On Engineering Web-based Enterprise Applications

Srinivasa Narayanan, Subbu N. Subramanian, Manish Arya, and the Tavant Team

Tavant Technologies
3101 Jay Street, Santa Clara, CA 95054 USA
{srinivas.narayanan, subbu, manish.arya}@tavant.com

Abstract

Enterprises are fast adopting SOA and On-Demand technologies in the context of building web-based applications and web-enabling existing applications. These give rise to a new set of engineering challenges. We discuss these challenges based on our experiences in building enterprise applications for Fortune 100 customers. Integration plays a key role in these applications and presents challenges such as the need to do partial integration, dealing with connectivity problems between systems, and integrating with legacy systems. We have used techniques such as caching meta-data and data about partner systems and adapting semantics based on user expectation to address performance issues in integration. In many cases, the customers require that the solution be available on-demand – i.e. hosted in a multi-tenant mode with the ability to share the entire hardware and software stack across multiple customers, the ability to perform software upgrades without any application down-time, and the ability to offer a range of customization features for its customers. We share our experiences and the challenges we faced in this context from the perspective of system engineers who have built this application

1. Introduction

Over the past few years, the World Wide Web has been radically changing the way enterprises conduct their business. Easy and ready accessibility of information via the ubiquitous browser, its concomitant effect of improved business efficiency, and availability of tools for rapid web application development and deployment has been driving corporations to adopt the Web as their primary medium for delivering applications. Companies have been building new applications specifically targeted for the Web to make use of the easy accessibility of previously difficult to obtain information. Also, they have rapidly enabled legacy applications to be available on the Web.

In recent times, enterprises are increasingly adopting two new technology paradigms – Service Oriented Architecture (SOA) and On-Demand Applications – in the context of web-based solutions. This paper is a result of our experiences in building applications using these technologies. We discuss the motivation behind enterprise adoption of SOA and On-Demand technologies, engineering challenges that arise in building these applications, describe solutions, and highlight opportunities that exist for researchers working in these areas.

2. Experiences: Integration and SOA

The main access point of Web-based applications is the browser. Thus, unlike conventional applications, this allows for the possibility of customers external to an organization to directly access the applications on the Web. Exposing a set of independent but related applications to the customer lends itself naturally to the need to allow for business processes that span across these applications.

To illustrate, consider the example of a bank that enables its customers to perform online banking on the internet (say by web-enabling its internal banking application). Also, let us say the bank enables its credit card division's credit account management system as well on the web. A natural transaction of interest to the bank's customers would be the ability to transfer money from their savings account to their credit card account. This necessitates integrating the disparate banking and credit account management applications.

Besides facilitating instant and easy access of an application for end customers, the web allows for

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14-16, 2006

accessing the application's data and services by other *applications* as well. Traditionally such integration is performed in an ad-hoc manner using custom integration techniques that is unique to each integration touch point.

Technologies such as webservices and recent architectural paradigms such as SOA [1] that stipulate standard integration patterns are becoming a popular alternatives to integrate multiple applications. In this section, we discuss several SOA integration challenges and solutions in the context of enterprise applications.

2.2 Challenges in SOA

SOA provides a loosely-coupled architecture to integrating disparate applications. A typical SOA application is constructed from many smaller applications that provide specific business services. These business services are orchestrated together to implement a higher level business service. The orchestration can be hand written or be specific through languages such as BPEL [2]. The interaction among the applications is usually a combination of asynchronous communication via messaging or through synchronous communication via web services or other technologies such RMI/EJB/CORBA etc.

Building such SOA based applications brings several challenges in design, implementation, testing and monitoring phases. At Tavant, we built a comprehensive system for dealing with all these aspects of the development lifecycle of a SOA application.

The design time tool allows one to declaratively specify the nature of the interactions between the services including characteristics such as the type of interaction (sync or async), the schema of the message exchanged, the required elements in a message for each interaction – some of these validations go beyond what can typically be captured through XML schemas.

We also developed a testing framework that simulates interactions between the loosely coupled systems in a SOA application. This allows for a tester to test one application independent of another. The framework allows some systems to be deployed live during testing and others to be simulated. This allows for the incremental testing of a large application by gradually adding in more live systems as they are ready.

