The Adapter Architecture for Distributed Web Services Using XML Components

Why use a component adapter approach?

It is now widely recognized that business applications should be separated into multiple "tiers", such as backend services, business services and user interface. This architecture promotes reuse, simplifies maintenance and makes they information system more resilient to both technical and business change.

The tiers in such a system are generally separated by means of some "middleware", such as XML over HTTP, Soap, EJB, MQ-Series or Corba. Services in each tier are accessible over one or more of these distributed protocols and provides for this more robust enterprise architecture.

Given the desire to embrace distributed web services, developers frequently select one of these middleware solutions and start building technology and business services. For example, they may build a "schedule service" using EJB and one of the IDE toolkits that support EJB. These toolkits provide wizards and other aids to produce the service.

It is our contention that while the desire for a multi-tier architecture is correct, building "business logic" directly into such a distributed protocol is the wrong approach for a number of reasons;

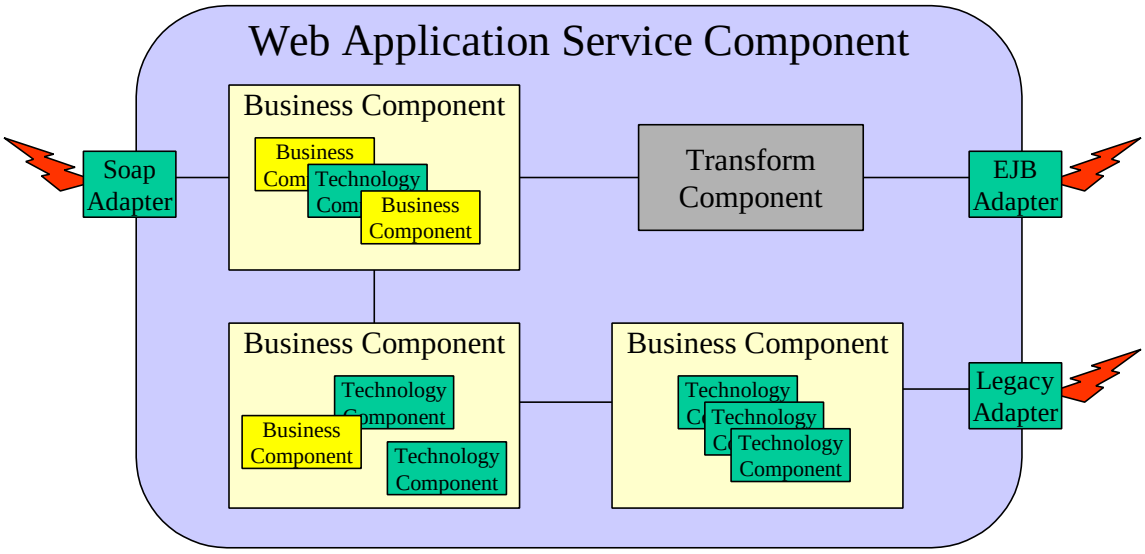
- Technologies change – inevitably the middleware of today is the legacy of tomorrow.
- Multiple middleware – it is a rare enterprise that works with just one middleware solution. There is a frequent need to support multiple protocols, sometimes for the same service. Integration is required across multiple protocols.
- The implementation inevitably ties into the business logic and technology solution. Distribution concerns and business logic are not the same and frequently need to be managed and changed separately. Distribution and business logic often involve very different kinds of expertise.
- Not everything need be distributed – While the capability to distribute is wonderful, the overhead of these protocols makes it impractical to synthesize new services from others and combine them within a single application. Services should be able to be local or remote within the same architecture.
- It's still just code. The approach for building distributed services directly on these platform frequently requires specialized coding skills, it is difficult and expensive. The "wizards" that assist the programmer produce one-shot results that are difficult to trace and reproduce.
- Separation of concerns – Good design dictates that separable concerns should be separate. Business logic and distribution are clearly separate concerns.

