B2B Interoperability through Presentation Level Integration

Alan Rickayzen, SAP AG.
Keith Swenson, Fujitsu Software Inc.
December 2002

1 Abstract

While business to business (B2B) integration of business processes is highly desirable, most discussions of the solution revolve around data level integration where process status and properties are passed between the cooperating businesses. This approach has some serious limitations and is in complete contradiction to the billions of dollars that are spent on online vendors' user interfaces. Instead, we propose that B2B integration be done through both data level and presentation level integration so that not only data and process status are shared, but also the user interfaces can be integrated together. This leads to savings for the customer and gives the vendors with good user interfaces a competitive edge. Examples show how this approach can allow service providers and vendors to offer unique capabilities that help users to accomplish their task. As one of the examples in this article shows, this can even lead to extended flexibility in the choreography of the processes, without the complication of having to map a detailed choreography of the processes with one other. The capability to integrate at the presentation level can be built on top of existing standard protocols, such as Wf-XML[1]. Two different approaches are outlined in order to support businesses with varying levels of BPM technology. The technology to accomplish this is readily available, what is needed in only a standard whereby such BPM systems can be linked, without a great deal of upfront setup work. While solutions to enable Presentation Level Integration are provided, these should be considered as suggestions to motivate the real work of developing the standard that will certainly come to be in the near future.

Although this article focuses on B2B integration, the underlying methods are equally appropriate to portal or homegrown Web transaction integration, too.

2 A Missing Capability

A Business Process Management (BPM) system, because of its association with "Business", can be helpful in coordinating the actions of people within an organization, as well as across to other organizations. A process in one company to purchase something could be much more powerful and useful if it could be linked to the associated process in the vendor that is going to fulfill that purchase. Supply chain integration is the motivation behind the standards developed and promoted by RosettaNet[2]. OASIS[3] has set a goal to set up standards to enable B2B integration of business processes in order to promote better efficiency in our markets.

While there are a plethora of B2B scenario examples, most of the solutions are oriented toward communicating the information needed for a business transaction from one business to another, and toward standardizing the choreography of interactions that must take place. In such interactions, a message that can be understood by both sides must be formally agreed upon for the data transfer to take place. This is understandable since the automated systems on both ends must not leave room for ambiguity in the interpretation of any of the items of data. This works well for reasonably formalized interactions, such as between an auto parts retailer, and a parts distributor – an on-line catalog can list all the available parts along with options, and the part can be ordered by part number, option code, quantity, etc.

The limitation of data level integration appears in two ways. Agreements can be made to support interactions of surprising richness. But those agreements take time to set up, and as such are difficult to change in response to changing conditions. When agreements are made between three or more parties, the agreements tend to be along the lines of the least common denominator – only those things that all parties support will tend to be included in the interaction. The ability for a given vendor to provide a better and unique level of service is diminished.

Some businesses are limited by the ability of a customer to find the thing they are looking for. For example, an online bookstore needs powerful search features to help the customer who does not remember the exact title and author, but knows a few fragments about the desired book. Such bookstores can help one in finding books that are similar to other books that you like, for example books recommended by other customers. With a machine-to-machine interface instead of support for a user interface, a customer may find it impossible to complete a purchase. In addition the vendor that has invested time and money in a unique ergonomic user interface loses the competitive advantage.

A hotel, or other service organization, distinguishes itself from its competition by the level of service it offers to clients. Success of these organizations is often dependent upon being able to provide new and unique offerings. The more that a hotel can find out about the preferences of a customer, the better it can meet those needs. It is a significant factor that the customer reserving a room often has a profile stored on the remote hotel-chain’s system, rather than stored on the business system calling this service. The lifespan of the customer’s personalization in the hotel chain system may well be greater than that of his master record in the source business system so it is important that this can be reflected in the presentation layer. It is the presentation layer that merge the customer’s personalization data with the business data (such as the corporate id) coming from the B2B system, and this will be reflected in the results returned to the business system.

A company that has a purchase order approval system, that allows members to purchase books and computers, will want to link those purchase orders directly to the book store or the computer vendor. It will clearly not be possible for the purchase order approval system to provide the power to configure a personal computer, capabilities to search for a particular book, or the flexibility for unique services offerings.

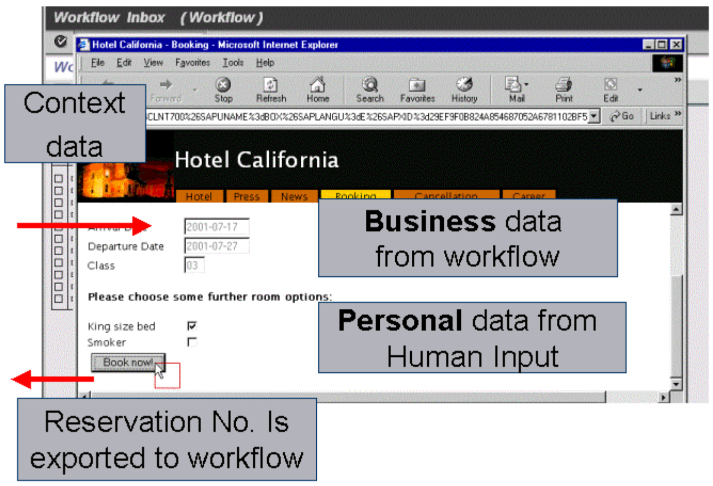
3 Proposed Solution and Examples

The solution is to integrate not only the data level, but also at the presentation level. The company purchase order system needs to provide a UI for those activities required by the company for budget control and approvals, but bring in the UI from the vendor for supporting what it does well. A couple of examples will help guide the discussion of this capability.

