Abstract

There have been a lot of new tools being released lately that provide the foundation for the creation of Web Service based applications. Now we need to figure out how to use this power. How, for instance, could a packaged e-Commerce application, such as WCS, be utilized to exploit the vision of Web Services.

While enabling a packaged e-Commerce’s functions to be available as Web Services is a relatively straightforward process, there are a number of issues related to how that packaged application could interact with other Web Services inside, and outside, a customer’s environment.

Many of the key issues are centered on data. E-Commerce services take in data as input, process it, store it, interpret it, and typically, return it as output. In the traditional, tightly coupled view of applications it is acceptable to pass codes and data identifiers around as input to a process. The assumption here is that the packaged application either knows how to get details about the data in question or it can store that identifier in the database and services can understand what that identifier means via DB joins. Related to this is the fact that in this environment the database can control the integrity of the data.

However, when attempting to de-couple the key business services that make up an e-Commerce package and make those services available to others, there are a number of issues that need to be addressed. For example, how much data needs to be passed into a business service in order for it to complete the processing that is in the domain of that business component.

This paper will 1) Present these and other key application design issues, 2) Present examples to illustrate the problem, 3) Discuss the motivation behind the need for a solution, and 4) Present various options available to addressing these issues. In addition, special consideration will be given to issues such as auditing, reporting, and system management.
Table of Contents:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>The Vision</td>
<td>3</td>
</tr>
<tr>
<td>Some of the key goals include</td>
<td>4</td>
</tr>
<tr>
<td>Foundation Premises</td>
<td>4</td>
</tr>
<tr>
<td>Issues</td>
<td>8</td>
</tr>
<tr>
<td>Strategies: So how can applications be designed to address these issues</td>
<td>14</td>
</tr>
<tr>
<td>Utilize standard based tools for service lookup</td>
<td>14</td>
</tr>
<tr>
<td>Design systems that are layered</td>
<td>14</td>
</tr>
<tr>
<td>Design for “Loosely Coupled” Components</td>
<td>15</td>
</tr>
<tr>
<td>Define clear contracts between layers and services, but utilize the “Parameter List” paradigm where possible.</td>
<td>15</td>
</tr>
<tr>
<td>Business will have to reach agreement before systems can interact</td>
<td>17</td>
</tr>
<tr>
<td>The previous item points to some of the challenges associated with designing for “dynamic” e-business. The real question is how many transactions or interactions will really be dynamic. Just how often will application designers be required to lookup business services and execute them dynamically.</td>
<td>17</td>
</tr>
<tr>
<td>If we could assume that business will typically reach agreement as to what to call and what to pass before application development begins, then the complexity of designing business systems will be greatly reduced.</td>
<td>17</td>
</tr>
<tr>
<td>Utilize standard XML structures to pass data</td>
<td>17</td>
</tr>
<tr>
<td>Components/Services should be design to limit dependency on other components in the same layer</td>
<td>19</td>
</tr>
<tr>
<td>Implement all cross “domain” business processes in a “control” or “workflow” layer</td>
<td>20</td>
</tr>
<tr>
<td>Components should be designed to store data and Meta data and make it available</td>
<td>21</td>
</tr>
<tr>
<td>Business components should be responsible for managing data that is associated to it’s domain.</td>
<td>22</td>
</tr>
<tr>
<td>Web Services should be designed to allow for multiple versions to be in production at any one time</td>
<td>22</td>
</tr>
<tr>
<td>Web Services should be designed to leverage a Security or Policy Control Service that enables controlled access to process and data</td>
<td>23</td>
</tr>
<tr>
<td>Web Services should be designed to include application management interfaces from day one.</td>
<td>23</td>
</tr>
<tr>
<td>In order to manageable, components need to be designed from the ground up to enable effective manageability, trace-ability, reliability, and monitor-ability.</td>
<td>23</td>
</tr>
<tr>
<td>Implement or utilize a standard business code lookup service</td>
<td>23</td>
</tr>
<tr>
<td>System Level Services offer a good place to start for an organization to get into Web Services</td>
<td>24</td>
</tr>
<tr>
<td>Integrating with packaged applications using Web Services</td>
<td>25</td>
</tr>
<tr>
<td>Conclusions</td>
<td>25</td>
</tr>
<tr>
<td>Resources</td>
<td>26</td>
</tr>
</tbody>
</table>
**Introduction**

