

The SOA Magazine

Feature Article



SOA and Composite Applications

by Robert Schneider

Published: April 3, 2007 (SOA Magazine Issue VI: April 2007, Copyright © 2007)

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

Abstract: Composite applications (also known as “enterprise mashups”) deliver immediate, cross-system interactive value to users while showcasing the value of your investment in a service-oriented architecture. Today’s standards-based infrastructure and powerful development tools make these applications possible. Composite solution design is particularly compelling in an environment standardizing on SOA.

This article explores the synergy between the concepts and principles associated with composite applications and the service-orientation design paradigm. Further aspects of composite design are discussed and explored, along with guidelines that address security, reliability, and performance considerations.

Introduction: Software as a Service (SaaS) and Integration

Way back when, all applications were hosted on large, expensive mainframes, residing behind glass walls and tended to by men in white coats (and guarded by watchmen with guns). Eventually, with the rise of minicomputers and then personal computers and servers, hardware and the software systems they supported came out from behind the glass.

If we fast forward 40 or 50 years and look around, it appears we have come full circle: many new applications are once again resident in formal data centers. The main difference now is that these new solutions (now called “hosted applications” or Software as a Service) are accessible from anywhere you can obtain an Internet connection. Often available via subscription with minimal investment, they have helped to dramatically reduce the total cost of ownership for their users. However, along with that benefit comes new challenges, particularly in the area of application integration and especially with existing solutions residing behind firewalls.

Obstacles: Regulatory and Otherwise

Integration, while never a cinch, is much easier when all the applications are hosted inside the firewall. To keep these systems in sync, enterprises can draw on a wide array of technology. Ranging from extract, transform, and load (ETL) through data warehousing through enterprise service buses (ESBs), at their core these methods all involve copying data across silos. However, when it comes to integrating with SaaS-based solutions, there are a number of reasons why this approach can run into trouble:

- *Regulatory and Legal Obstacles* - Governments and other regulatory bodies often place highly restrictive rules around (especially customer-centric) data. Consequently, your enterprise may not be able to duplicate internal information onto a remotely-hosted SaaS-based system.
- *Organizational Policies* - Even if you encounter no regulatory or legal issues, your enterprise may not be willing to place sensitive internal information onto servers hosted by an external organization.

- **Data Volume** - This is the ultimate impediment – the sheer amount of data owned by your organization might outstrip the available capacity of the hosted application or unreasonably tax available bandwidth, rendering any type of data duplication strategy impossible.

Now that you've ruled out copying data onto the hosted solution, your next choice is to require that the user launch each application in parallel (and simply Alt-Tab among the systems). This is a band-aid at best, placing a heavy burden on the users who will be required to make real-time decisions on how to relate information across applications. That's asking a lot, especially when you consider that it may be difficult, if not impossible to perform these kinds of data gymnastics in real-time.

The Composite Approach

What if there was a way to present a single, unified view of all relevant information to your users? What if it was done in real-time, securely, and through a single user-interface?

Luckily, it is possible to do just that via a new style of application integration and development known as *composite applications*. Also frequently described as *enterprise mashups*, these new technologies remove complexity from users' lives while granting them significant added power and insight.

If you're familiar with portals, then you already have a head-start on understanding what composite applications are and how they work. In a nutshell, they typically render a single user-interface that spans multiple data silos in real-time. In many cases, the composite application is launched from within a "master" applications user-interface. This makes it appear that the master application has been extended with loads of new functionality when in fact this is all handled behind the scenes, by the composite application.



Figure 1: A portal that presents a unified and federated perspective provided by a back-end comprised of a composite application.

For example, look at Figure 1. It shows the user-interface of a hosted CRM package with a composite application embedded within the Account page. The composite application renders, at runtime, important information from other applications, such as order management, inventory control, financial data, and so on. Unless you had strong expertise in the CRM package, you would never know that three or four systems were being consulted to present this data. And, unlike a portal which frequently requires information to be copied and synchronized into a centralized data warehouse,

the composite application retrieves its rolled-up information on a just-in-time basis: users see live data.

