

The SOA Magazine

Feature Article



SaaS, Composite Applications, and SOA: Understanding their Differences and Making Them Work Together

by Robert D. Schneider

Published: July 5, 2007 (SOA Magazine Issue IX: July/August 2007, Copyright © 2007)

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

Abstract: Software as a Service (SaaS), Composite Applications, and Service-Oriented Architecture (SOA) are three distinct platforms, each of which is garnering a tremendous amount of coverage in the technical press. In this article, I describe these concepts, compare and contrast them, and finally show a unifying vision where each has an important role to play in a progressive-minded IT organization.

Introduction

Aside from the media buzz, at first glance SOA, composite applications, and SaaS appear to have very little in common. As it turns out, however, increasing numbers of organizations are coming up with innovative ways to combine the unique strengths of each of these technology platforms to produce new types of solution environments capable of enhance the user experience while increasing the organization's overall ROI and responsiveness.

To illustrate the examples in this article, I refer back to Pay-N-Pray Motors, the deep-discount car rental company I introduced in my previous two SOA Magazine articles ([Web Service-Enabling Relational Databases for SOA](#) and [SOA and Composite Applications](#)). Despite their relentless focus on cost-cutting, as you'll soon see Pay-N-Pray Motors has a very progressive IT department, one that is constantly investigating new ways to deliver value to their users.

What is SaaS?

Back in the late 1990s, venture capitalists showered buckets of money on a new class of technology company: the Application Service Provider (ASP). The primary mission of many ASPs was to host some of the better known enterprise software packages that had previously been installed in the customer's data center. Going forward, the customer was still responsible for purchasing the software, while the ASP then acted as an outsourced hosting department and charged either by user or by deployed application. End users would then connect to these applications via either a browser, or more likely the same PC-based rich client software they always had used: from the users' perspective, the ASP simply provided a different endpoint. At the time, the expectation was that many large enterprises would gladly pay someone to take the expensive and tedious burden of running and maintaining heavyweight enterprise software off their hands.

Unfortunately, the combination of the .com implosion, enterprise software vendor resistance to new business models, and general customer skepticism crippled this nascent industry. Many ASPs shut down, customers lost vital data, and the venture capitalists were forced to downgrade from proudly driving Ferraris to skulking about in Porsches.

However, just as the downfall of the plodding dinosaur soon saw the ascent of the nimbler mammal, a new class of vendor sprang up from the wreckage of the ASP. Rather than simply host someone else's bloated packaged application, these new providers built newer, more focused and light-weight solutions. In addition, the primary means of accessing and maintaining these new applications was via thin client browsers, which translated to much less work

to get a customer into production. The selling model was also simplified, with most customers paying a simple monthly subscription per user or instance. This approach has proven very successful; now that high quality, high-speed Internet connections have become commonplace, many organizations (large and small) have opted to let someone else shoulder the burden of developing and maintaining critical enterprise applications.