One of the hottest developments recently has been the standardization of Web Services. This technology is yet another piece of puzzle to web enable business systems and to allow asynchronous or synchronous (real time) access to systems where ever they are. To date there has been a lot of focus paid to the plumbing layer; how do you make a business process available as a Web Service and how do you find one. This includes discussions on what and how to use UDDI, SOAP, WSDL, etc… However, there are still a number of questions as to how you design, or redesign, an application to be made available as a Web Service. This white paper will focus in on some of the issues and things to consider at the Application Architecture level.

![Diagram of Application Architecture]

Figure 1.

Using an eCommerce business model, this paper will review what it means to build an applications with Web Services, some fundamental premises, the key issues typically encountered, and selected strategies to address these issues.

**The Vision**

The vision behind Web Services, and this paper, is that applications can be built/assembled by leveraging key business and system functionality in applications no matter where they are deployed, who built them, or the language they are developed in.

In addition, Web Services technology makes it possible to make business services located within an organization in systems available to other parties, both inside and outside and organization. The later opens up new potential revenue channels and new ways to service customers.
Some of the key goals include
1) reusing services across applications without having to build or deploy the same code over and over again within an enterprise
2) making key business processes available to external parties to better facilitate business process integration

You may notice that this sounds very much like the concept behind EAI, or Enterprise Application Integration. Web Services can, in fact, be used to address both synchronous or asynchronous integration of business systems. The issues and approaches differ somewhat depending on whether one is trying to implement real time verses asynchronous communication. This paper will primarily focus on using Web Services to implement real time communication between organizations and system, though issues encountered in asynchronous implementations will be discussed.

The Web Services approach and standards provides the core tool set for implementing applications that can dynamically utilize applications components from many sources. However, the real challenge lies in HOW we design applications to fully leverage the power provided in Web Services.

**Foundation Premises**

**Components form the Architectural Basis for Web Services:**
The vision of creating a web of services to implement an organizations business is built on the same premise that is used in creating an application out of loosely coupled business components. In essence, Web Services are nothing more than the public interfaces to business components. Therefore, before preceding it is important to define an architectural definition of what it means to create a component based application out of loosely coupled components.

Each of these will be explained further in the strategie’s section.

- Large Grained: A component, in this context, represents a set of business processes and data that make up a business domain.
- Loosely Coupled: A component is designed to be as independent as possible of other business components.
- Components are responsible for implementing business processes that are internal to that business domain, business processes that are part of cross component business processes, and functionality to create, retrieve, update, and delete business data related to a component.
- Encapsulation: The user of a component should not care how the component is implemented.
- Clearly defined and published interfaces: Interacting with a component should be via a contract approach.
• Layered Approach: The view of an IT Architecture as being made up of layers is critical to the implementation of a component, or Web Services, based system. There are 3 core classes of components: System Services, Business Services, and Control.
• Components can be deployed and re-deployed without affecting the overall application.
• Complex Business Process Management: Business services that are provided in a component can be combined with those of other components to implement complex business processes.
• Components may store data in separate databases or data sources that are not linked or viewed as a single entity.
• Well defined application management facilities.

While this application architectural definition has been laid out with little mention of Web Services, all of the architectural concepts defined above also apply to building a system out of Web Services.

Looking at the problem space from different perspectives

Business Perspective (Non-IT view)
In order to truly appreciate the challenges to designing an application such as this, I find it often valuable to look at the problem from a non-IT perspective. Imagine, if you will, a business is looking to implement an order capture process, without using computers. They also have outsourced their customer/account management functions. The following diagram illustrates this model.
Using this diagram we can walk through several business processes and identify a number of issues. You may notice that some of the specific details on the processes listed below are not typical of a paper-based process. These are highly simplified, but they do help to illustrate some of the issues.

1) Take an order #1: Let’s say that the customers’ order form requires them to enter in what they want to order, their customer ID number, and their customer type code.
   - How does the person processing the order know who they are based on that ID? They would have to ask customer management for details on the customer? How would customer management look this data up? They would have to use the customer ID number and an identifier relating to the company asking for the data.
   - Can they get the information fast enough?

