

## Service Oriented Architecture Come of Age

### Industry Overview

Service Oriented Architecture (SOA) is moving surely along the further end of the adoption parabola. The Web Services vendor community, once considered an exotic species within the mainstream IT cohort, is at last providing solid tools for application creation and reengineering via services composition. This soft milestone is verified by the fact that one is generally no longer required to be completely conversant in the W3C specifications when employing the latest generation of Web Services tools. - this indicates progress.

The SOA grail-quest is an old as computing itself, and indeed mimics our life's desire to simplify and understand relationships, causes and effects of our actions. Certainly, if one could travel back in time to read the mind of Alan Turing and Company, toiling away at Colossus in 1943, the first digital computer used to break the WWII German Enigma Cipher, you would detect an innate psychic murmuring, 'bollix, I wish we could just invoke some well defined service, without all this fuss and bother'.

The computing industry has listened, as reusable code libraries and dynamic programming models have multiplied like genetically engineered rabbits. Great innovators of computing architecture led the industry through iconic changes; from the centralized Iron giants of yore, to the minis and super minis, on to the PC and the catalyzing influence of the Internet and the World Wide Web. Each revolution in computing hardware and connectivity begat a concomitant revolution in system's programming methods.

And here we are, newly arrived as beneficiaries of years of struggle in the data format wars (a.k.a. XML), and in an era where the world's computing infrastructure is decidedly enmeshed. Though it took a while for the academic and vendor communities to settle down, we are now seeing post-early-adopter uses of SOA.

### Tracking SOA Adoption

Presently, capital line-of-business applications and custom programming may be invoked as network based services with well-defined functions. Perhaps as important, the spreading of database management systems is becoming far more manageable in the guise of component services presenting a unified data model. This synergy hastens the promise of SOA, as this seamless unification of application functionality and data cohesion allows for the composition of new applications, and creates an agile growth environment for existing systems to evolve. Our new service components may be composed of both new and older cloth; we will build a new generation of applications with minimal disruption to current processes, preserve existing system emplacements, and orchestrate the consumption of these network-available resources in a ballet of Service Oriented Architecture.

For those of us tracking the Web Services Rodeo from the outset, the relief etched in the faces of 'real users' is apparent. The progression of inscrutable terms such as "executable end-point", resulting in the blankest of stares, is being replaced with a relaxed comprehension and an increasing grasp of the substantive subject matter. Naturally, larger, resource rich companies

took up the SOA gauntlet first, adapting legacy infrastructures using the principle of composition.

Another promising hallmark of the SOA adoption curve is capital line-of-business software vendors, like SAP and Peoplesoft. Vendors who have long offered voluminous platform specific API integration libraries have warmed to the Web Service siren call. The benefits are manifest, as callers and service providers become increasingly platform agnostic using the lingua franca of web services : 1) XML for data/metadata, 2)SOAP –XML-RPC for messaging envelopes, and 3) WSDL for self-documenting service parameters. These basic building blocks of SOA provide a foundation for further enhancements and services for security and reliability.

The advent of XML's ubiquity is a study in the chaos of convergence. Mirroring the renaissance of the Web, which the Internet proper preceded by years, the wonderful-horrible marriage of HTML and Web Browser created a democratizing influence over on-line business and personal publishing. An increasing set of ever more accessible tools is making SOA the latest chapter in a fairy-tale of web services convergence. The delivery of XML-based services crossing boundaries of OS specific silos of functionality is but one of our IT era's latest success stories.

This is the Server Side revolution. Application frameworks have evolved with particular zest, keeping pace with the explosion of better performing, lower cost databases. Significantly less mental capital is being squandered agonizing over the application/data relationship. Core data relationships are still the subject of intense design scrutiny; yet the legacy of the last forty years of IT science is changing rapidly with the proliferation of object databases, object-relational mapping, dynamic languages, and web services. This domain of applications and the relationship to underlying data is now a much more flexible endeavor due to the foregoing innovations.

It's a rosy picture, yet we still struggle with systems that are the progeny of monolithic schemas, vast ER diagramming, and massive joins and foreign key relations. For SOA, a real renaissance is just starting with dainty, yet increasingly confident steps. Data is becoming more universal, contained in smaller functional domains (or made to act like it), with wider accessibility, and with better tools for accessing monolithic *and* heterogeneous data storage systems. These tools and new "Super Platforms" are being innovated at a furious pace. Data is becoming smarter, with abstract object storage freeing programmers from the 'tedium of where'.