Application-level monitoring of a large distributed SOA system also adds several challenges. A large business process involves interactions between many small independent distributed systems and failures could happen in any of those systems, and detecting failures in such workflows is a non-trivial task. We built an event processing framework that collects events from several distributed systems and allows the user to declaratively specify patterns in such events that constitute success or failure. A workflow is modeled as a "AND-OR" graph where each node represents an event. Certain nodes are declared to be terminating nodes. The model also allows the user to declare service-level agreements (SLAs) for workflows. For example, one can declare that a certain workflow is supposed to complete within 2 minutes. As events happen, the event processing system updates the status of the various running workflow instances and allows for alerts to be generated based on declarative rules (such as if more than 1% of workflows of a certain type are incomplete).

2.3 Partial Integration

Many businesses are web-enabling their business processes in a phased manner by doing them one by one instead of all at once. The business reasons are obvious -- minimize risk, see early return on investment, and synchronize technology changes with user adoption rates. Hence, an integrated application needs to deal with integrating partial business data and processes belonging to the component systems.

Consider a system that acts as an order management hub system whose canonical model consists of concepts such as Orders, Invoices, and Shipping as well as the business processes involving these concepts. In the long run, the hub system maybe required to interact with backend Invoicing Systems and Shipping Systems to fulfill the business processes. In our definition, a "hub" system is one which acts as an orchestrator of various business processes and invokes services from various target systems to fulfill a request. In this context, consider a scenario where a business wants to achieve integration and collaboration involving its invoicing system first and wants to do the same for its shipping system in a later phase. To address this requirement, the hub system needs to represent Invoice data in the canonical model when the corresponding data for Orders and Shipments is not available. This causes interesting technical challenges: (a) Data modeling becomes a more complicated task – e.g., foreign key restrictions in the canonical model need to be relaxed. (b) Business logic needs to be made inherently aware that only parts of the data in the data model graph may be genuinely available. (c) With no clear models for specifying unavailability of data in relational database systems, queries on the data model must be careful not to return incorrect answers because of the partial availability of data.

2.4 Leveraging Integrated Data to Handle Connectivity Challenges

Query processing in a distributed systems environment is a well-studied subject [4]. While many of the ideas are

relevant in the context of a data integration setting as well, there are key differences due to the fact that these two environments differ in some fundamental ways.

While the distributed systems environment is tightly coupled, the data integration environment comprises of loosely coupled, autonomous component systems. In particular, the reliability of the component systems as well as reliable communication among them cannot be guaranteed. Thus in practice, the integration system should be able to cope with unavailable component systems and serve useful purpose even when some of the component systems are unavailable.

A technique for addressing the above problem is to use build a integration hub (like the Order Management hub discussed in the earlier section) and use the integration hub to cache data. This cache can be used to substitute for the component system when the component system is unavailable. This approach is based on the belief that it is better to present a possibly stale result to the user rather than not present any results at all. In these instances, the hub system may want to inform the user about the existence of the cache and warn the user that the data may not be current.

As an example, let us say that an application needs to present pricing data to the user. The product catalog in the integration hub cache can cache the pricing information for the various products. Consider a scenario where while answering a user query regarding product price and availability, the hub fails to obtain the latest pricing information from the master component system (say due to connectivity issues). The integration system may still want to present the user with the pricing information from the cache along with an indication to the user that the information may not be accurate.

It is important to note that the above techniques for dealing with unavailable component systems may not be applicable in the context of all integration scenarios. The integration system should allow for declaratively specifying the scenarios where the cached information from the integration hub can be substituted and/or query relaxation techniques can be applied. The system should make use of these specifications to apply the appropriate technique for a given scenario. Also the integration system should allow for an option where the end user is made aware of the fact that in order to generate the results to her query, one or more of these scenarios have been applied.