The first example is that of a purchase order system, and an online book retailer. A user goes to the PO system, and starts the process of buying the book. Before the purchase can be approved, he must list the books that he wishes to buy along with their cost. He is presented with possibly a number of links to various online bookstores. He chooses his favorite, and navigates off that store's external web site. He browses around a bit and finally selects one or two books before clicking pre-order which not only returns the user's browser screen to the PO system but also returns a key to the book selection to the workflow, with additional data such as cost and delivery dates. The key might be a set of ISBN numbers, or it might be the unique id of a shopping cart that holds the goods that the user has selected together with the price and one or two other bits of business information. This key together with the rest of the returned data is stored in the PO system process instance so that a day or two later the manager can view the book using the bookstore's external Web site before making the approval. The manager sees not just the title of the book to be ordered but can also view the book, its ratings, online reviews, and alternative books in as much detail as necessary before making her decision -- as well as viewing an image of the cover in case that might be help in judging the value of the book. Once the order is approved and routed through the company's required steps, the key once again becomes useful in putting together an order and officially submitting it to the online bookstore. After the order is placed, the purchaser can follow the progress of order fulfillment, including tracking shipping and so on.

The second example is for a travel authorization (TA) system to allow the traveler to book a hotel room. The user is planning to attend a conference. From the conference web site, there are some links to some of the nearby hotels. He finds a hotel for which he can get a special discount rate due to membership in some affinity club, and he knows that there are a limited number of such discounts available, and the hotel is likely to fill up early. So he goes ahead, using the hotel web site, reserves a room with the discount rate. At this time he can go ahead and specify his favorite options, such as a non-smoking room, vegetarian breakfast option, and his choice of morning newspaper. A few days later he finally gets around to starting the TA process. This process has links to major hotel chains to aid in hotel reservations. He navigates to the hotel he has already reserved but instead of creating a new reservation he simply locates his previous reservation, and *attaches* the reservation to the TA process instance. Once attached, the TA system can pick up standard hotel data items, such as address, reservation dates, and price. The user might also attach reservations for air travel and conference registration. The process can proceed through approval and provide a unified itinerary for the traveler. If, the traveler decides not to attend this event, it might be possible for the TA system to automatically cancel the hotel reservations, but at a minimum it will allow navigation to the hotel's web site to allow the reservation to be cancelled there.

Of course, this business scenario can also be dealt with using the same algorithm as in the first example. In other words the initial reservation is made as one of the first activities within the process and the corporate customer passes the context data to the Hotel Web site. However we have deliberately described the alternative method (private reservation "attached" to the process) to illustrate the flexibility, which may be required and the services (e.g. *attach* as opposed to *select*), which may be called.


Workflow Inbox (Workflow)
Hotel California - Booking - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Forward Stop Refresh Home Search Favorites History Mail Print Edit
CLNT700%26SAPUNAME%3dBOX%26SAPLANGU%3dE%26SAPXID%3d29EF9F0B824A854687052A6781102BF5
Go Links
Context data
Hotel California
Hotel Press News Booking Cancellation Career
Business data from workflow
Personal data from Human Input
Book now!
Reservation No. Is exported to workflow

Departure Date 2001-07-27
Class 03

Please choose some further room options:

King size bed ☑
Smoker ☐

Figure 1

In these examples it is clear that the business benefit is achieved by merging the business data, with the presentation data (including the user's personalization on the vendors Web site). It is now time to see how this could be achieved.

4 Detailed Analysis

Lets look closer at the examples to understand what would be required of such a standard.

We clearly want to build on existing standards as much as possible. With so much acceptance around XML, SOAP[4], and web services, it is clear that we should try to find a solution based on this.

In general, we are discussing integrating two services together. These services may or may not be workflow/BPM services. In many cases, the service on one end of the other may be nothing more than a simple web application that has the use of a database. This service might be considered a *trivial workflow*: the process has only a single activity; the process stays in the same state for its entire lifetime; and there is no routing of work to anyone else. In some cases, the system on one end might not need to store any data on its side of the connection. We place these assumptions on the scenarios:

- There is a source service and a target service. The requirements are not the same for each of these. It is possible for a single service to act both as a source and a target, but not all services are required to fill both roles.
- Both kinds of services must have what we call *service instances.* The service instance is place where all the information about this particular case is held. A key uniquely identifies the service instance. For example, in the case of a hotel the service instance is known as a reservation, and the key is composed from the reservation number.
- The source service instance must be persistent – the service must store the information about the case somewhere safe such as a database.