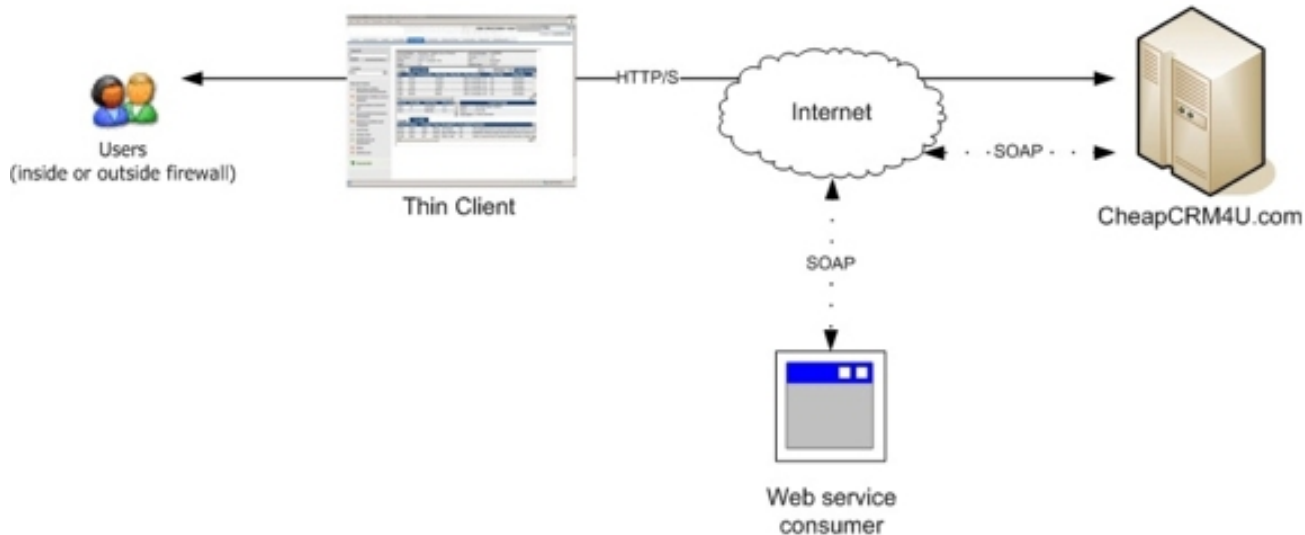
Software as a Service (SaaS) is the commonly accepted name for these new application providers. While each has their own unique value proposition and positioning message, they generally have the following traits in common:

- The software is developed, hosted, and supported by the vendor.
- Access to the software is via a browser.
- Administration and maintenance of the software is also handled remotely via a browser.
- Web services act as the primary API for the solution (for customers interested in real-time integration).

From an SOA perspective, what makes SaaS so interesting is that, by and large, these vendors "get it": their offerings are much more SOA-friendly, and often comply with many of its best practice guidelines. Of course, there are some horrendous exceptions to the rule, but SOAP-based, WSDL-defined services have become the de-facto standard for integration with SaaS solutions. Many SaaS vendors also understand the value of composite applications, and are only too happy to facilitate deploying these user interface-based integration components.

Pay-N-Pray Goes SaaS

Turning our attention back to everyone's favorite deep-discount rent-a-car shop, it seems that Pay-N-Pray Motors has seen the light, and will be deploying a deep-discount SaaS solution: CheapCRM4U.com. This new vendor, owned by the golfing buddy of Pay-N-Pray's CEO, offers a suite of hosted Customer Relationship Management (CRM) programs. This software is accessible from anywhere – inside or outside the firewall. In addition, they also offer a full set of Web services that enable developers to interact with their solution. Figure 1 shows a high-level view of this architecture.



*Figure 1:
An overview of
an SaaS
architecture.*

The diagram illustrates how users (no matter where they might be) interact with the CheapCRM4U.com SaaS solution. The user interface is displayed in a browser, meaning no software is installed on the users' machines. Communication is secure, via HTTP/S. At the same time, Web service consumers can communicate, via SOAP, with services offered by the vendor. This lets developers at Pay-N-Pray write applications that work with their data, even though it's hosted by the CRM vendor.

What is a Composite Application?

The widespread adoption of standards-based technologies such as thin client applications, SOA, and Web services has fueled new and innovative ways of tackling the age-old challenges of information integration. Enterprises of all shapes and sizes are increasingly bypassing the traditional strategies of synchronizing massive amounts of data

between closed silos. These approaches had traditionally required significant investments in specialized software, process definition, trained personnel, and custom application logic to deliver value. Instead, these organizations are turning to composite applications to provide real-time, on-demand integration via the user interface. Also known as "enterprise mashups", these new classes of solution are much lighter weight in nature, and generally do not require any data movement among disparate systems. They're focused on solving the long-running problem of providing consolidated, cross-silo information to the end user.

For example, users may spend most of their time in a CRM application. Periodically, they may need to look up or modify data from another application such as an ERP package. Before the advent of composite applications, the user in essence had four choices:

1. *Launch the ERP package and then navigate to the appropriate record(s).* This task required that the user be trained on both the CRM and ERP solutions, have appropriate logins and permissions for both, and also have cross-system semantic understanding of how data is related.
2. *Navigate to a custom object in the CRM system to view data from the ERP system.* Of course, this assumes that someone in the organization has taken the time to develop and maintain a data synchronization-style integration between the two systems.
3. *Contact a peer who has access to the ERP system, explain the situation, and wait for an answer.* This process could take anywhere from minutes to infinity.
4. *Do nothing.* As just about anyone who has dealt with a customer service department can attest, this option is depressingly common.