2.5 Data Semantics as a Function of User Expectation

This section discusses yet another scenario that leverages the existence of the integration hub cache. It exploits the fact that in real-life, semantics is context sensitive. For example, when a user is browsing the product catalog, it can be acceptable to present possibly stale and hence approximate price information. But when the user is placing an order for the same item, it is important that the user is presented the current accurate price of that item. A smart integration system can take advantage of these semantic distinctions that arise in real-life. This technique has been used in several order processing applications that we built - the catalog prices are cached on the hub on a nightly basis. Thus, prices for items can be more efficiently obtained for the more frequent catalog browsing activity. For the less frequent ordering activity, current prices are obtained from the component system using real-time web service queries.

3. Experiences: On-Demand Web Applications

The traditional model of deploying applications dictates that the necessary hardware and software be procured, installed, and maintained at the customer's location. This model adds huge costs to the customer who has to buy the hardware and also pay for the expertise to install and manage these applications. Besides, this model does not account for fluctuations in demand – investments in hardware and software will need to be typically based on peak demand considerations and optimistic business growth projections.

In recent times, several vendors are offering applications as a hosted service to customers with the promise of reduced total cost for customers by amortizing the expertise and management costs across many customers. These pay-as-you-go services help to speed returns on investment by reducing up-front project and infrastructure costs.

3.1 Multi-tenancy

Architectural issues in enterprise software systems have forced On-demand vendors to follow the ``one instance, one box" model in which the vendor has to create a separate instance of the application on a dedicated set of servers for each customer. With a large number of customers that are needed to reap the benefits of the On-demand model, it creates a complex hardware and software environment with significant administrative burden that cannot be amortized much more than in the case of the traditional model.

Multi-tenancy can be defined as the ability to host applications for multiple customers by sharing the

hardware and/or software resources among these customers to reduce the overall cost of the application for these customers.

For Web-based enterprise applications, typical hardware resources are the servers hosting the application - in most cases, Web Servers, Application (Business Logic) Servers and Database Servers. The software resources are the corresponding software code - the Web Server code, the application server code, the databases containing the persistent application data and the database manager managing these databases.

Using the above definition, Multi-tenancy can take many forms depending on which resources are shared among the customers. The greater the administrative costs involved in managing the resource, the greater the benefits of sharing the resource. However, not all these resources lend themselves to be shared across customers with equal ease. For example, sharing the database instance across customers involves designing the database schema from ground up to support multiple customers. Similarly, sharing the application software instance involves designing the object model, the process flows and other components of the application logic in such a way that it can be leveraged for multiple customers.

Tavant has deployed On-demand applications to its customers that have been designed at all levels to support ``complete" multi-tenancy - where all of the above mentioned resources can be shared across customers. The complete multi-tenancy model helps in amortization of high-cost and medium-cost activities such as database administration for the database instances, management of application servers and web servers including continuous monitoring of server processes, software upgrades etc.

Thus, the complete multi-tenant architecture provides a level of cost reduction far beyond what can be achieved by simply hosting an application. On the flip side, this model causes security concerns as the code and data is shared between customers. Also, providing independent performance guarantees to customers and isolating problems so that errors with one customer do not affect another are difficult to achieve.

On first glance, it may seem unlikely that two different enterprises would want to share the same stack considering the above mentioned security and problem isolation issues. While this would probably be the case for competing or totally different large businesses, multi-tenancy within a enterprise across different business units, across collaborating businesses, or for small and medium enterprises can be quite an attractive proposition due to the significant cost savings that it can provide. In fact, the multi-tenancy architecture has been used in a very successful way among different internal businesses units in some of Tavant's large (Fortune 100) customers.

3.2 Multi-tenancy Facilitates Collaboration

In a multi-tenant system that shares the database instance, the data of different customers is in a common place. This enables the system to provide new business processes to make use of this data and thus create new collaboration opportunities between these customers. Consider the example of an enterprise with different business units that sell related products. Lets say that these business units have started to share a complete multi-tenant system for their order processing application. These business units that have traditionally been relying on completely isolated systems, now have the opportunity of devising new business processes such as product promotions across business units and product searches across business units. If the data had been distributed over multiple systems, such business processes would have been difficult to implement due to the lack of technologies for performing cross-system distributed querying over the Intranet or the Internet.