Another key difference between an enterprise mashup and a portal is that the former often sports built-in semantic interoperability (i.e. logical cross-silo relationships) while the latter typically requires users to make that mental leap on their own.

Figure 2 drills down into the user-interface. In this example, you can see how the user remains within one HTML page, yet seamlessly interacts with data from multiple systems. Observe how the composite application pulls information in real-time from multiple silos (both internal and external) and coordinates the data into a single user-interface. Of course, this is just one way of deploying such a solution.

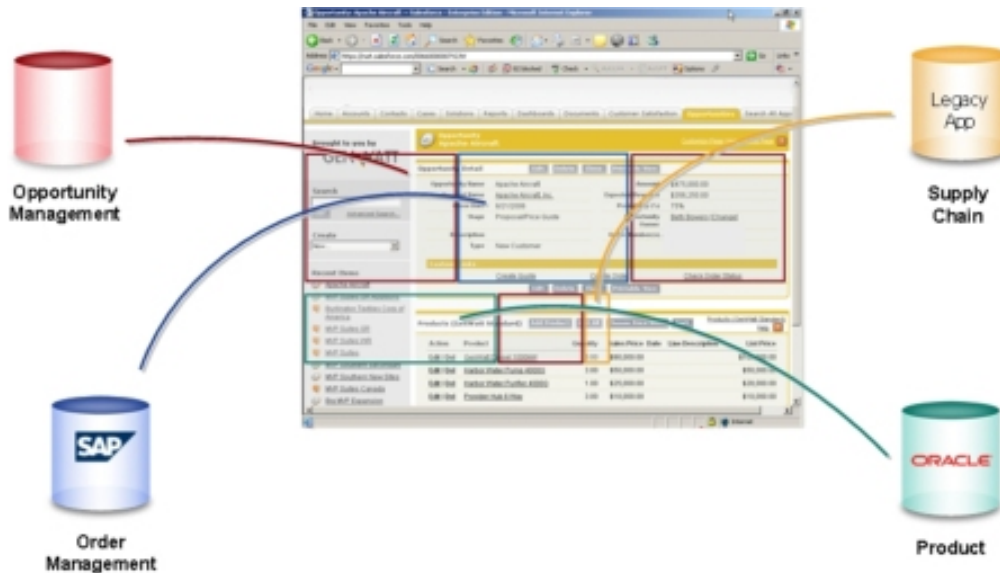


Figure 2: The same portal page displayed in Figure 1, only its underlying data sources are revealed.

Let's take a look at some more qualities of a composite application.

General Characteristics of a Composite Application

While it would be difficult to get any two developers to agree completely on what exactly constitutes a composite application, here are some general characteristics that are commonly accepted:

- **Real-Time Access** - There's not much to gain if the user is forced to look at stale data from other silos. For example, imagine the incorrect decisions that might be made on a sales call if the user thinks that the customer's account has been sent to collections when in fact all past due invoices were paid three days ago.
- **Minimal Cross-Silo Data Movement** - Leaving data in its native silos is one of the key rationales behind a composite application. However, there will be the odd occasion where it's necessary to copy foreign key or other cross-referential information into varied silos.
- **Transactional** - This is one of the more controversial (and hence optional) traits of a composite application. If the business logic and rules of the underlying systems permit it (and if the user has the authority), then transaction-based interaction is theoretically possible through a single user-interface. However, as I'll soon describe, most organizations choose to deploy transactional composite applications in a second or third phase.
- **Just-in-Time Integration** - One common concern about composite applications is their performance impact. However, it's important to realize that no resources are consumed until the user makes a request, typically via the user-interface. For example, you may have structured your enterprise mashup to return information about a customer's transaction history from an ERP system. In most cases, though, the ERP system will not be contacted until the user requests this specific information.
- **Semantic Interoperability** - This characteristic refers to the ability of the enterprise mashup to understand and deal with the data relationships among all of the applications. For example, the composite application would need to know that a customer number in the CRM package translates to an account identifier in the billing system. This is the primary reason to keep cross-referential information in sync across different silos.

Why Composites? Why Now?

Developers have been building applications for decades; why is it only recently that composite solution have gained acceptance?