A component approach to distributed web services may provide a way to realize the advantages of these open middleware protocols without the drawbacks. The component approach is applied both to the construction of the business logic components and the adapter components that expose the business logic components as distributed web services. The adapter pattern is well established, it provides for two distinct protocols that are joined with an "adapter" in the middle. In the optimal case these adapters are generic and can support any interface that can flow between the distinct protocols. For those familiar with UML terminology – adapters are boundary objects.

The somewhat non-obvious conclusion we came to is that an important part of building components for distributed services is to build *non distributed components first*. This may seem counter intuitive since our experience tells us that "local" things, like Java classes don't distribute well unless they are designed for it. This is also true. The approach we settled on is to build non-distributed components that are designed for distribution. That is, the components interact using the kind of "large grain" interaction style that works across the internet – but the base implementation of these components is within a single server (E.G. Java VM).

Copyright © 2001, Data Access Technologies, Inc.

The adapter pattern is then used to "wire" business object components to adapters that provide for the connection to one or more standard distributed protocols. (E.G. Soap or ebXML). This allows the same business logic component to be wired to different adapters in the same or different application server. It also allows transformations, computations, lookups and other integration logic to be put between components – providing for the natural differences in business or technical interfaces. Since the components can be wired within an application server, this adaptation of both simple and efficient – eliminating unnecessary distribution overhead while maintaining a separation between "tiers".



Adapters turn business components into web services

The above diagram shown how a set of enterprise business components can be "wired" together and also wired to adapters to support distributed web protocols such as Soap, EJB and legacy systems. Business components may contain other business components as well as the technology components and other business components needed to make them work. It also shown how transform components can be used to adapt business process or technologies between independently developed services or components. Web application service components will typically be deployed under an application server's "container".

Adapters do what ever they need to connect to the distribution technology. In some cases they may need to generate or read-in "header files" or other configuration or definition artifacts. In other cases they may need to invoke services or provide components that live in an application server "container" – whatever the technology requires, it is up to the adapter to make the connection (including specialized implementation which may still require using technology specific IDE tools). Note that this pattern works just as well for limited, legacy or proprietary protocols as it does for open and standard protocols.

To maintain an open environment, we have chosen XML to be the basis for component interaction. Each component produces and/or consumes XML through well defined interface "ports". An interactive environment then allows components to be assembled, configured and wired together. Since our base implementation is in Java, Java events carry XML from one component to another. Adapter components turn these "local" XML interactions into distributed web services. Adapters are just components that "speak" Java-XML on one side and a distributed protocol on another. The other side of the web service may be additional components or may be a web service implemented in another way – providing a very open environment. Since XML is the basis for component to component interaction, supporting XML based web protocols (such as Soap, ebXML…) is quite simple.

Whenever a distribution layer is required, adapter components may be inserted between any two components and they can then be deployed on separate servers – thus providing for load balancing and scalability.

Copyright © 2001, Data Access Technologies, Inc.

Additional components are provided to do transformations, DBMS interactions, control flow and other integration or application development needs. This allows much of the web service to be built in an interactive component assembly environment and allows for components that take care of the inevitable "impedance mismatch" between independently designed, business units services or components.

There are times when a tight interaction is required between components – one that is not practical to distribute. While the base architecture encourages the production of distributable components, the component builder may elect to build and use components within the same application server . These co-located components may have a higher "bandwidth" of interaction and may share data. Coupling components is a design decision that turns out to be relatively easy to make once the distinction is clear.

Since components may also be implemented in Java, Legacy integration can be achieved in two ways – by directly integrating with an applications distributed protocol (if it has one), or by using Java to call into the legacy application, and in doing so making the legacy application into an XML component. Legacy applications as well as system services may all appear as XML components.

In building Component-X ™, we have found this local but distributable component architecture to be highly effective and very efficient, while maintaining a scalable enterprise solution.