2) Request a Catalog:
   - Again, the Customer Management “service” would have to be accessed. But who maintains the association of catalogs to customers? Who has the appropriate information?
3) Check Product Inventory:

- In this scenario, there are two suppliers of product. How does the inventory manager know which one to ask? Is there a mapping between product and supplier that has to be maintained?

The bottom line: In order to make this work people need to know core data, such as customer ID, but they also need to know data about the data, or Meta Data. Stated another way, if company X is offering a “Customer Management Service” as an offering that can be purchased by other companies (e.g. IBM, Siemens, Amex), then company X needs to not only keep track of the end customer (e.g. John Doe), they also need to keep track of who John Doe is a customer of (e.g. IBM).

**IT Perspective**

Now let’s approach the problem from an IT Architecture perspective. Instead of people we have application components that execute the business processes or business process steps.
These components provide business and or system services to the application, and any of these interfaces into the components could be implemented as Web Services in an intranet or extranet environment. One important item to note is that this view only looks at it from a single application view. There are number issues that result when one attempts to assemble an application from multiple “providers”. This will be reviewed below

**Issues**

The previous section has provide a high level vision of an application Architecture structured to use loosely coupled components and their interfaces, the Web Services.

As the saying goes the “devil is in the details”. While there have been a number of papers published on the tools and processes to locate and connect Web Services, there are still numerous issues related to designing applications to work with components, deploying the components and application using them, and operating these systems. This section will review some of these issues, all of which are based on the architectural premises described above.

The architectural issues can be categorized under a number of categories, among these:

- Object Identification & Understanding: In other words, how do we uniquely identify business entities between Web Services and how do we understand the meaning of the data?
- Referential Integrity: How to we keep the data, maintained by Web Services in sync and accurate?
- Dependency Management: How do we manage the dependencies of Web Services such that we can effectively deploy and manage an application using Web Services without breaking it or including unwanted services?
- Process Control: How do we chain business process steps together to create complex business processes using Web Services?
- Finding the right Web Services: e.g. How do we find the proper order management service for office supplies when there are 3 vendors?
- Reporting across components: If data is spread across multiple services, how do we get a consolidated view?
- Application Management: How do we manage applications that potentially use services our of our control?

Each of these will be explored in detail below. One point to note is that in the following discussion Component and Web Service are used interchangeably, since it is assumed that Web Services are implemented by large grained components.

1. How do you know what data a component needs to perform it’s business process?

2. If a business process requires data that is maintained outside it’s domain, can you simply pass the process an identifier to this data or do you need to pass in a representation of the entity (i.e.
the whole object).? From a strict “loose coupling” standpoint the component should not have logic as to how to get such an object.

3. If a business component is responsible for maintaining data that has a relationship to data outside it’s domain, how should it store this relationship? For example. If an order management component needs to store customer information related to an order how does it do this? Is storing the identifier enough? Does it also need to store meta data about the source?

4. If a component is designed to receive data from another component or “domain” how should it represent that data within it’s domain? Does it need to define/create it’s own object structure to hold this data? If it attempts to create an object to hold the data, that was defined in the other component, then there is coupling on 2 fronts. If it doesn’t put the data retrieved into some sort of container that gives meaning to the data how does in know which attribute is which?

5. If a business component stores identifiers or data that is owned by another domain (see above) and that data is changed or removed by the owning component, how does the consuming
component know about the change? How is referential integrity handled? For example: if an order maintains an id for the customer that ordered that and the customer is deleted, how does the order know about that? Does it need to care? What if the order holds a copy of the customers address and that changes? Does it care? Refer to Figure 4.

![Figure 4.](image)

6. If web service /component is dependent on implementing a business process that requires others, how do we design this to minimize coupling? We don’t want to accidentally “drag along” a whole set of relate services that we may or may not want. Where are cross component processes implemented? See Figure 6. [1]
7. If a component adheres to the loose coupling principle, then how do you represent complex object associations that cross Business Component Domains? For example, potential issues could result in the following areas; Customers to addresses, Orders to Customers.