A composite application is a much more productive and elegant solution than the above 4 tactics. With a composite application in place, the user simply remains in their primary application and makes requests for any external enterprise data in real-time. In fact, this data is often gathered and available even before the user makes the request. To the user, it appears that the primary application has been magically "extended" to include data from other silos, when in fact this remote data remains safely ensconced in its primary database. Modern composite applications further leverage standards-based communications technologies (such as Web services) to seamlessly work through firewalls.

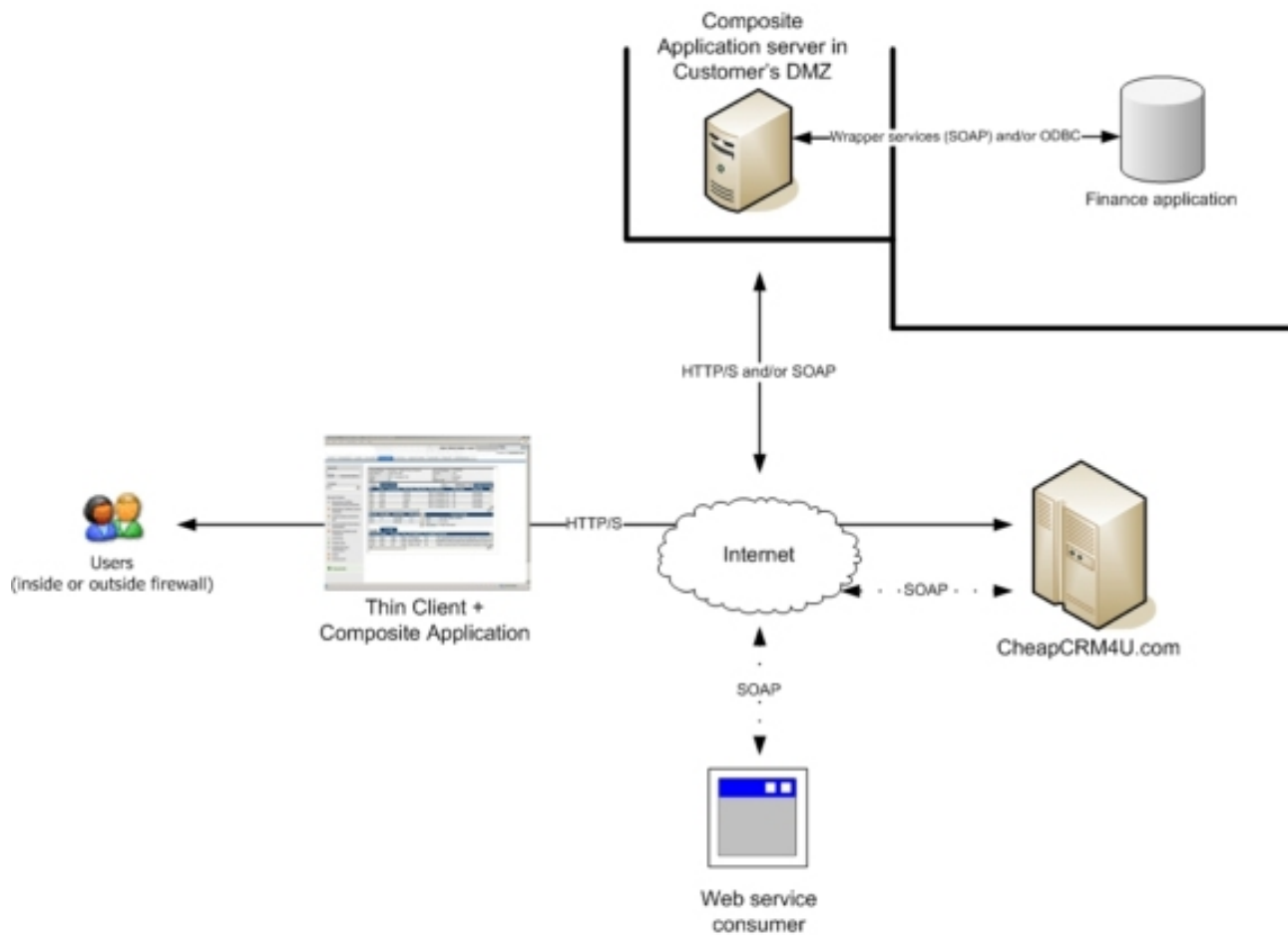
Composite applications are a natural add-on for SaaS applications. Many organizations can't or won't copy massive amounts of internal data into an SaaS provider's database, which means that the only way for the user of an SaaS system to see this data is to view it in real-time. The composite application often simply occupies some screen real estate in the SaaS vendor's user interface. Fashioning a composite application these days is relatively straightforward, and the availability of open source-based software (such as Ruby on Rails) is making this platform increasingly affordable.