Programmers need to know less about the minutia of rows, columns and foreign keys as advanced development frameworks become even more closely bound to modern middleware, Virtual Databases, and native object relational mapping. Each year brings us closer to an ideal of universal data objects that act with increasing intelligence.

In this pouring from the empty into the void, questions arise – these were my analyst-style questions in the dim, early days of Web Services and SOA:

- How will services be created and broken out of their monolithic application stack?
- How will new services access data from previously monolithic database schemas?
- Does a services invocation approach obviate the need for old-time data modeling?
- What am I going to do with all this SQL and stored procedures? Throw it all away?
- How am I going to Query XML native objects in a new SOA world?
- What about data in multiple DBMS systems (ours and our partners) that are not in XML?

### **Creating a Culture of SOA**

The march of time has not changed the fact that applications still require data access and persistence; what has changed is the *diversity* of data storage platforms. It is not uncommon for application 'A' to access one DBMS, and applications 'B,C,D, and E" to persist data in other platforms. The two-tier application revolution, now very mature, has created data and application

silos that may use SQL, applications server session data, flat files, and industry specific messaging formats, like EDI, or ebXML.

To institute an effective SOA plan, we need to factor essential service functions by type and compose individual and composite services that make sense. Furthermore, an important key to this network accessible service model is data availability via a unified data model spanning our diverse storage platforms. A service factoring process therefore comprises an application service invocation construct, resulting in a set of URI end-points that deliver finite, well-defined services with concomitant access to data.

The foregoing is an enduring drama played on the stage of IT history; if anything has changed culturally, it's the competitive concerns of delivering innovative services in an agile fashion.

The middleware era is alive and well, and universal data access for applications (ODBC,JDBC) is still a vibrant and profitable sector. Virtual databases are also carrying a substantial portion of enterprise integration workloads. A minor shortfall in this approach is the somewhat dated reliance upon explicit SQL wrangling. The Web Services renaissance, based on XML protocols, makes this reliance on explicit table definitions messy. The world of SOA is striving for a more elegant medium in which to express data object relationships for applications invoking web services.

## **Data Services Construction**

Evolving a new model of the data universe in light of SOA philosophy devolves upon functional isolation – liberation of discrete services by function. In the real world, we have applications that Create, Read, Update, Delete, and Calculate, etc., while database systems may be scattered all over the place with poorly documented schemas, and application bindings. For a fact, most two-tier applications, client/server or Web Based, rely upon services bound tightly to an application's event loop. Such architectures do not readily lend themselves to reuse; even those with ample, bulging API binders.

SOA modifies the approach of classical database schema design in the grand tradition; working from what currently exists, we liberate data service classes as objects, and create new functions in order to transit these service classes upward towards network-based interoperability.

The concrete approach of service composition is at the core of data services architecture. Common, plainly understandable services are mapped to business objects - such as inventory and supplier service objects. These services abstract the data interaction layer of a typical enterprise application, and provide an explicit implementation of a generic method. Like Oz behind his curtain, a service invocation may interact with any number of data resources through a unified data model spanning multiple DBMS systems locally or at far remove.

This view of network accessible services is very different in terms of types of service containers or classes, rather than explicit functions calls:

- Abstract data services rather than explicit SQL declarations
- Services for application-centric functions, which manipulate the abstract data objects.
- Service container classes for inter-application, pardon me, *inter-service* messaging – a transit service for data that must be shared and combined
- Service methods that cater to user interfaces and machine interfaces

Variations of the above are becoming standard constructs, some call this a "Design Pattern", and

there are many. The important point is that SOA methods embrace a notion of data abstraction through invocation of service methods, which in turn expose functions of the underlying data management layer. This all should occur with lower development overhead. This 'thought bending' spills over into application frameworks and IDE's that encompass both light and heavyweight implementations of the Web Services and SOA paradigm.

## **In the Real World**

A service abstraction consists of the application logic for servicing a function. Natural orders of selection of functions arise from our legacy applications, and as composite functions that need to be invoked in tandem.

In our medium manufacturing example, we know that Supplier ERP and inventory records are interrelated by our legacy systems. Although one application is store bought (SAP), while the inventory system is older, using an aging database system, we have in embryo an ideal test case for an SOA approach to integration.