- The target service instance will usually be persistent, but we will show an example where the target service does not need to have a database.
- The target service must be publicly available. It needs to have a name, and be available for requests at any moment. In other words, it must act like a web site. The service might be anything from a simple web application to a workflow system with a web interface.
- The source service can be hidden safely behind a firewall, has no need for a public name, and can (in many cases) be run on demand. The source service might be anything from a personal application to a sophisticated enterprise workflow system.
- We normally think of the source service instance as starting the target service instance, but as the second example shows it does not need to happen in that order.

The Wf-XML protocol was designed to support interactions between systems with these characteristics. The current version, Wf-XML 1.1, is not built on top of SOAP. The WfMC is working on a version 2.0 of this protocol based on SOAP. This article will assume that this work is done in order to provide the data level integration. We then can investigate what extensions need to be made in order to allow for Presentation Level Integration. It is worth noting that the Wf-XML protocol relies on an asynchronous exchange of messages, which is ideally suited to inter-process integration but also is ideal for presentation layer integration, where many presentation devices, such as mobile phones or tablet pc's using wireless LAN, rely on packet connections rather than continuous-on connections.

Lets return to our examples. In the first example, the source service is a workflow system that is already providing benefit by handling the approval routing for the purchase. At the activity for specifying what is to be purchased, the system displays a number of links to retailers. There are two possible ways that the resulting link might be caused to work.

The first possibility is that the link is a command back to the source service that causes it to make a SOAP request to the target service factory. The factory creates the service instance, and returns at least two items: the key and the *presentation level key*. The service instance is a shopping cart that will hold all the selected books during the purchase process. The presentation level key is simply a URL that can be used to direct a browser to the UI representation of the shopping cart. The initial SOAP request carried with it some general information useful to a vendor: who the purchaser is, what company they work for, and which promotional deals have been worked out for them. The shopping cart remembers this information, as well as the key of the source service.

This interaction is shown in the role interaction diagram below. Step ① shows the user creating the purchase order process instance. Step ② is the preliminary interactions with the PO instance filling in the process instance data. Step ③ is where the user clicks on a link to a bookstore. The PO instance makes a soap request to the bookstore for a shopping cart. The presentation key is part of the result of the creation, and the users browser is redirected to the UI for the shopping cart. Step ④ shows the user interactions while browsing the bookstore finding the required books. Step ⑤ shows the user clicking a "done" button, causing the shopping cart to make a SOAP request back to the PO instance giving it the final total amounts in question. This last step could alternately be implemented in the other direction e.g. the user clicks a button on the PO instance, and it makes a SOAP request to the shopping cart in order to pick up the latest totals.