8. Much of the business functionality required for most systems involves simply retrieving or searching for data. How should this be done if the data that needs to be presented crosses multiple components? For example: Orders per customer, Orders for a specific product, etc…

9. If a component/service returns complex data to the caller how should it do this to minimize coupling?

10. If you use an XML document to break the object dependency between component, how will the consuming component know how to interpret the XML tags to get the appropriate data? For example: If you choose to return an XML document to describe a customer, instead of a serialized object with related assessor methods, the process that is “catching” the customer.xml document must be able to understand the name of the elements to access for specific customer information.
11. If a component has a business rule that is dependent on a business code from another domain, how will the logic handle potential differences in the value of the codes? For example: Customer Type Code or Shipping type code? How will it handle different business rules related to different customers of the service?

12. Assuming that we have addressed all the issues related to executing processes and CRUD operations, how can we generate reports of the business data that is maintained across business components/services? Remember that we may or may not have a central database to pull from. Also note that the meaning of the data may be different.

13. How can components (Services) be deployed and upgraded in an orderly manner while maintaining a production environment?

14. Because Web Service/Component based applications are not compiled, all knowledge of what to call, and the interface details are not validated for syntax and type before the actual call is made. How can applications avoid problems with connections breaking?

15. If a business process is implemented in multiple Web Services, say submitting orders to multiple suppliers, how will the system determine which Web Service to call? See figure 7.
16. If a business application utilizes multiple Web Services, hosted by different organizations, how can these services be monitored, managed, and measured in a reliable manner?

17. To what degree can dynamic business communications take place? For example should it be assumed that an agreement has been made ahead of time and the appropriate data has been setup?

18. Where should transactions be controlled from and where is state managed from?

19. If an application using 2 Web Services, and each logs activity and/or problems to different log files in different locations, how do we track down problems?

This is by no means a complete list of the issues an application architect will encounter in building an application architecture that leverages Web Services. It does represent some of the key problems in this area.
Building Business Applications using Web Services

**Strategies: So how can applications be designed to address these issues**

Now that we have walked through some of the issues, what could as application architects do to address some of the issues. This section will review some strategies and approaches that can be applied to the problem. You will notice that many of these are not new to Web Services. As was mentioned earlier, we need to address many of the same problems encountered in building loosely coupled component applications.

Utilize standard based tools for service lookup

This one may seem like a no brainer, since this is essentially the mission of the UDDI services. One of the real questions related to this will be, just what do we setup with the registration? That will be discussed below.

Two important items to note: First, while UDDI promises to provide the capability for Dynamic Lookup of services to call (i.e. give me a Tax Calculation Service), the reality may be closer to more of a dynamic binding capability to a known service.

Second: Even while UDDI defines an approach for registering Web Servics, there is potential need for a facility to provide a “Web Service” Cross reference. Has noted in issue #16 in the previous section, having the ability to call a business process at an external vendor, say a supplier, is one thing. However, if you have multiple suppliers there needs to be a System Level component/service that provides the application with this ability to map business data (e.g. supplier or customer) to an identifier of a business service to call. This Web Service mapping capability would need to be pre configured and then accessed during the course of the business process flow to determine which Web Service to call. [7]

Design systems that are layered

By clearly delineating the layers of services within an application the architecture not only becomes easier to understand, it also aids in the definition of dependencies, of interface “contracts”, and provides structure as to how the application or services within the application could be packaged up and deployed.
Design for “Loosely Coupled” Components

A Web Service by definition is an interface into functionality. This functionality lives inside of application level or system level components. Therefore, it is very important to be cognizant of the impact of integrating loosely coupled components.

Define clear contracts between layers and services, but utilize the “Parameter List” paradigm where possible.

It is very important that the interfaces between components (the services) are clearly defined and understood. However, it is also important to recognize that if you define an interface that specifically delineates the input and output parameters on the Web Services definition, any changes made to the interface, such as adding a parameter, will impact the calling sequence in all applications that are using the Web Services. Alternatively, if the interface to a Web Service utilizes the concept of a “parameter list” as the input and output vehicle, the contents of what is being passed in and out can change without having to
republish the interface. The service, in this case has the responsibility of checking and validating
the parameters and data that are passed in to ensure that it can perform it’s task.

