



How Integration Appliances Simplify and Accelerate SOA Implementation

By Fred G. Meyer

Abstract

Service-Oriented Architectures (SOA) were designed to make life simpler, but many SOA implementations are becoming more complex than the architectures they are replacing. The key to a successful SOA implementation is to minimize complexity at every turn. In this paper, Fred Meyer, CEO of Cast Iron Systems and the former Chief Strategy Officer of TIBCO Software, shows how a “back to basics” approach to architecture design can be combined with *integration appliances* to drive down complexity and accelerate Service-Oriented Architecture implementations. Specifically you’ll learn:

- The three core drivers that led to the birth of the Service-Oriented Architecture
- How to drive complexity out of your SOA
- How to select the optimum mix of technologies required for a successful SOA
- How integration appliances cut the time, cost and complexity of SOA construction

Introduction

Integration professionals have realized that point-to-point integration dramatically increases the complexity of building and maintaining corporate business processes. One manifestation of this is the increasing popularity of Service-Oriented Architectures. In order to achieve maximum benefit from SOAs, it is important to carefully consider architecture and design principles before choosing technology and going to work on practical examples.

In a properly factored design, integration appliances can greatly reduce cost while simplifying implementation and ongoing maintenance. Appliances can also help insulate companies from the inevitable changes common to any integration system by minimizing supporting infrastructure.

Central Concepts of the Original SOA

In the mid-90’s, commercial integration packages were being used to build large, cross-system business processes for the first time. However, the limitations of first generation technology became a serious barrier to success:

- 1) Integrating every system to every other system results in complexity which grows as the square of the number of endpoints. This clearly will not scale. The introduction of hub and also bus models reduced these connections, with all of the data flowing through a central hub and repository.
- 2) Once all the data for a given system was flowing through a central point, designers could think about reducing the amount of work required to maintain the endpoints. Even moderately complex systems have dozens of endpoints, and the bulk of the IT

administration effort involves maintaining connectivity as the endpoints change. SAP comes out with a new version of software, Oracle buys PeopleSoft, the mainframe payments system moves to a UNIX server, or vice versa.

Early integration systems were usually tied directly to low-level API's at the endpoints. This was done by using adapters. Unfortunately, standards were few. People forget today that in 1995, even ODBC didn't always work consistently. This resulted in large numbers of adapters having to be co-located with the target systems – at least one per application type and often more, to deal with multiple versions of an ERP system. Every time the endpoint changed, that interface had to be reworked. Worse, all the business logic behind it often was tied tightly to the adapter and had to be rewritten or ported. This led to the idea of a higher-level abstraction than a programming API.

- 3) The concept of using abstract services, rather than low-level APIs, was an improvement. But it, too, has a serious weakness. The number of interfaces that are services is really not any smaller than the number of physical connections to end systems. By creating a service, you get a better abstraction—but there are still too many of them. Worse, it is often the case that a well-designed service touches more than one underlying physical application. This creates the potential for overlapping services, data duplication and data inconsistency. Again, the design does not scale.

These are the central concepts of the original SOA: Simplified connectivity, service abstraction and data mastering. Unless all of these ideas are present in your SOA design, you will not derive the maximum benefit from the resulting system. Conversely, a true SOA not only insulates you from underlying change, but allows you to reuse the services. Consider the requirements of a retail bank. They need similar services on the web, for their call center and in their physical branches (they call this a *multi-channel architecture*). The SOA gives you double benefit: It protects you, with an abstraction layer, from changes in the underlying systems; and it abstracts client side applications from server side systems. The savings in reimplementation costs can be enormous over the life of such a system.

The Key To SOA Implementation: Divide and Conquer

An SOA can be implemented in many different ways, using a variety of tools and technologies. However, one additional insight may help you to minimize the cost and implementation effort of your SOA.

At the genesis of the SOA, one overriding trend is apparent. Integration tools were originally undifferentiated: One tool for every connection to every system. As the SOA concept evolved, the architecture broke into categories with different characteristics. At the highest level these are:

- 1) Applications: The underlying programs that actually do the work.
- 2) Directories: Common services such as entitlements, service directories, etc. These plug into the service layer in an SOA, to insulate the overlying business process from changes.
- 3) Data synchronization: The integration components used for building and maintaining master/slave data relationships
- 4) Business process layer: The layer that makes use of services and embodies those business processes which span multiple services – meta-business processes, the stuff of BPM and workflow.

These components have different functions and very different structures. So the last insight is this: They are often best constructed using different technologies.