Here are some of the driving factors:

- *Transport Layer Standards* - Earlier attempts at broad application interoperability, such as CORBA, relied on the widespread adoption of often-arcane transport layer protocols. Now that HTTP(S) and TCP/IP have gained universal acceptance, the enabling wire-level infrastructure is in place everywhere.
- *Application Vendor Support of Web services* - Back in the not-so-good old days, each application vendor could be counted on to provide their own proprietary API or integration technology (along with busloads of expensive consultants to milk these cash cows). However, by-and-large these vendors now offer Web service interfaces to their solutions, making cross-system composite applications much easier to develop. These new services also help solve the firewall challenge inherent in many Internet-era integrations.
- *Service-Oriented Design Philosophies* - Just as software vendors have done their part to ease integration, so has the widespread adoption of robust service-oriented design methodologies. As I describe later, composite applications leverage many of the key underlying principles of service-orientation [REF-1]. In fact, numerous organizations use composite applications to highlight the benefits of investing in service-oriented architectures.
- *Sophisticated Technology Stack* - In a few moments, I'll cite an example of a Microsoft-oriented technology stack that makes composite applications much easier to develop. In the absence of such a stack, developers have a heavier burden to shoulder when attempting to construct an enterprise mashup.
- *HTML user-interfaces* - Finally, all of the above factors would have much less impact if there was no easy way to deploy a composite application. Modern technology has helped us out here as well, specifically with the ascent of HTML-based user-interfaces. These types of front-ends are much more flexible and configurable than the previous generation of client-based GUIs, and easily lend themselves to incorporating composite application designs.

Challenges to Overcome

Even with new technologies and design philosophies things are never quite as simple and streamlined as they might seem (or we might hope!). Here are several key obstacles on the road to building composite applications, along with some guidelines as to how they can be surmounted:

- *Semantic Interoperability* - It's hard to construct a good composite application when you have no way of relating data across silos. To get around this problem, many enterprises compose higher-level Web services that span multiple silos. These new, abstracted services then form the foundations of composite application architectures.
- *Logins and Permissions* - Successfully maintaining a single application's security infrastructure can be daunting enough. The mind boggles at the thought of keeping three, four, or more applications all in sync with regard to security. Luckily, this is not as difficult as you might imagine. A little later in this article, I'll discuss some ways to simplify a cross-silo composite application's security footprint.
- *Latency* - For many system and data administrators, the first thing that comes to mind when evaluating a composite application is the impact that its cross-silo nature will have on performance. As it turns out, a well-designed composite application has little or no added latency costs. In fact, it's reasonable to assume that it will save time for both users and data servers. I'll cite some best practices momentarily.
- *Lack of a Service-Oriented Infrastructure* - It's surprising how many application developers never even attempt to construct a composite application because they assume that a full-blown service-oriented architecture needs to be in place. As I cited in my earlier article on Web service-enabling relational databases [REF-2], there are various ways to phase into a service-oriented architecture without requiring a major up-front investment.

Composite Applications and Service-Oriented

You might wonder how a user-interface-driven solution such as a composite application would have anything to do

with the common principles of service-orientation. As a matter of fact, a well-designed composite application leverages, reflects, and relies on many of these established design principles [REF-1]:

- *Service Reusability* - While you won't typically reuse a composite application (since it's been developed with a specific purpose in mind), hopefully you'll be able to re-employ many, if not all, of the underlying services for future solutions.
- *Standardized Service Contracts* - This is one area where you need to be really careful. It's one thing to flippantly alter a service contract when a service only interacts with other services. However, it's a different story when a user-interface (with a person driving it) is relying on the service.
- *Service Loose Coupling* - The best designed composite applications serve as aggregation vehicles, grouping a collection of loosely coupled services into a meaningful, business-driven solution.
- *Service Abstraction* - Ideally, you'll be able to compose and/or aggregate your business logic into its own set of higher-granularity services which are then leveraged by the composite application. The alternative is to encode this logic into the composite application itself, which can make reuse by other applications more difficult than you probably want.
- *Service Composability* - By definition, a composite application serves as the end result of composing a collection of granular services into a well-defined, user-oriented solution.
- *Service Autonomy* - While the core services that drive a composite application are autonomous, the application itself serves to aggregate these services into a value-added package that provides semantic interoperability among the silos.
- *Service Statelessness* - Read-only solutions are the typical starting point for a composite application. Consequently, these applications are stateless by design. As developers begin to construct applications that modify information in native silos, they'll rely on the state maintenance capabilities of the underlying services.
- *Service Discoverability* - From the users' perspective, this is somewhat less relevant, since they'll be interacting with an already-constructed solution. However, the development tools used to build a composite application will certainly leverage these capabilities.