3.3 Online Software Upgrades

Many enterprises that have Web-enabled their applications have high demands on the uptime of these systems. With the applications being accessed by users all over the world, it is impossible to identify an reasonable time for bringing systems down. In keeping with such demands, the Tavant solutions are designed to be highly available with almost no down time. Many traditional enterprise applications have downtime requirements for activities such as applying patches and release upgrades.

The Tavant solutions have been carefully engineered at all levels to avoid down time during software upgrades (for both bug fixes and release upgrades).

A typical Tavant hosted on-demand solution has a multi-tier architecture with a cluster of application servers that all connect to backend databases. A new software release involves the following steps in that order -- (1) Database Schema Change: upgrade the database schema so that the new code corresponding to the new schema can function properly; (2) Application Code Change: upgrade the application software on the application server cluster one server at a time; and (3) Configuration Data Change: once the software has been upgraded on all the application servers, perform the necessary configuration and migration activities to update the database content and make it ready for the new changes to function correctly. Sometimes, some or all of step (3) is done prior to step (1), but we will ignore this detail for sake of simplicity.

Database schema changes are done before application code changes because, in most cases, the new application code will be unable to function without the new database schema.

Ensuring that the application functions correctly throughout all these steps is a non-trivial task. We give an overview of some of the challenges faced during these steps and how Tavant addresses them.

Consider the first step in the release process - database schema change. The requirement of zero down time during database schema changes is achieved by making the schema changes backward compatible with the previous version of the application code - i.e. the old application code has to function correctly even with the new schema. Let us illustrate this with a concrete example. A change such as dropping a column in the database table is not backward compatible with the previous version of the code because the previous code may be still using this column. We solve this problem by rolling out a drop column change over multiple releases - the first release would essentially make the application stop using the old column and migrate all the data in the old column to the relevant parts of the new schema; a later release can then drop column as it is totally unused by now and hence dropping the column will not affect any part of the application. With a similar line of thought, other kinds of database schema changes can also be engineered to be backward compatible. While defining the exact process by which a database schema change can be made backward compatible is fairly straight-forward, it is interesting to note how, with zero down-time requirements, such simple operations have become more complicated and lengthier to manage.

Let us consider step (2) - Application Code Changes next. In a cluster of application servers, code is updated one machine at a time. Hence, at any given point in time, some servers may be running the previous version of the code and some servers may be running the upgraded code. This implies that these two versions of the code need to run in a compatible manner with each other and each version should update the database content in such a way that the other version is able to function correctly on it.

Performing configuration data changes (Step (3) of the release process) may take a reasonable amount of time and its important that the application functions appropriately even when the configuration steps are in progress.

It is important to note that we do not expect the application to function exactly the same way during the release process as it will at the end of the release process. However, the way the application behaves should be ``appropriate enough'' for it to be used meaningfully by its users at all times during the release process. This notion of ``appropriate enough'' is usually dependent on the specific contexts within the application.

4. Conclusions

We discussed several challenges that arise in the context of building Web-based enterprise applications that are built using the SOA and On-Demand technologies. As this breed of applications continues to grow more popular, we expect the two key issues mentioned earlier – integration and multi-tenancy -- to become even more important. We highlighted some of the challenges in each of these areas and described Tavant's solutions to these challenges. We expect the multi-tenancy deployment architecture will become an increasingly important deployment model for enterprise applications. The hosting and multi-tenant deployment model also creates a need to define new software release processes to adapt to this new model since it has an impact on traditional processes such as software upgrades, code releases, quality assurance and testing, and release management.

Acknowledgments: We would like to thank Laks V.S. Lakshmanan for a detailed review of an earlier version of the paper.

5. References

[1] Erl, Thomas. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. *Prentice Hall Service-Oriented Computing Serie,* 2006

[2] Matjaz B. Juric. Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition. 2006

[3] Barga, R. and Weikum, G. Recovery guarantees for multi-tier applications. In *Proceedings of ICDE Conference, San Jose, CA (March 2002)*, pages 543--554, 2002.

[4] Ozsu, M.T. and Valduriez, P. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

[5] B.Shegalov, G., Weikum, G. and Lomet, D. Eos: Exactly-once e-service middleware. In *VLDB 2002*, *Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China*, pages 1043--1046, 2002.