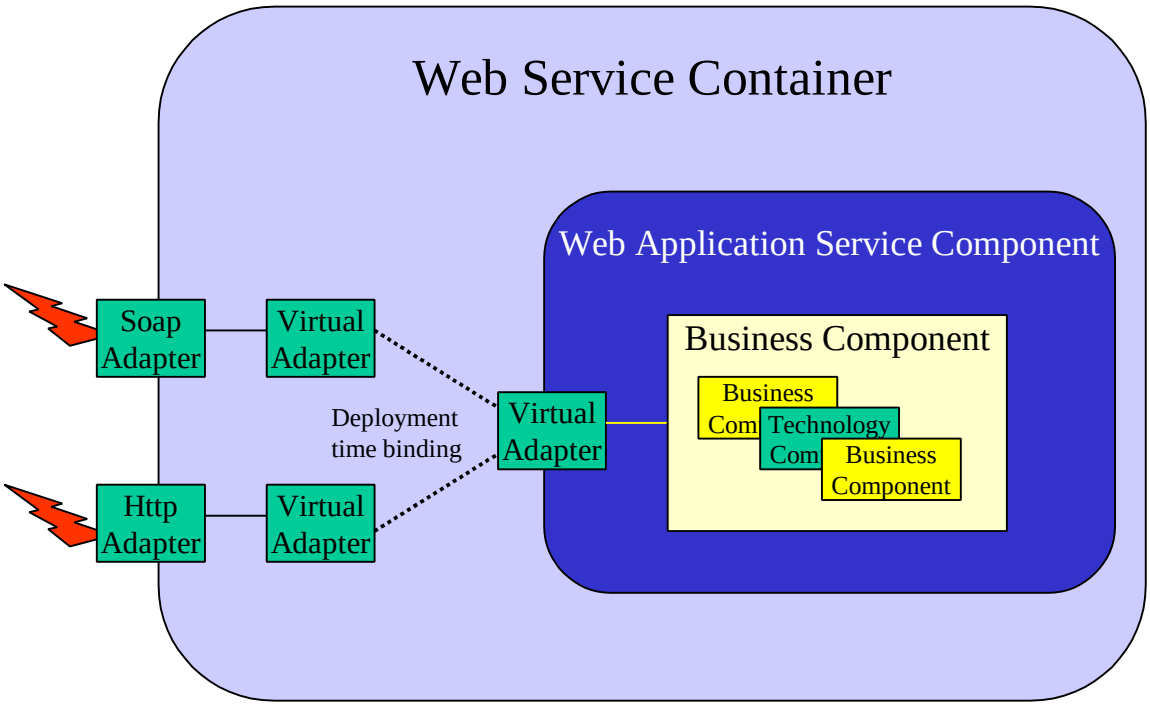
Component-X Adapters

The Component-X environment already provides for connecting components to components and building components from other components. Component-X adapters are built on this base, an adapter is just another component that can be connected to other components in the usual way. Component-X studio is used to connect adapter components to business logic components to create the application service components. Adapters are usually "primitive" components (they are implemented in Java). Due to the basic capability of Component-X to integrate any primitive component, the library of adapters may be extended by anyone able to crate a Component-X component in Java.

Virtual Adapters

One additional requirement was identified for the Component-X adapter architecture – the ability to produce technology independent application service components that allow the selection of the actual protocol to be deferred until deployment time. E.G. You need to be able to put the same component in a "Container" that supports EJB, HTTP or Soap. The requirement for deployment time binding is taken care of with "virtual adapters", these are components attached to the business logic component that communicate with the physical adapters via events. Virtual adapters are "bound" to physical adapters based on information that can be altered at deployment time. The use of virtual adapters is optional, physical adapters may also be directly connected to business logic components.

Copyright © 2001, Data Access Technologies, Inc.



Use of virtual adapters

The above diagram shown how the same web application service component could respond to both SOAP and Http requests at the same time, provided it was within a container that supported both protocols.