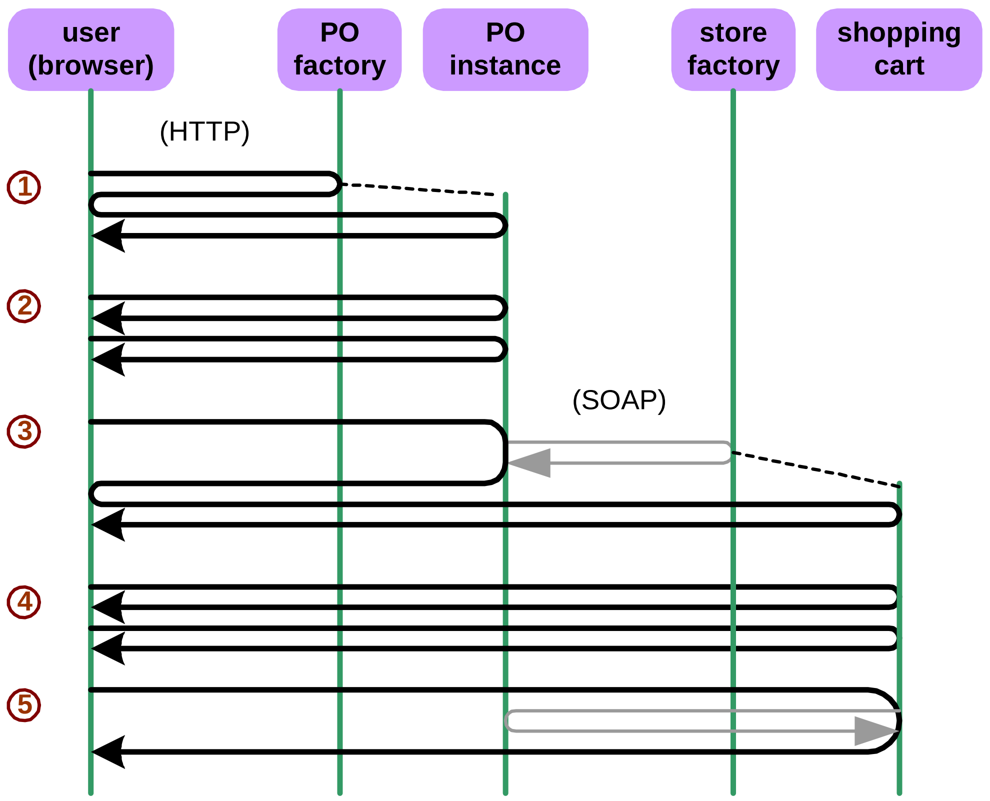


Figure 2. Integration with a Bookstore Shopping Cart

The second possibility is that the link is actually an http link to the UI of the service factory. This URL has coded into it the callback SOAP address of the source service[1] so that the target service can reply later. This HTTP GET request causes the target service instance to be created. The HTTP GET is used because we want the vendor to have the chance to display information directly in the browser. The HTTP GET command is designed exactly for this purpose, taking into account proxies and local caches. It is worth noting at this point that the request for "linking" two running processes is not a part of the Wf-XML 1.1 protocol; an extension would be required to support this. Furthermore, the HTTP GET URL is formed by the source system by combining a supplied URL, and adding the callback key as a parameter; the rest of Wf-XML makes no assumptions about the structure of the URLs themselves, using them only as opaque values. This manner of integration requires that at least one parameter of the URL be standardized, or at the very least configurable.

[1] In Wf-XML terminology this is a data level key known as the *observer key*.

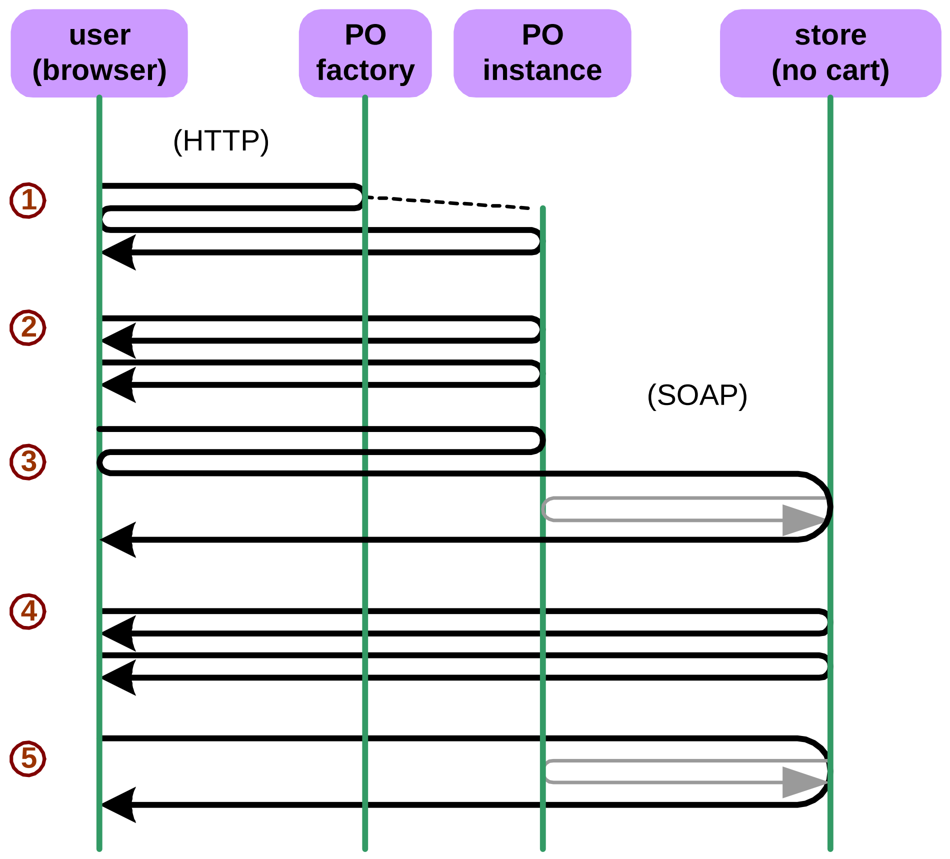


Figure 3. A Bookstore without Shopping Carts

It is the second interaction pattern that allows the possibility that the target service does not need persistence. Steps ① & ② create and initialize the PO instance. Step ③ redirects to the store, but does not create a shopping cart there. However, it does pass the store a callback data level key (URL) as a parameter[5] so that the store can query the source service for additional context data if necessary. This same callback URL is used later in step ⑤ to let the store reply to the source service. Step ④ shows the interactions of the user selecting the books. Finally, at step ⑤, the user indicates that he is finished. The resulting SOAP request back to the source service contains all the state information of the service instance, including result data and a new PLKey that has all the information in it necessary to redisplay the selection of the books. Even though the bookstore has not stored the contents of the shopping cart, the PLKey URL itself could contain all the information necessary to display the book selection to the manager. This algorithm is particularly easy for the online vendors to implement because it simply requires a fixed service address and either one or two SOAP requests initiated in sequence by the vendor. The first request being the optional request for additional information is shown in the last part of interaction ③. The second request is made in interaction ⑤. It is the reply via the callback URL to the source instance containing the results of the service interaction (e.g. books selected, prices and PLKey). Whether or not the target service needs to return structured data or whether the PLKey alone is sufficient depends on the business scenario. However, because in many cases a PLKey alone is sufficient and because this is simple for the vendor to implement, the concept of a PLKey is helpful in this context.