Selecting The Right SOA Technologies

The complexity of an SOA project can be greatly reduced by choosing appropriate tools. You probably wouldn't build your purchase order system on top of LDAP. You also shouldn't use a BPM tool for data integration.

Good BPM tools are very expressive and abstract. They think in terms of end-to-end business collaborations, which often involve multiple individual transactions done by humans. BPM tools are ideal for building high-end workflows containing complex business semantics, implementing best practices across multiple groups of users. They are also complex to learn, difficult to implement and costly to maintain. If you really need what they have to offer, however, they are worth it.

In situations without complex manual workflows, however, an appliance is often more cost-effective, quicker to implement and easier to maintain.

Integration Appliances In SOA

Even the best planned SOA project has its complexities and requires a broad skill set to implement and maintain. The thinking which led originally to the SOA was focused on reducing that complexity. Complexity reduction is the only sure way to reduce cost and improve the chance of project success. Complexity reduction pays dividends each time a change is made anywhere within the SOA, as the number of moving parts affected by the change is directly related to cost, time and difficulty of implementation.

Appliances are useful because they are simple. They require minimal support infrastructure, are highly repeatable and secure. An appliance that can be accessed only via the network can, by definition, be configured, operated and maintained from anywhere with a secure network connection.

Appliances are simple because no infrastructure preparation is required beyond electricity and an IP connection. Software solutions require infrastructure composed of host hardware, network hardware, an OS, JVM, patches, RDMS, security software, management software, etc. Since the software manufacturers are constantly 'improving' their products, the odds that any set of infrastructure servers, however carefully standardized, are actually identical is almost zero. Appliances, on the other hand can easily guarantee form/fit/function to be identical because all the software and firmware in the box is under a single release control program. They are, in effect, one large software, hardware and firmware distribution.

Because appliances are not general purpose, their components can be optimized for the task they perform. Similarly, there is no need to provide user-level access to internal components, such as RDMS, RAID arrays, OS functions, etc. These come preconfigured and can be managed by the appliance itself. All the user needs to do is to configure the device for exactly the task it must perform.

As network devices, appliances can be placed wherever they are needed and easily managed and upgraded. Local IT presence is not required. Indeed, many appliance users manage worldwide implementations from a single center of excellence.

Appliances fit three places within the SOA: They are the most logical solution for data integration, they are often the best way to integrate various directory services and can be used to construct the services exposed to the workflow/BPM layer:

- 1) Data integration. By avoiding the complexity of manual process integration, appliances can also avoid the bane of every integration project: Adapters. Adapters allow rich interactions with endpoints and users at the BPM level. They are also hard to maintain, hard to upgrade, expensive and, by definition, every type of adapter is a thing unto itself. With recent improvements in standards, they are usually unnecessary. In simple cases,

direct data integration suffices. In more complex cases, the service can often be built directly on the target applications. Solutions that use adapters don't scale.

There may still be a few cases in which the use of adapters is unavoidable. Data integration is not such a case. By combining the advantages of an appliance with the elimination of adapters, the master data integration scheme described above becomes simple and cost-effective enough for real projects.

- 2) Directory services. Directories for entitlements, service discovery, etc., need to present their information in a single format through a unified interface. Though simple in theory, this is almost never the way real corporate directories look and act. This is really a data integration problem in disguise. As such, an appliance can be a very good solution for normalizing directory access.
- 3) Service presentation. The complexity of a service is intermediate between that of a data interface and an application. A properly designed data integration appliance is not suitable for manual workflow, but is capable of holding and representing enough state (via BPEL, for instance) to serve as an intermediary between applications and the workflow/BPM layer. Since the most common place from which to access a directory is the service layer, it is possible to build a uniform and easily managed infrastructure that abstracts all but the services themselves away from the developer of business process.

Conclusion

The success of a Service-Oriented Architecture depends on how well complexity has been reduced. Construction of large, robust integration projects that have a long, maintainable lifetime requires the architect to minimize complexity at every step. Rich, agile business process is where the value of the completed integration project lies. By forcing complexity out of the service layer, integration appliances allow more resource to be applied at the business layer where the ultimate payback for the company lives.

Contact Us

To learn more about the Cast Iron Systems Application Router, please call us at 650.230.0621 or visit us online at www.castironsys.com

© 2005 Cast Iron Systems. All right reserved.

This document is provided for information purposes only and the contents are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. Cast Iron Systems™, Cast Iron™, and Application Router™ are registered trademarks of Cast Iron Systems. Other names may be trademarks of their respective owners.