The down side to this approach is that it limit’s UDDI’s capability to effectively detail out all of the
attributes and type information. Thus, a caller needs to have a better understanding of what to pass
in and out before coding and the receiver will need logic within the Web Service to validate the
input.

![Diagram of interface approaches]

Figure 9.
Business will have to reach agreement before systems can interact

The previous item points to some of the challenges associated with designing for “dynamic” e-business. The real question is how many transactions or interactions will really be dynamic. Just how often will application designers be required to lookup business services and execute them dynamically.

If we could assume that business will typically reach agreement as to what to call and what to pass before application development begins, then the complexity of designing business systems will be greatly reduced.

It should be noted that this is a key assumption in many web enabled e-business efforts today, for example: The effort to define XML based standard business messages and interactions that is being addressed as part of ebXML (www.ebXML.org). In fact, much of the standards work in that area revolves around the work to define the negotiation process for doing business on the Web. This includes standard data and process definitions.

Utilize standard XML structures to pass data

As discussed in the issues above, one of the biggest challenges to effectively using Web Services is the need to not only pass data, but also the meaning of data between services. This challenge will only increase as applications move away from using simple “calculation” or “information services” such as tax calculators or stock quotes, to more advanced business processes/services such as “Process Order”.

This problem will need to be address in much the same way that society addressed the problem of regional and global commerce generations ago: with a standardized “currency” and conversion processes.

The standard currency for Web Services in this case is an XML document. Of course, simply using XML is not enough, in order for a Web Service to understand the elements (i.e. metadata) of a document the structure and meaning must be standardized. There are a number of organizations and efforts that are or have been put in place to define this standardized set of XML document definitions, among these are Rosetta Net and ebXML. Note that the ebXML initiative has two focus areas. One is creating common processes and data definitions to enable business of all sizes to participate. The other is to define the Technical Architecture to support this effort. For the purposes of this paper I am primarily recommending consideration of the former, since work is being done to merge WSDL/SOAP/UDDI with ebXML.

Using a standardized XML definition, for say a Purchase Order enables a Web Service to be able to know how to interpret and validate the data coming in without needing to explicitly define all named attributes on it’s interface. (See Above).

---

1 EbXML is an international effort jointly sponsored by the UN/CEFACT (United Nations Center for Trade Facilitation and Electronic Business) and OASIS. The mission: to define a global eBusiness standard for data and processes. [4]
By utilizing standardized XML structures as data transport vehicles, we also enable an application architecture to utilize standardized XML transformations, the “currency conversion” so to speak, between caller and sender.

Alternatively, if utilizing standard XML documents is not an option, then the specification of a clear set of either name/value pairs on the request would be an option. Note the UDDI “discoveryURL” would have to be set in order to communicate this information to the user of the service.

Of course there is a downside to using XML documents, or even large sets of Name/Value pairs: that is performance. There are potential performance issues with sending potentially large XML document between Web Services in an intranet or extranet environment. However, the power of the Metadata that is contained with in the XML structures and the need for such information must be weighed against the amount of data being sent on the line. It is important to note that a) there are also performance hits encountered when serializing objects and b) there is no rule that says you have to send ALL the data related to a standard XML document. You only need to send what is
Building Business Applications using Web Services

required. The perceived impact of this performance hit is lessened to the end user if you are capable of making the cross service calls asynchronously using an EAI solution.

Components/Services should be design to limit dependency on other components in the same layer

Managing dependencies is one of the key challenges in utilizing Web Services in an intranet or extranet scenario. Common dependencies that occur in an application design are:

- Call Flow Dependency: Business processes implemented by systems are not typically within the domain of one business component. For example when processing an order it is often necessary to validate the customer, check credit, and check inventory when processing an order. Please refer to the next section for details on this.