We have considered two possible implementation strategies for one scenario. Now we will consider a completely different scenario to illustrate how the integration of the two services *at presentation level* adds flexibility to the sequence in which the process is executed. Once again, this gives a competitive edge to the vendor offering the better service from a presentation point of view. However in this case it is the flexibility in navigation, which is the significant factor as opposed to the advantage of additional visual information, which we discussed earlier.

This second scenario involves reserving a hotel room. The hotel reservation is made first (a typical scenario when the user is on the road). The hotel system saves the reservation in some form, usually providing a reservation number. Later the user creates the travel authorization request process instance. In order to link them up, the hotel system needs to provide a key that can be given to the TA system that can be manually carried from one to the other by navigating to the original reservation and pressing an ‘attach’ button that has been rendered as a consequence of the method called. This button would not be visible when the customer navigates to the hotel Web site directly. This reservation is the *data key,* which the TA system can use for a SOAP request back to the hotel system, and in the SOAP response the hotel system will return the PLKey for further interactions.

Looking at it from the process point of view, the source process makes a *create reservation* request to the target hotel system. However the target service is sophisticated enough to provide an *attach* feature to give the user the flexibility to create the reservation in advance (in other words the choreography is reversed). The source process will receive a PLKey to the reservation together with the reservation data just as if the room had been reserved at that moment instead of being attached. In other words the source system does not need to be aware of this sophistication. But by offering this sophistication within the service this hotel chain clearly gains a competitive advantage over another chain that does not provide the *attach* feature. Without the presentation level integration this flexibility would not be possible.

Now lets look at this scenario in more detail as shown in Fig 4. At step ① the user is browsing the hotel, setting up for a reservation. Step ② is where he actually makes the reservation. Step ③ is further refinement of the reservation. Step ④ might be much later, and it is the step where the travel authorization instance is created. In order to link the two systems together, the user must somehow requests a link-up page, shown in step ⑤. The hotel displays all the existing reservations that that user has. The user chooses an existing reservation, which fetches a page from that reservation to link up with the TA system, which is does through the SOAP call in step ⑥. Once the two service instances are linked up, browsing back and forth between the systems becomes very easy.

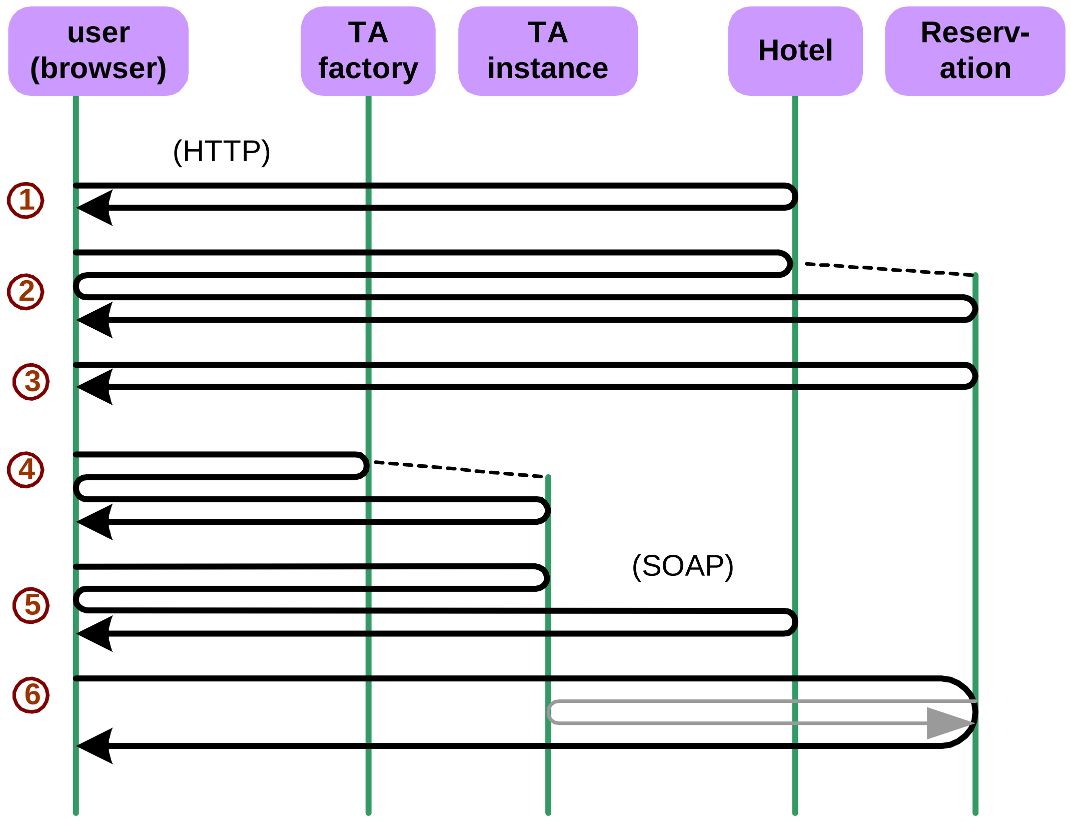


Figure 4. The Hotel Reservation Example

5 Security, Authentication, Authorization

Security, Authentication, and Authorization are three separate topics that are very often confused with each other. Proper handling of all three is critical for success of any publicly deployed system. A detailed treatment of security and user authentification is out of scope of this short treatise, but we can outline some of the particular issues that must be taken into consideration.