An add-record method exposure for the supplier/inventory service class may be proffered as a composite access invocation. Using an SOA framework providing a design time environment for the exposure of SQL stored procedures and application code, we may create a WSDL service, in the truest sense of the word, comprising one or more SOAP invocations to Create, Update, Delete, or Read the Supplier data in both inventory and supplier ERP DBMS systems. This service mapping should not at all be disruptive of the ERP system's primary user interface, as all the rules of the underlying data structure are kept intact.

Service container descriptions and 'super service calls' are one way to structure data access and application functions in the SOA world.. A modest implementation might be better served by a list of direct service invocations, listed in a UDDI directory. One may be limited by the type of tools adopted; as with all with new techniques, SOA applications are an expression of tools, talent, and persistence.

Many companies, not all of them gargantuan, are eager to offer web services for partners and clients. These companies also have a motivation to simplify an aging and increasingly diverse data infrastructure. In one fell swoop, we are witness to an increasing number of SOA pilot projects and internal (possibly covert) laboratories engaging and contributing to these worthy aspirations.

## **SOA Methodology in Practice**

Integration implies a certain amount of work; the scope of a project's complexity depends on the underlying data access system's type and the quality of application code held over from pre-SOA project days. There are nuances in the world of SOA; services may be called via traditional programming methods, or the latest declarative and graphical design tools may be employed.

Services managing information in our CRM/Inventory example provide two invocation calls for storing or retrieving supplier data. Each service invocation has an XML Schema describing record metadata (described by WSDL). Providing transparent access to the data residing in each system mandates the formation of a unified data model in order to normalize the calling mechanics to what were two distinct data management sources.

A Classic programmatic method of service invocation requires a code loop or script to retrieve

information returned from aggregate data sources called from the service object. If this data is bound for a web based UI, Java Script or XHTML style sheets can do the job, while many new programming frameworks are pitching new-fangled AJAX. The downside is that this repetitive scripting must be duplicated for each service invocation.

Without the base-class empowerment of a robust framework, possibly Django, Struts, or Ruby on Rails, the programmer must be resigned to a certain amount of vigilance. Any modification of schema or method handling will impact maintainability. While a framework's ORM class handling can help with repetition and adaptation, ultimately, placing service invocation in a code or script loop can be bothersome. New techniques for modeling service objects are being innovated at a furious pace by framework vendors (and the Open Source Community); these advances will obviate some of the pain involved by dynamically propagating schema changes to view methods.

An alternate path uses declarative queries and process orchestration. XQuery and BPEL, shaken well, is a hybrid methodology worth examining. Ultimately, any combination of scripting, frameworks, XQuery, and BPEL, will be completely interoperable as development environments mature.

A declarative approach to SOA on Web Services might use XQuery to define super methods for marshaling data services. Executed with aplomb, we have created an 'abstraction of an abstraction', returning a unified dataset within the WSDL invocation URI for 'add supplier'. This declaration can be defined centrally, consolidating all potential changes to the data access methods in one location, freeing service consumers from yet another layer of integration details. A properly implemented XQuery processor leverages all the power of the underlying data virtualization layer, with its concomitant power to optimize query syntax and cost modeling.

A small but growing number of contemporary super application serving platforms provide this highly flexible layering of core database services, data access virtualization layer, and in-line query transformation. Such platforms are all about application composition via Web Services, i.e., the creation of SOAP objects hosted in HTTPS accessible virtual servers, automatic generation of WSDL, with integrated BPEL to tie all service invocations together at a highest level of abstraction within an integrated application hierarchy.

## **Conclusion**

The adoption of a Service Oriented Architecture in no way implies a single methodology for accomplishing an integration task. Services invocation may be packaged and executed via programmatic and declarative methods, or a hybrid of these techniques may be desired. The future basks bright in the sun of convergence, where a golden lasso ties together the best of Web Services platforms, Web Application Frameworks, and data unifying architectures.

Data services are a basic building block of SOA. The explosion of middleware components and super application servers help us get SOA designs off the ground in a unified, data-agnostic fashion. New Application Frameworks for creating Web Applications provide even more leverage for the programmatic consumption of these services.

The layered power of Data Virtualization, declarative abstraction for querying data supersets (XQuery), and Web Services in the form of SOAP, WSDL, and BPEL, create a rich environment for the creation of Service Oriented Architectures with less angst.