- Object Association Dependency: Using OO techniques it is easy to model a business problem by associating objects together, such as Customer to Order. However, from an implementation perspective, doing so increases the linkage from one component (e.g. Order Mgmt) to another (e.g. Customer Mgmt). The use of interfaces helps somewhat, but it does mean that both Components need to be built using the same set of Interfaces and/or technology. (see Figure 11.) [6]
Implement all cross “domain” business processes in a “control” or “workflow” layer

Building upon the concepts described in the previous item, the flexibility of an application is increased if all business processes that cross multiple business domains are implemented in a workflow layer. In doing so the application architecture has more flexibility in what is called, when it is called, managing the call (i.e. exception handling) and performing any translation on the data that is passed in or out.

The impact of this is that many of the core value business processes are pulled from the typical business domains (e.g. Order Mgmt) into a higher level. However, the Business Processes that are defined at this layer can also be thought of as Web Services. Of course following many of the same strategies as discussed above. See Figure 12. [2]
Components should be designed to store data and meta data and make it available.

As was the case with early commerce systems, understanding the meaning and value of the information passed was critical to implementing distributed commerce systems. In e-commerce systems this challenge holds true too. Typical applications developed today are structured based on a common data repository. In this approach, knowledge about the data is in the data model. When moving to a loosely coupled component, or Web Services approach, many of the assumptions related to data modeling are not valid. In essence, data models also need to be loosely coupled. This will more than likely mean that components/services will need to store data and meta data.

Since components commonly implement business processes, they also manage the data related to the business process, consideration will need to be given to providing the basic services to store, retrieve/search, update and delete that data.
Business components should be responsible for managing data that is associated to it’s domain.

The management of data is a critical function of business applications. As noted above, exposing a business process as a Web Service that is available to users within and outside an organization raises a number of issues related to the data and the database.

If a Business Process is designed to use a relational database structure that leverages the natural referential integrity management capabilities of the RDBMS, there will be issues if a process is executed expecting data or identifiers to data that are not in the database. For example; What if an Order Capture data structure has a referential integrity check for a registered customer id? Where would it look?

Building an application to be used/exposed as a Web Service will more than like result in the following:

1) Referential integrity checks will need to be relaxed, at least at the DB level. This will need to be moved to the application level.
2) There may be duplication of data since components will need to manage the data that they need to execute their business process or they will need to record the location of there source of the data. The later would involve obtaining and storing the information on the Web Service that contains the data. Using this information, a Business Service could, theoretically, use a factory concept to look up the details related to a business entity instead of storing a duplicate copy of the data.

Once you have started down this path, also plan on the need for a “data consolidation” layer to provide for the ability to pull information back together again OR to populate a Data Warehouse full of information since the need to report on data does not go away.

Web Services should be designed to allow for multiple versions to be in production at any one time

As we know business applications are never static. There is always change in either the inputs, outputs or the steps that are taken. Because of this the implementation of Web Services will also be changing. This means that consideration must be given to how we package, manage version, and deploy versions of Web Services within an architecture.

Because Web Services can be made available to applications that are outside the control of the above it is also very important to publicize version information in the UDDI definition. This could be in the DiscoveryURL, the name, or the description. Either way, it will be important for suppliers of Web Services to roll out multiple versions, for example to test new code, and users to have the flexibility to use what works without being forced to cut over to it.
In addition, consider managing the UDDI data load as you would any other code asset. This means storing it in source control in a versioned manner and including it part of the overall build packaging process and installation process.

Web Services should be designed to leverage a Security or Policy Control Service that enables controlled access to process and data

Access control really needs to occur at two levels; first there should be a “service” in place to perform access control checks to ensure that a user/customer/application has access permissions to a Web Service.

If this is required, you could design build a system service to manage this yourself, but you should strongly consider utilizing a product such as Tivoli Policy Manager which is design to provide fine grained access control to processes and data. Of course this does not come without a price in complexity and cost. It will also require that your User Management component interact with it or a common naming service to ensure that you have a common definition of the person or system requesting access to a “resource” (i.e. process or data) [3]

Web Services should be designed to include application management interfaces from day one.