Simple & Compound Web Services

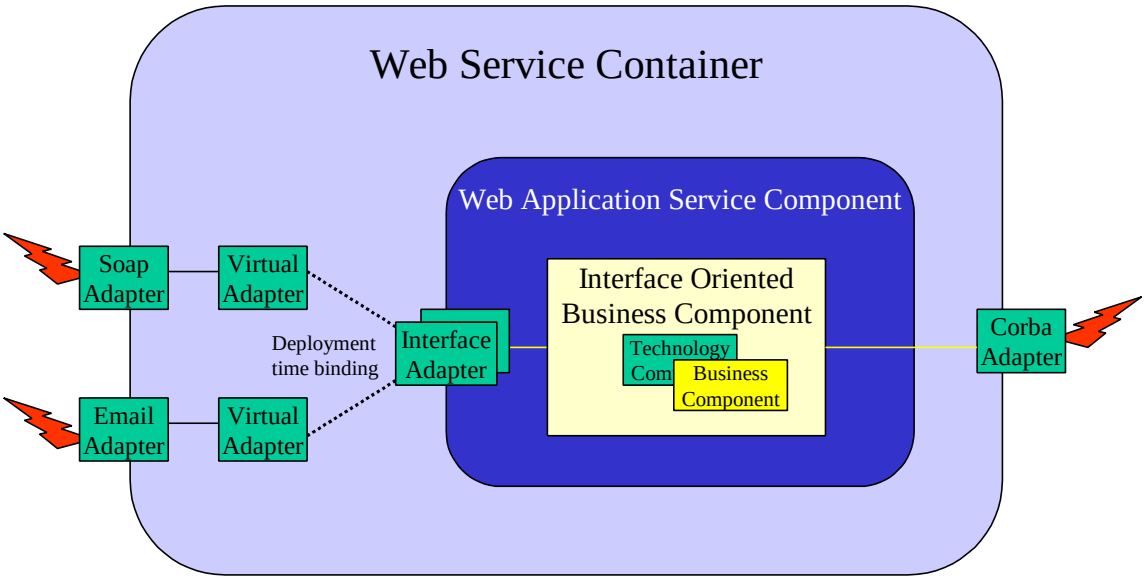
Two basic styles of web service interaction may be found; Those that are oriented to a single message or perhaps a message and reply, and those that describe more complex interfaces or protocols between business partners. HTTP and Soap are examples of message or message/replay protocols while Corba, EJB and ebXML are examples of richer forms of interaction. Component-X supports both styles of interaction with the "flow port" and "interface port", respectively. We will call the richer form "Interfaces" to correspond to excepted terminology. However, we consider that such interfaces include the full contract of interaction between business partners, including message sequencing and constraints.

Simple protocols are generally used for short-running services, those that can be completed in a single interaction or session. Interfaces are generally used for longer running or more complex interactions.

From a technical viewpoint these are not totally distinct. Multiple simple interactions can be used to create an interface out of a simple protocol (this is done explicitly in ebXML) and interfaces can be reduced to a single interaction. To support complex interactions using simple protocols, an "interface adapter" can be used. An interface adapter uses a simple protocol to implement the more complex interface. For example, a single interface may produce and consume multiple Soap documents to implement a single "order processing" interface.

Component-X provides a generic interface adapter to adapt a "virtual" simple protocol to any interface. The interface is attached to the generic interface adapter and configured as to what messages should implement each part of the interface. In many cases the defaults are correct and minimal configuration is necessary.

Copyright © 2001, Data Access Technologies, Inc.



