mobility commonly found in Deviceland are looked for. Dogmatic opinions and sacred cows are replaced by pragmatic decision making, smoothing the process of making the right choice at the right time. Table 5 shows the central attributes of the United Federation of S+S, and the originating influence.

What does all this really mean?

The goal of the preceding "tongue-in-cheek" story was to convey the fundamental premise of Software + Services: hybrid architectures are good.

Akin to fusion music where pop, folk, and reggae are mixed, or fusion cuisine where foie gras sits on top of a sushi, Software + Services blends architecture patterns found in the enterprise, on the Web, in desktop applications, and in devices. Instead of choosing one over another, Software + Services embraces them all and acknowledges that patterns in each domain bring unique opportunities to an overall solution.

The proposition is therefore to part from a "one size fits all" model, and instead to blend multiple architectures within the same solution aiming for a "best of all worlds" approach.

It is no longer compromising richness for reach but offering both; it is not compromising strong control and data ownership of on-premise solution for a more economical multitenant cost structure, but achieving both.

Many examples of Software + Services exist today. The Xbox / Xbox Live combination provides a perfect example, by providing rich 3-D experiences maximizing the graphical rendering power of the local console ("Desktopistan" influence), while allowing participative experiences through the Xbox Live service ("People's Republic of the Online World" influence). This pattern is of course not only used by Microsoft, but is becoming an industry trend. Apple's iPod / iTunes, Salesforce.com / Salesforce.com Offline Edition combinations are also examples of this model.

Software + Services is not limited to rich local clients accessing SaaS applications, however. Another example could be the extended Service-Oriented Architecture (SOA) model where enterprises run part of their service portfolio locally, and part of it in the cloud, as illustrated Figure 2.

A large number of scenarios are possible under the Software + Services umbrella, and I would argue that the three most frequent realizations (at least initially) will be:

1) Local software complementing a cloud service (for example, an Outlook-based interface to CRM Live). This model combines the rich, responsive, familiar, local user experience with economy of scale, one-to-many SaaS delivery.
2) Local software being augmented by a cloud service (for example, a cloud-based anti-spam, anti-phishing service augmenting a locally run mail server). This model allows a multitude of value-added services to be added in the cloud, freeing existing systems that control data deemed preferable to keep within the corporate boundaries.
3) Location-independent, many-to-one service consumption in corporate IT (for example, "extended SOA" scenario). This scenario is a typical IT optimization scenario.

Finally, to conclude this tall tale, I leave you with the recommendation that would have been, I am sure, the opening statement of the hypothetical United Federation of Software + Services constitution: "Embrace Diversity, Demand Choice."

Acknowledgment
Big thanks to my brothers in arms, Eugenio Pace and Frederick Chong, for the conversations leading to the different fictitious countries.

About the Author
Gianpaolo Carraro is director of Service Delivery – Microsoft Architecture Strategy. You can learn more about him through his blog: http://blogs.msdn.com/gianpaolo

ARC

Microsoft®

THE ARCHITECTURE JOURNAL™
Input for Better Outcomes