In order to manageable, components need to be designed from the ground up to enable effective manageability, trace-ability, reliability, and monitor-ability. This will include thoughtful consideration in:

• How exceptions are raised to the caller. Since the caller may not have the same class definitions for exceptions that you have, exceptions may have to be “raised” using XML structures.
• How exceptions, and problems are logged. A common logging service should be used to help identification of problems. Granted, this is easier said than in an extranet scenario. One option is to request the caller provide the service information for where to log information. Of course, this is simpler when “contracts” are predefined.
• Raising problem notification to an application management system. Tools such as the Tivoli Event Console could be used as a central collection point. This is another need to have a mapping of Web Services tied to an organization.
• Monitoring if a service is running. Consider implementing a ‘Ping’ service, as one interface to a component enables the ability to create a “healthcheck” process that periodically pings all known services to ensure that they are indeed running.

Implement or utilize a standard business code lookup service
If the adage that a business runs on data is true, the “fuel” in many cases is the codification of that data. It seems that every organization has a set up business codes that are used to run it’s business. Examples of these codes include: Address Type Code, Customer Code, Order Type Code, etc…

The challenges include:
1. The size and format of codes. These could differ based on the caller and will affect the data model.
2. Referential Integrity; if codes are different then model will need to include information on code and source.
3. Application of Business logic based on the codes.

Alternative approaches include:
1. Use industry standards for codes
2. Implement a code mapping system level service utility to translate codes from internal values to external values.
3. Utilize predefined values using UDDI tModels.
4. Design data models that have flexibility in the storage of codes

System Level Services offer a good place to start for an organization to get into Web Services

It is probably apparent by now that there are a number of issues related to deploying Business Processes as Web Services. However, there are fewer issues related to implementing system level services as Web Services.

These services which include, but are not limited to:
1) Security / Access Control
2) Concurrency Mgmt
3) Session Management
4) MetaData Management
5) Print Management
6) Search Engine Support
7) Logging and Audit Trail Support
8) Exception Management, tracking and application monitoring and problem management

The key question to ask at this level is, “Is there value added in using a Web Service for these services or would it be better to simply use a common support services framework for all business components.” As noted earlier, there are performance implications of using Web Services and overdoing their usage, just for the sake of using them is not recommended.
**Integrating with packaged applications using Web Services**

Up to this point we have primarily focused on the design considerations of designing and building component based applications to enable them for Web services.

This section will highlight some of the key issues related to integrating functionality from existing applications, such as Webshere Commerce, into a Web Services based architecture.

As noted above there are a number of issues related to using Web Services and strategies for designing applications to utilize Web Services. Typically packaged e-commerce “engines”, such as Websphere Commerce Suite, have the following characteristics:

1) There are designed as packages and this the object model and data model are not easily broken apart. This is especially true in the context of the data model since the use of Foreign keys and Referential Integrity checks are prevalent. Granted, this is not necessarily a bad thing, it just affects their reusability in a bigger application of Web Services.

2) They are not designed to either utilize business or system services from other applications, nor are they designed to enable other applications to use the services within the eCommerce Application.

3) They do provide integration points that enable the transmission of data, either in batch or real time, to and from other systems (e.g. Order Fulfillment systems).

4) The interface to business processes is not based on the Request/Response Model. Instead it is based on passing in Name/Value pairs. The contents of the Value are often data identifiers passed from the JSP or Net.Data code. This is directly related to #1.

5) They rely on a JSP’s and JavaBeans or JDBC to access the database for typical data retrieval processes. Command or Process interfaces exist to execute business processes and update the database, often one in the same, not retrieve data from the databases. This results in a two step process to execute a many business processes, step #1: Send data in (i.e. Shopping cart) to be processed and step #2: query the system for the results. If implemented using Web Services this would result in two Web Service lookups and executions for many business processes.

This is not to say that there are not options for how to create a wrapper around this type of application. You could, for example using WCS 5.1, create a Java wrapper that acts as the Web Service and controls the execution of multiple WCS commands either using the standard WCS call sequence, or via the MQ “back door” interface.

**Conclusions**

The use of Web Services does offer the application architect many new opportunities for designing not only e-commerce but all e-business applications. While Web Services does opened up the potential of creating an application that can execute business processes, in real time, no matter where, there are many issues that need to be thought through. Most of these issues are not new to
Web Services. They have their roots in the struggle to build loosely coupled, component based applications, and in building enterprise data models.

**Resources**

7. UDDI Data References Structures, (See [http://www.uddi.org](http://www.uddi.org))