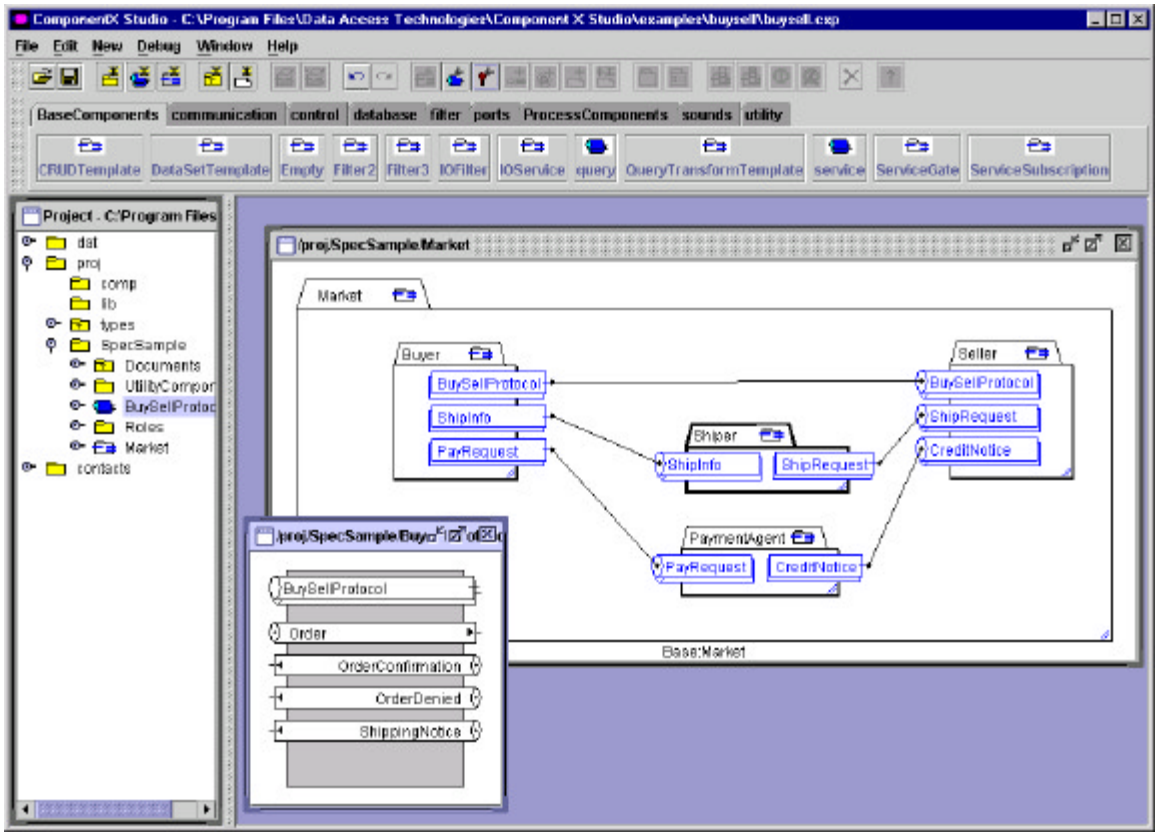
Example of using an Interface Adapter

By providing for both simple and interface oriented interactions, and supplying a generic interface adapter, Component-X is able to deal with all the major forms of web service – from Email to Corba.

Copyright © 2001, Data Access Technologies, Inc.

Using adapters with role based business processes

High level business processes are frequently specified as “roles” that interact to achieve a business purpose. For example, the purchase of goods may involve a buyer, a seller, a shipper and a payment agent. The interfaces are then described between each of the roles to specify the business process. Specific business then decide which role(s) they will play in this business process and implement the corresponding web services. This role based process specification is used in both the ebXML business process specification and the OMG enterprise collaboration architecture.