Marrying SaaS and Composite Applications

With the CheapCRM4U.com SaaS offering successfully deployed, things are going very smoothly at Pay-N-Pray Motors. However, all good things must eventually come to an end, and Pay-N-Pray is no exception. Apparently, an executive in Finance decided to institute a retroactive 3% 'courtesy fee' for all rentals that occurred in the past 2 years. All credit cards on record are duly charged the additional 3%, and details about these transactions are stored in the company's packaged, self-hosted financials system. Many customers never even notice the new charges placed on their credit cards. Unluckily for Pay-N-Pray, though, a popular blogger was one of the affected consumers, and published the details on his website. Traditional media soon picked up the story, and as you might expect, the phones are ringing off the hook at Pay-N-Pray. The besieged customer service representatives (who spend all of their time in the CheapCRM4U.com application) have no easy way of viewing these transactions, and it's even harder to reverse them.

IT management reacts with uncharacteristic speed, constructing a cross-system composite application in a matter of days. Meant to be launched from within the HTML-based CheapCRM4U.com user interface, it presents a unified view of a given customer's contact detail (from the CRM package) and billing history (from the financial software). It even allows the customer service representative to make a note of the client's dissatisfaction in the financial package. However, actual refunds will require the customer to fill in, by hand, a printed form in triplicate and mail the original car

rental receipts back to the company.



*Figure 2:
How a SaaS
and a
composite
application
can work
together.*

As shown in Figure 2, the SaaS application continues to function as before. However, the new composite application logic, which is hosted by the customer, interacts with the packaged financial system. This contact can occur in a number of ways, from new Web services that wrap the packaged financial software vendor's API, to ODBC calls, to a combination of techniques. Of course, all of this behind-the-scenes action is hidden from the user. From their perspective, the CheapCRM4U.com user interface now has been extended to provide details from Pay-N-Pray's financial system. It's important to note that the financial data never is stored outside the financial system; the user simply views it (and optionally makes modifications) in real-time from within the CRM user interface via the composite application.

What is SOA?

You're probably reading this article because you're interested in Service Oriented Architecture. It's also likely that you already know quite a bit about the topic. If not, then while there are many ways to describe SOA, it's probably simplest to think of it as a technology architecture philosophy whose goal is to help organizations construct and then employ a set of well-designed services as the foundation of their data processing world. By taking the time to correctly deliver on this approach, these enterprises are able to realize tremendous productivity, efficiency, and agility benefits that more than justify the required investments. There are a host of technologies and products at your disposal when implementing your SOA vision. What's important to note is that regardless of your chosen tools, a well-thought-out SOA strategy delivers on the following service-orientation design principles [REF-1]:

- *Service Reusability* - A service should be reusable for other tasks by other consumers.
- *Standardized Service Contracts* - A service should present a well-defined contract to help consumers understand how to interact with it.
- *Service Loose Coupling* - Services should evolve and work independently, yet still interoperate correctly.