Now that you've decided to deploy a composite application to solve your initial integration challenge, here are some things to keep in mind as you begin your journey.

Choosing a Development Technology Stack

Putting together a workable technology stack to build and maintain composite applications is not as difficult as you might fear. As with most software projects these days, there are at least two competing schools of thought: Microsoft and open source software. For this article, I'll use Microsoft as a reference:

- *user-interface* - WebForms and Extensible Application Markup Language (XAML)
- *Web Services* - ASMX or Windows Communication Foundation (WCF)
- *Database* - ActiveX Data Objects (ADO)
- *Workflow* - Workflow Foundation (WF) and BizTalk
- *Development Environment* - Visual Studio .NET

Note that 1) there isn't enough space to describe how to deploy each of these technologies, and 2) there are open source counterparts to each of these layers.

Designing for High Performance

As I mentioned earlier, it's quite natural for administrators to see composite applications as performance hogs, driving down throughput for all the systems that they touch. However, this needn't be the case. Here are three simple steps you can take to keep your applications speedy:

1. *Use Indexed or Other Optimized Queries* - For example, if your composite application makes calls to a financial system to retrieve account history and other information, make sure that the invoked services leverage indexes or other performance-related technologies offered by the underlying application.
2. *Let Users Determine When to Retrieve Data* - Many over-zealous developers construct their composite applications to retrieve all possible data when the user-interface initially loads. However, this imposes a multi-system performance hit that may not be necessary. For example, suppose that the user doesn't want or need to see cross-silo data for a given record, instead focusing on the data contained in a single system. Why not let the user decide when, if ever, to query secondary and tertiary applications?
3. *Don't Allow "Fishing Expeditions"* - One guaranteed way to bog down an application (composite or otherwise) is to let users have unlimited query power. This becomes especially dangerous when deploying a composite solution, since poor query hygiene can infect multiple silos at one time. It's far better to force the user to zero in on a record in the primary application and then use that record's key information to retrieve data from additional silos.

Mashing up Safely

Security is often (and justifiably) a serious concern for composite application developers and administrators. It's natural to fear unauthorized access to, (or worse, alterations to) secondary systems. Thankfully, you can mitigate your risk in several ways.

- *Deliver Read-Only Applications* - This is a natural first step for most developers. While it doesn't eliminate the potential for unauthorized access, it does prevent the possibility of unwanted data modifications.
- *Rely on the Primary System's Security Model* - A typical composite application is launched from within a primary system. For example, in this case the composite application would be launched from inside an instance of a customer record within the CRM package. You can assume that if users are seeing customer records from the CRM package that they should be able to see related financial and other details from secondary systems.
- *Use a Single Security Repository* - If you have the time, money, and discipline, this is probably the most effective technique to enforce a consistent security policy across all underlying systems. It's also an ideal infrastructure to support composite applications.

Conclusion

Composite applications share great deal of commonality, both conceptually and technology-wise, with service-oriented solutions. In fact, applications comprised of services can be legitimately qualified as composite. The term "composite application" is sometimes associated with specific application characteristics (such as federated user-interfaces); however, as evidenced by the previous comparison of composite design and service-orientation principles, the synergy between composite and service-oriented applications is undeniable. We only need to remind ourselves that one of the fundamental mantras of service-oriented design is that services must be inherently composable.

References

- [REF-1] "SOA: Principles of Service Design", Thomas Erl, Prentice Hall/PearsonPTR, 2007.
 [REF-2] "Web Service-Enabling Relational Databases for SOA", Robert Schneider, The SOA Magazine, November 2006.