

The SOA Magazine

Feature Article



The Benefits of a Data Abstraction Layer for SOA

by Kirstan Vandersluis

Published: June 16, 2008 (SOA Magazine Issue XIX: June 2008)

[Download this article as a PDF document.](#)

Abstract: Enterprises have complex information structures as a result of evolving applications and databases deployed through the years. In addition to modern databases, environments commonly include mainframes, outdated storage technologies and custom data sources.

Companies seeking increased agility and reuse through service-oriented architecture quickly find that making sense of widely distributed and disparate data is a major roadblock to achieving the benefits of SOA. To build a successful SOA, architects need to pour the foundation first – they need to begin with a data abstraction layer that makes sense of an otherwise chaotic data landscape.

Data abstraction leads to the ability to leverage physical data, no matter how it's structured, as new, logical schemas that exist only in middleware – creating a common data layer that architects can restructure as needed, rather than making costly changes to the physical database or core services.

This article examines the use of a common data layer leveraged through data abstraction, its key advantages, and the value and ROI it generates for enterprises that deploy a SOA.

Introduction

Companies use information technology to automate business processes in an effort to reduce costs, increase efficiency and provide better service to their customers. Over the years, new systems and applications have been deployed to address specific needs, most often without an overarching design. The result is a wildly diverse sea of data, spread across multiple systems, in different formats, and often times duplicated. This data is the lifeblood of a company and the core of any new business process.

Historically, the practice of automating a business process involved a large investment of time and money. If the business environment was stable and unchanging, then this large investment would deliver business value year after year, indefinitely. Today's businesses, however, face constant change caused by new regulatory requirements, fluctuating market expectations, mergers and acquisitions, and the desire to continually improve their overall operational processes. Because of the high reliance on IT, such changes in the business environment lead to changes in IT. As a result, the value of IT in this environment is largely measured by how quickly it can satisfy these new requirements.

The evolving architectural style of SOA offers hope for providing rapid implementation of new business requirements, enabling a company to be agile in the face of a changing market environment. An agile business is one that is not afraid of new requirements; rather, it sees change as an opportunity to surpass the competition by exploiting its ability to rapidly adapt to new environments and circumstances. It achieves market advantage by efficiently retooling to meet new challenges, avoiding impending business obstacles, and increasing its market share by competing and winning against less nimble competitors.

The single most important aspect of SOA is its ability to enable access to the enterprise data that powers the business processes. Without efficient access to data, the entire business suffers. A data abstraction layer reduces the architectural complexity required to provide access to enterprise data, and allows organizations to create and leverage services that support critical business objectives.

Managing Data for SOA

While the need to manage data is nothing new, recent IT trends have been emphasizing traditional techniques. Adoption of SOA in particular, while showing great promise for increased agility and reuse in the enterprise, calls into question old methods of accessing data.

The premise of SOA is that processing capabilities are built into callable chunks that roughly align with business operations such as "get auto policy," "process order," or "verify credit limit." Then, processes and applications are constructed by "orchestrating" these services into compositions that automate business processes. When this vision is realized, businesses can react to change quickly by simply re-orchestrating services rather than succumbing to the traditional six, 12 or even 18 month development cycles that are common with silo-based application delivery.

There are three main characteristics of services that drive the value of SOA: they are typically defined in business terms, loosely coupled, and coarse-grained. Specifically, this means:

- Defining services in business terms helps align IT with business needs, allowing the business to use IT services as well-understood building blocks for business processes. For example, you might define a service as "update customer record" instead of "send MQ update."
- Loose coupling is the notion that services should have minimal dependencies on one another. Loosely coupled services generally have a defined "contract" under which they interoperate. One service may change its implementation without affecting the other, as long as it continues to abide by the contract.
- A service is coarse-grained when it delivers a significant piece of data given a limited interface. A service with a coarse-grained interface intentionally inhibits flexibility in favor of formalizing and standardizing a well-defined behavior and data set. For example, a service that provides a "get auto policy" operation might take a single parameter, the policy number, and return the policy document using the company's standard format.

The need for services to be loosely coupled, coarse-grained, and defined in business terms presents several important problems that must be solved for a successful SOA. Somehow, architects and developers must find a way to deal with the complexity and diversity of data in the back-end systems, develop representations of data at a suitable granularity, and define information objects in business terms, like "order," "customer," and "invoice."

It is worth noting the distinction between simply exposing data as services and truly building an SOA. Many applications are written without a data abstraction layer. Indeed, modern development environments for Java and .Net make it very easy to bind an application or service directly to a physical data source. While data services can often be quickly developed and delivered in this manner, this approach predictably results in services and consuming processes becoming tightly coupled to the underlying data. Any change in the data or the consuming processes causes a ripple effect throughout the entire stack.

Managing Complexity with a Data Abstraction Layer

Data abstraction hides the complexity of data by letting you define a new, better organized structure that exists only in the middleware. The result is that an application (or another service) can request the data in a well-organized, logical format, without having to worry about its actual physical layout. As an example, an application may request a customer record from the data abstraction layer. Data is fetched from potentially many data sources, transformed into the agreed logical structure, and returned to the calling application.