Taking proper security precautions means making sure that the data exchanged between two parties is kept safe from being seen by third parties. Secure sockets layer is deemed secure enough for most business applications, and any standard (such as SOAP, SAML, WS-Security) that can make use of this will be sufficient. Digital signatures are being accepted more and more widely, and these are particularly applicable to presentation level integration. Because these pieces of the puzzle fit so well together we don't foresee any hurdles in this area.

Authentication is a trickier problem. Most eBusiness sites require users to create a unique identity for that site. A given user may have dozens or even hundreds of different log-in identities. Remember the purchase order case where the system created the shopping cart for the user, and then redirected the user's browser to the site? The shopping cart should be accessible only by the intended user. The problem is that the user may have a different identity at the bookseller site. In the near term, it will be very difficult or impossible to prove that "jsmith" at one site is the same person as "johns" on a different site. Work is going on in various standards groups, such as Liberty Alliance, to provide to a user an identity that can be used at all sites, but widespread adoption of such a standard is many years off. In the mean time the system must work on the assumption that users have different identities at each site.

Authorization and access control is the main reason that users need to identify themselves through authentication. While the source system may have an understanding of what users can access a target process, it cannot know what those user's identities are in the target system. Therefore, access to the shopping cart must be controlled by controlling access to the link (id) of the shopping cart. The bookseller must allow anyone who knows the id of a shopping cart to have access to the cart, and it conversely must guard the id from being discovered by users who should not have the right to access it. The bookseller must not offer a search function to find shopping carts. The same principle is used in Swiss numbered bank accounts.

Or the mechanism may work in a way similar to that of the online photo sharing sites, where the owner of a photo site can send an *invitation* to others. Users who make use of the invitation are given access to the site. In our example, it is the process instance that stores the *invitation* and controls its availability to the different participants and actions.

However, it is worth pointing out that there are many scenarios, where security is not going to play a significant role. For example, in the purchase requisition scenario, the selection of a book and the viewing of the book in order to make an approval require no security obligations from the online bookstore. Everything is handled by the PO system. True, the agents may logon to the online bookstore using their own private ID's, but this will simply make use of that user's personalization settings to make the selection more comfortable.

6 Other Issues to be addressed

Another consideration that should be addressed is the ease with which a user can abort an online transaction simply by killing off the browser or shutting down the pc. This abort-behavior is something that will happen far more frequently than in server to server integration so it does need special attention. What it boils down to is whether or not the initial call to the remote vendor can be repeated without side effects if the previous call was aborted midway. Two-phase commit logic would help solve this problem but in practice many online stores do not have systems that support this, so relying on two-phase commit would make the implementations more complex and it would restrict the number of B2B vendors that can participate in this integration. This problem could be tackled using session ID's which are generated on a one to one basis for each remote presentation level action, allowing the target service to reject duplicate calls. But there is a simple pragmatic alternative. By restricting the services available to query services (such as locate a book), there is no need to have any persistence in the remote B2B site. The service to order a book, or commit a hotel reservation would be executed in a subsequent step in the business process using a standard Web service based on the data gleaned by the previous query service. Although this is by no means a complete solution, this would still cover enough scenarios to make it useful to both the B2B vendors and the purchasing divisions that control the process.

7 Summary

In conclusion, we believe that a very modest extension to the existing Wf-XML, that of providing a standard piece of data to hold the presentation key, together with the specification of a simple handshaking sequence of two to three Wf-XML messages, can yield a significantly richer integration of two BPM systems but at the same time makes it easy to enable presentation level integration between processes and remote services. This article demonstrates the utility of such an agreement, as well as some of the issues that must be considered in the making of such a standard. This capability is likely to be included in Wf-XML 2.0 when the WfMC releases it.

8 Appendix A - Role Interaction Diagrams

This section explains the details of the role interaction diagrams that are used in the article. Like all role interaction diagrams the vertical axis is time in the downward direction. The vertical lines are resources. The horizontal lines represent interactions between the resources that are at the ends of the lines. The resources that are crossed over are not involved in that particular interaction. Below is a diagram of a simply http request.