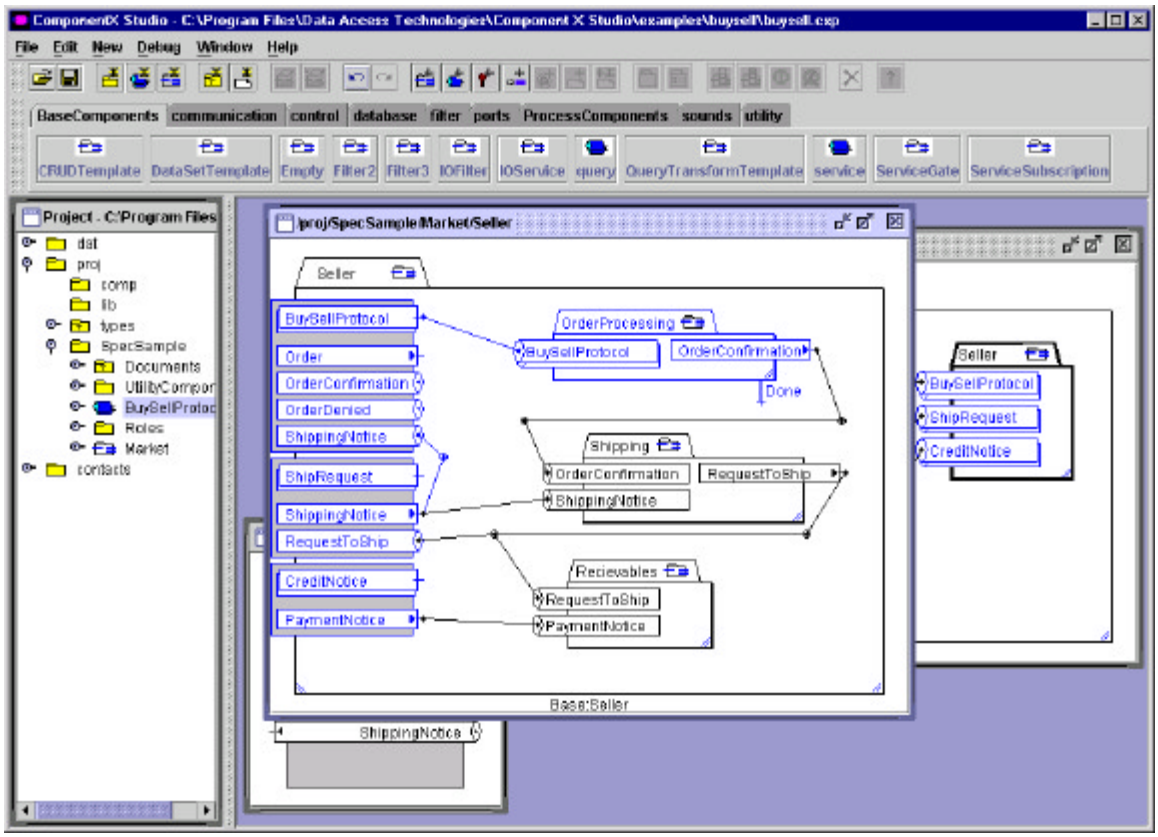
Role based business process

The above component diagram shows the high level view of such a role based business process and one of the interfaces. The roles are; Buyer, Seller, Shipper, Payment Agent. It shows protocols for each role to communicate with the other (note that this is not always the case, many processes involve communications between a subset of the roles). The “BuySellProtocol” details the messages (and associated XML documents) that may flow between the buyer and seller. A business process such as this and/or the associated protocols may be provided by some standard or may be create for a custom process.

A particular business, say “Beautiful Flowers Wholesale” decides it wants to participate in the B2B marketplace and will implement the “Seller” role. This means it must put components into the “seller” that perform the correct business processes and must put adapters in the buyer, shipper and PaymentAgent roles to implement the web services that these other business will use. From the perspective of “Beautiful Flowers Wholesale”, these components are “proxies” for the other companies it will be dealing with. By implementing the seller role and putting adapters in the other roles as proxies, the company will have fully implemented it’s role in this business process.

Note that another company, say “Fast Airfreight” will use exactly the same model but will implement the seller and the buyer as proxies. The shipper does not communicate with the payment agent so it needs no proxy for that role.