- *Service Abstraction* - A service should hide, or abstract, its inner workings from the outside world.
- *Service Composability* - An individual service should be able to be grouped, or composed, into larger units of work.
- *Service Statelessness* - A service should strive to maintain as little state information as possible, for as brief a time period as possible.
- *Service Discoverability* - A service should provide enough meta-information to be locatable by potential consumers.

It's important to note that not every service will fully deliver on all of the above design principles. For example, a given service may have only one purpose, and may therefore not really be reusable, while another may need to maintain state for extended periods. However, in any case, a key success factor to achieving successful service implementations in support of a larger SOA initiative is that these principles are consistently applied to whatever extent feasible.

Making the Investment in SOA

Back at Pay-N-Pray Motors, both the SaaS CRM solution and its associated composite application have been delivering real value to the user community. IT management decides that it's now time to begin making a true investment in SOA. The design principles listed above serve as the cornerstone for this work. Utilizing a top-down approach, numerous core information assets now feature well-designed services as their primary interface. This lets the IT team build internal applications more efficiently, and help improve the organization's competitiveness and overall agility.

How Do SOA, SaaS, and Composite Applications Relate?

Now that you've seen concise definitions and use cases for each of these technologies, it would be natural for you to wonder how they rate on common criteria. Table 1 highlights where each stands in relation to a collection of considerations. Note that these classifications are somewhat fuzzy and open to interpretation. Their purpose is to simply provide valid generalizations about the strengths, weaknesses, and relative costs of each approach.

	SaaS	SOA	Composite Application
Strategic or tactical vision	Both	Strategic	Generally tactical
Scope	Departmental	Departmental or Organization-wide	Departmental
Purchased/chosen by	Business users in tandem with mainline IT	Business Analysts and IT Architects in tandem with mainline business users and IT	Mainline IT in tandem with business users
Speed to deploy	Very fast	Slow	Relatively fast
Technical skills required	Low	High	Medium
Market penetration	Medium	Low	Very low
Speed to ROI	Relatively fast	Slow	Very fast
Eventual total ROI	Medium	High	Low
Expected longevity of a solution	Medium	Long	Short
Standards-based	Somewhat (primarily for HTML UI and Web service API)	High	Low

Maturity of vision

Relatively mature

Relatively new

Very new

Tying It All Together

At Pay-N-Pray, IT leadership decides that it's time to take advantage of the benefits of SOA, SaaS, and composite applications. Figure 3 shows how they've managed to incorporate all three of these technology concepts into a unified vision.