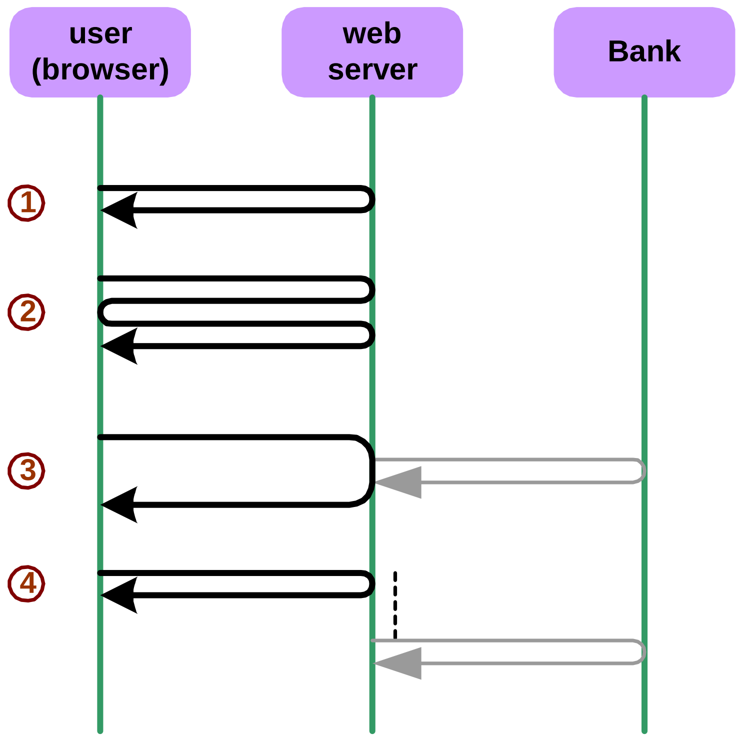


Figure 5. Role Interaction Diagram Examples

Every HTTP interaction involves a request and a response. The request is a block of text transmitted to the web server. The response is a block of bytes sent from the server back to the browser. Since the web server typically handles the request very quickly, we tie the request and the response together with a U bend to tie the request together with its response as shown in interaction ①.

The web is *resource centric*, meaning that every web page is a separate resource with an address. The browser is aware of those resources for which it has an URL. Strictly speaking there should be a vertical line for every possible URL on the server. Of course, this would make the diagram far too cluttered, so we simplify the diagram by combining groups of resources together as significant to the discussion.

In some cases the web server response will be a *redirect*, which causes the web browser to immediately make another request back to the same or to a different web server. This is shown in interaction ②. Such a redirect is usually invisible to the user, who is only aware of the final resulting page. Again we use the U bend to indicate that this request is part of the earlier interaction, and that the user may not be aware that two resources were interacted with.

Most resources exist for the entire duration of the diagram, but some resources are created (or destroyed) during the interaction, and in these cases the vertical line will start (or end). A resource is created by another resource, so a dotted line is used to indicate this.

The client drives the majority of interactions; the server is only sitting and waiting for the next request to handle. When the server does things *synchronously* it does those things before the response is sent back to the user, so we elongate the U bend to show that the request/response interaction is held up for another interaction as shown in interaction ③. But in some cases, especially in the case of workflow, the server will continue processing things *asynchronously*, after the response has been sent back. We indicate these places where the server is continuing to do things outside of a request with a dotting next to the resource line like interaction ④.

9 Appendix B - Wf-XML

We describe here the capabilities of Wf-XML in order to better understand the extensions necessary to provide Presentation Level Integration.

The purpose of Wf-XML is to be able to start, monitor, and be notified of the completion of processes on distant systems without knowing anything about the implementation of those systems. In a way this is similar to a web server serving up documents without the browser needing to know how the server was implemented. SOAP is a protocol that allows for interactions with web services, again without knowing how those services are implemented. Wf-XML is based on SWAP[6] and it was invented before SOAP was defined. Now that SOAP is well accepted as the standard for composing messages between web services, the WfMC is defining Wf-XML 2.0 which runs on top of SOAP. This is coordinated with the Asynchronous Web Services Protocol[7] (AWSP) effort which is an early attempt to build SWAP like capabilities over SOAP. AWSP allows interactions with asynchronous services, but it does not include any workflow features. Thus WF-XML 2.0 is built on top of AWSP, which is built on top of SOAP. What is described below is an early idea of how this protocol will work.

SOAP by itself is not sufficient for this kind of interaction. Clearly SOAP could be used to purchase books on the web. SOAP by itself described only synchronous interaction: a single request followed by a single reply. The transport that SOAP works over can be synchronous or asynchronous (meaning it can work over a direct socket connection, or over a store-and-forward email transport). The interaction has no defined way to start something in one request, and then get back to it in a later request. Wf-XML (and AWSP) include semantics for this. A universal aspect of this situation is that there needs to be an address which can be used in the future to monitor the status. As with everything else on the web, an address is a resource. In the workflow world, this resource is described as a “process instance” but it need not have any aspects of a process other than it can be started and monitored – a simple program invocation can be a process instance in this regard. For more generality we can then call this a “service instance”.

Wf-XML (and SWAP and AWSP) requires an explicit interaction in order to create the service instance. There is a well-known address of a factory resource. A standard request to this factory resource causes a service instance to be created, and it returns the address of this instance. This address is called a “key”, and in this article to avoid confusion it is called a “data key”. The data key is the address which SOAP requests are made to. WF-XML defines standard requests to find out the current status of the service instance.