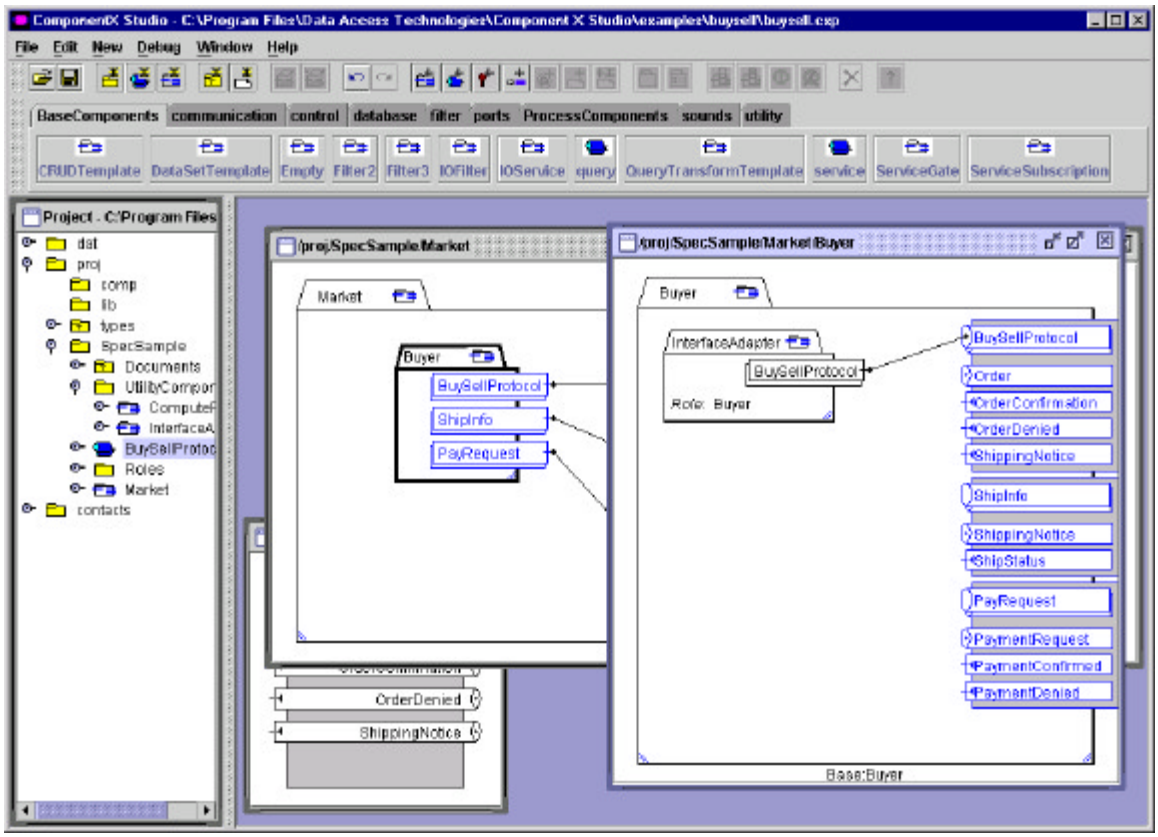
Copyright © 2001, Data Access Technologies, Inc.



Expansion of seller role

The above diagram shown an expansion of the “Seller role” as it may be implemented by “Beautiful Flowers Wholesale”. It shown the interfaces being delegated to internal systems form order processing, shipping and receivables. Each of these activities may be implemented with other components or with legacy systems.

Copyright © 2001, Data Access Technologies, Inc.



Expansion of Buyer from the perspective of seller

The above diagram expands the buyer role from the perspective of "Beautiful Flowers Wholesale". It implements the buyer role as a proxy (using an interface adapter) to a web protocol. When put in a web server buyers will now be able to communicate with the seller's backend system using the buy/sell protocol. If this was put in an EMAIL and ebXML application server, buyers would be able to send orders via email or ebXML to "Beautiful Flowers Wholesale" and get back a response. Note that the buyer role could also be implemented using a web server, so buyers could interact interactively with "Beautiful Flowers Wholesale" Via their web page.

Conclusion

Applying the "adapter pattern" to web services using an open and configurable component architecture maintains the advantages of a multi-tier architecture while providing independence from any one distributed protocol. It also allows for components to be built from other business components, technology components, transformations and web services using a drag-and-drop component assembly paradigm. This has been proven in the implementation of Component-X.

Copyright © 2001, Data Access Technologies, Inc.