Various types of data abstraction layers are possible, each suitable for a particular purpose. Examples include a virtual database that presents a relational model of data that is "queriable" using SQL, and virtual spreadsheets that present back-end data for easy manipulation in a spreadsheet form. Broadly speaking, a virtual relational database maximizes the flexibility in the types of queries that can be issued, due to its rich query language. To provide data to services, however, the natural implementation of a data abstraction layer is to model the data in XML, which makes it an ideal fit for contemporary SOA implementations because the richness of data defined via XML Schema fully supports the

creation of coarse-grained and loosely coupled services, defined in business terms.

An XML-based data abstraction layer solves both of our problems simultaneously: we hide the complexity of the back-end systems so that applications can avoid the messy details, and we provide data structures at an ideal level of granularity in support of SOA.

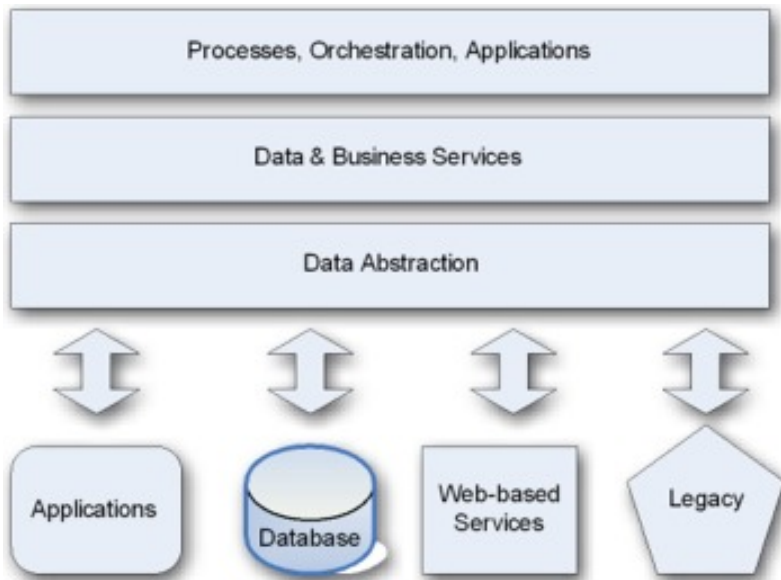


Figure 1: A data abstraction layer hides complexity and supplies appropriate data to higher layers.

Modeling Data Using XML Schema

One of the obstacles to defining a data abstraction layer is the ability to properly model the data using XML Schema. Like any data modeling activity, this task carries a degree of difficulty requiring input from both technologists and business experts. In effect, the business messages that drive operations need to be defined. This is generally done through an internal XML modeling effort, resulting in a set of XML Schemas defining pertinent business messages, each of which becomes the subject matter of a service. As an example, a business analyst responsible for order processing knows what data elements are required as input to the order process. Working with a data modeler knowledgeable in XML Schema, they define an appropriate structure and data set representing the order.

Many industries have defined standard business structures using XML Schema (for example, ACORD for insurance, MDDL for capital markets, HR-XML for human resources). These standards have incorporated the "wisdom of the masses" to achieve comprehensive XML-based definitions of business information. Using industry-defined XML Schemas has the additional benefit of being able to leverage well-documented business vocabularies. Since these standards are achieved through consensus, the tradeoffs include the potential for bloated messages, and the need to customize elements specific to your business. All things considered, implementing a data abstraction layer based on industry standard XML definitions leads to quicker implementations and better interoperability.

Each business message defined using XML Schema becomes a contract between a service implementation and its consumers. The data abstraction layer is simply the combination of these services. To that end, anybody that can build a service which conforms to an XML Schema can build a data abstraction layer and reap its benefits. Many tools and technologies are available to assist in this effort, and generally the selection criteria shifts to secondary considerations, such as ease of use, design time metadata management, and runtime management support.

Data Abstraction Benefits

The potential cost savings for building a data abstraction layer for your SOA fall into a number of categories. First, if you are convinced that SOA in general provides the benefits that you are looking for, it only makes sense that you'll want "SOA done right." The mandate to deliver data aligned with business definitions requires that logic be defined and exposed in a data abstraction layer. A properly defined logical layer leads directly to increased reusability opportunities, perhaps the most fundamental of all SOA-related benefits. ROI analysis has convincingly shown that

using a data abstraction layer dramatically reduces the cost of systems in the long run.

Perhaps the greatest benefits accrue due to the loose coupling achieved through a well-designed data abstraction layer. Alternate approaches of accessing data sources directly from applications, however easy development toolkits make this, inevitably lead to tight coupling and changes to the data or the application cause a ripple effect through many systems.

In contrast, the loose coupling available through a data abstraction layer provides long term cost savings by isolating many of these changes to the appropriate architectural layer. Physical data can be reorganized without affecting applications, as the "contract" between applications and data remains the same. Also, changes to the application can often be accomplished with modifications only to the data abstraction layer implementation. For example, if a new source is added to a customer's activity history, then that new source can be incorporated using the exiting formats, extending the implementation of the service without changing the interface contract.

Conclusion

As companies continue to pursue SOA benefits, implementers struggle with the fundamental mismatch between the logical data organization that a service should expose, and the diverse and often chaotic nature of data in its physical form. The solution to this dilemma is a data abstraction layer that hides the complexity of back-end data sources while providing a common information model better organized and positioned to meet the needs of service-oriented architecture.

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[Home](#) [Past Issues](#) [Contributors](#) [What is SOA?](#) [SOA Glossary](#) [SOA School](#) [SOA Books](#) [About](#) [Legal](#)

Copyright © 2006-2008
SOA Systems Inc.