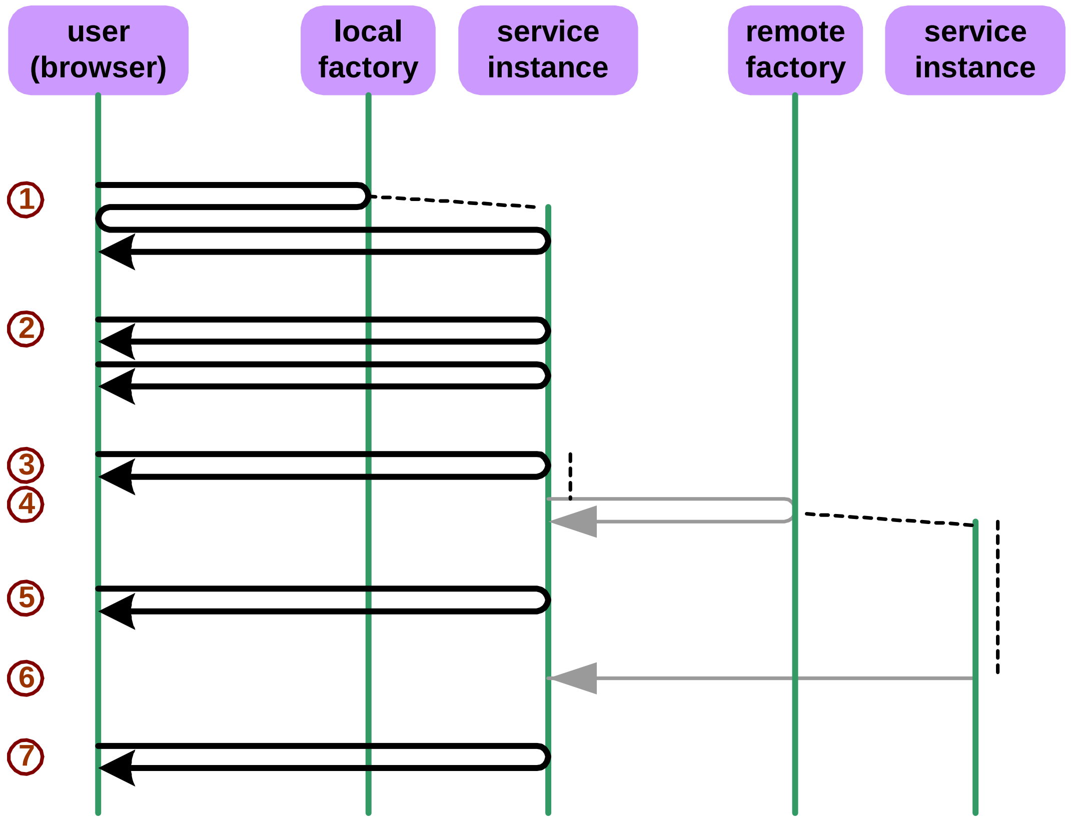


Figure 6. Standard WfXML without Presentation Level Integration-

This role interaction diagram would depict the interaction of a purchase order BPM system that uses Wf-XML to automatically order the requested items. At step ① the user creates a purchase order instance, and at ② interacts with it to fill out the purchase order details. At some point the user completes the activity, and submits the PO to BPM routing. Presumably it will go through various approval steps, which appear to be asynchronous to the point of view of the requesting user. When finally it is fully approved (step ③), the system at ④ makes a SOAP request to the factory resource of the remote system. This causes a purchase service instance to be created, and returns the address to it. That purchase service instance then operates asynchronously over time, tracking the actual purchase fulfillment. At step ⑤, the user checks on the status of his purchase order, and he can see that it has made it through all the approvals, and that a service instance has been created. Eventually, at step ⑥ the service instance will complete the order fulfillment, and Wf-XML provides a mechanism whereby the PO instance can be notified that the remote process instance has completed. Somewhat later, at step ⑦, the user again checks on the status of the PO process, and finds that the remote process has completed, hopefully with favorable results.

At the step ⑤, where the user checks on the progress of the local PO process, it would be possible for his server to go check on the status of the remote purchase service instance at that point in time. This would give the user up to date information on the remote process. But the user is limited to the information about that process that the local system is able to represent. What the user cannot do is to point his browser at the remote service instance, and see the way that it represents its current status. The only key to the service instance is the data key, and not address of the presentation level is defined by Wf-XML. Clearly, the goal then is to provide a standard way for the remote system to provide the presentation key for the service instance, so that the user can see the status directly, using the UI provided by the remote service instance.

10 References

[1] Wf-XML is a server to server protocol for starting, monitoring, and being notified about services. More information is at http://www.wfmc.org/standards/docs/Wf-XML-1.0.pdf
[2] RosettaNet is a consortium of computer makers, resellers, and users creating e-commerce standards for transactions centered data exchanges using a standardized set of terms for product, partner, and transaction properties. http://www.rosettanet.org/
[3] OASIS: The Organization for the Advancement of Structured Information Systems is a non profit consortium that drives the development, convergence and adoption of e-business standards and can be found at http://www.oasis-open.org/
[4] SOAP: Simple Object Access Protocol is standardized by W3C and more information can be found at: http://www.w3.org/TR/SOAP/
[5] This callback URL parameter methodology is described in more detail in the January 2002 edition of SAPinsider (archived on http://www.sapinsider.com).
[6] SWAP: Simple Workflow Access Protocol, http://www.isr.uci.edu/events/twist/wisen98/presentations/Swenson/
[7] AWSP: Asynchronous Web Services Protocol is a protocol based on SOAP which allows for starting, monitoring, and being notified about remote services. Information can be found at: http://www.awsp.info/