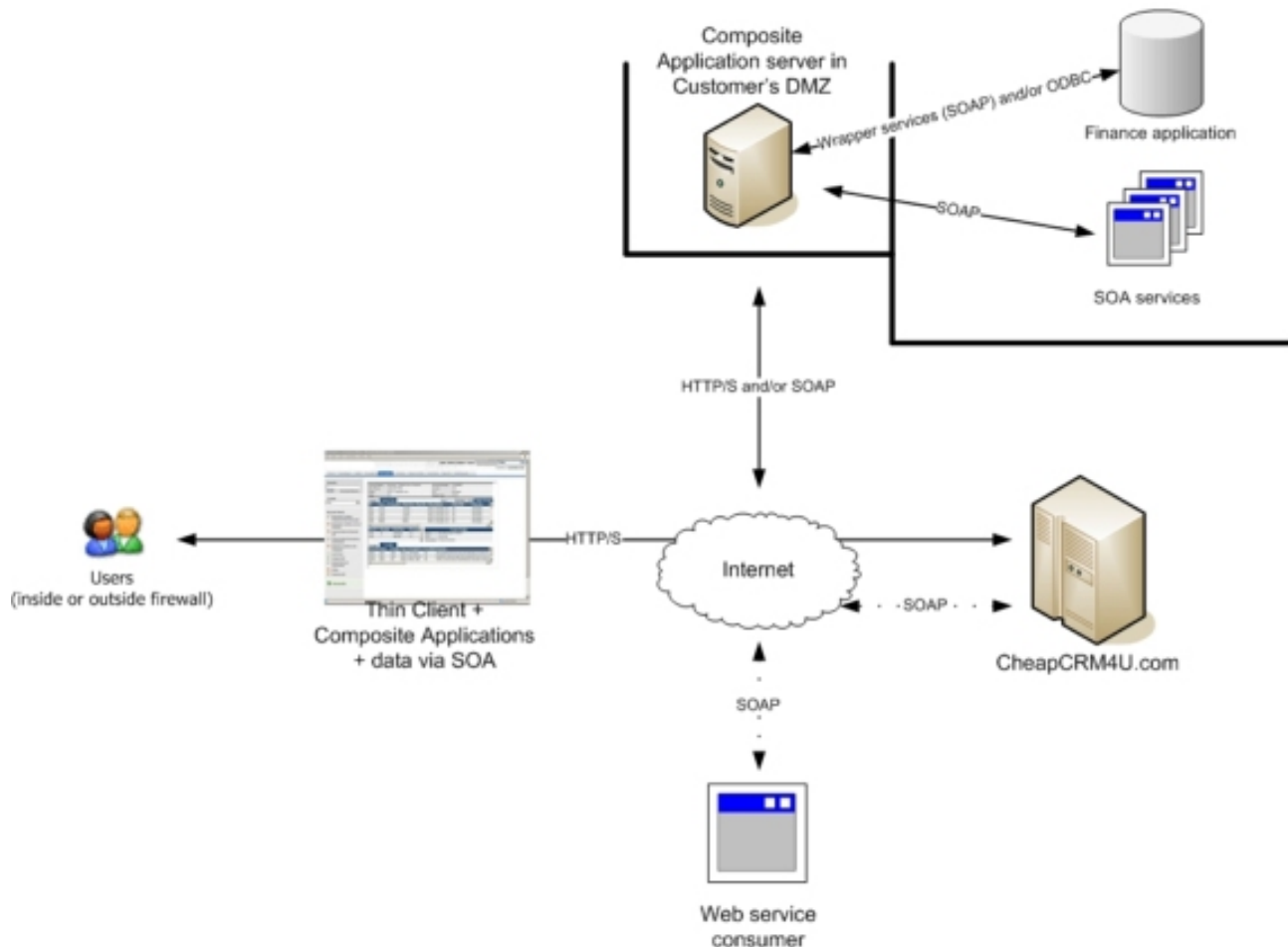


Figure 3: A unified architecture comprised of SOA, SaaS, and a Composite Application.

The end users continue to do their daily work inside the CheapCRM4U.com user interface. They also use the composite application whenever they need to interact with data from the financial application. However, as just described, there are now a set of well designed services that comply with SOA best practices. These new services are adding value throughout the organization, making their presence known in machine-to-machine transactions as well as human-to-machine interactions. For the benefit of customer service agents, a set of new composite applications are now in place. These applications leverage all of the hard work that went into designing the new services, and provide copious new functionality that boosts everyone's productivity. As an added bonus, these services are also available for many other purposes, too.

Conclusion

Each of the three technology platforms described in this article are still in the early stages of their life-cycles. In fact, one could base the next five years of one's career on any of them; each has a key role to play and will continue to gain in importance. To begin, SOA design principles will increasingly serve as the standard by which well-run IT organizations measure themselves. These concepts will impact all areas of the IT life-cycle, from enterprise data modeling through application development and integration. Meanwhile, SaaS-based applications are on a meteoric rise. It's not too hard to envision a time when IT executives will be forced to justify why they are going through the time,

expense, and headache of self-hosting packaged enterprise software. Of course, there will still be a role to play for these comprehensive applications, but increasing numbers of enterprises are looking for the quick ROI sported by SaaS solutions. Finally, composite applications will progressively serve as the user interface-based glue that holds together data from silos both inside and outside the firewall.

References

[REF-1] "SOA: Principles of Service Design", Thomas Erl, Prentice Hall/PearsonPTR, 2007.

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[Home](#) [Past Issues](#) [Contributors](#) [What is SOA?](#) [SOA Books](#) [Legal](#) [RSS](#)

Copyright © 2006-2007 SOA Systems Inc.

All Rights Reserved