Solution design in WebSphere Process Server and WebSphere ESB, Part 1: Solution design in WebSphere Process Server

What do solutions look like in WebSphere Process Server

Kim J. Clark
Brian M. Petrini

May 13, 2009

This article describes how to design service-oriented architecture (SOA) based solutions using WebSphere® Process Server and WebSphere Enterprise Service Bus. Part 1 of this article series explores how design techniques change as an SOA matures.

View more content in this series

Introduction

This is the first in a series of articles that discuss how to design solutions using WebSphere Process Server (hereafter called Process Server) and WebSphere Enterprise Service Bus (WESB). This article focuses on how Process Server is used differently at different stages in the maturity of a service-oriented architecture. To illustrate the high level view of the designs, we use the new solution views in WebSphere Integration Developer.

This article is part of a series that will cover key design aspects of Process Server solutions. The expected content of the other articles in the series is discussed at the end of this article.

How does "integration" change as an architecture matures toward SOA?

Creating a service-oriented architecture (SOA) is never a single step. It is a progressive maturation where each level of maturity builds on the last. There are a few key reasons why this is the case:

- You rarely, if ever, create an architecture from scratch, implementing an idealized architecture in one implementation. Infrastructure, packages, and legacy applications are already in place and you use that as the starting point.
- No single project can take on the cost and time needed to significantly mature an architecture. It requires the wider influence of a longer term program.
Not every part of the application architecture lends itself to the rigor required by a well-governed SOA. Some parts may mature more slowly toward SOA. At any given point, the architecture will be a mixture of the old and new architectural styles.

You don't create an SOA, you evolve it. Each stage in the evolution needs time to mature and settle.

Why is all this relevant to Process Server? Well, many products play their primary role in a particular stage in the maturity of an SOA. Process Server is unusual in this respect because it houses a broad array of capabilities that are relevant at many stages of development. To understand how to use Process Server effectively, you need to understand which parts of the product and which styles of use are appropriate at each level of maturity.

Next, we will look at a standard mechanism for measuring an enterprise's current and target maturity level, and then we'll discuss how Process Server is used at each level.

The Service Integration Maturity Model

The Service Integration Maturity Model (SIMM), shown in Figure 1, is an accepted way of representing the progressive stages through which an architecture and the wider business move toward an SOA.

![Figure 1. Diagrammatic representation of SIMM](image)

It is important to understand where an organization is on this scale, and where on the scale they are targeting before deciding what initiatives, and indeed software, will need to be put in place to affect the change.

Let's move straight on to see what this looks like in Process Server solutions.

As a final word on SIMM, this article focuses on a small part of SIMM - the architectural and application layers - and specifically the integration middleware aspects. SIMM has a much broader...
scope than this. For a more complete picture, take a look at the Related topics section at the end of this article.

Where do Process Server and WESB come into play in the maturity model?

Now that you have an understanding of the progressive levels of maturity, let’s consider the core capabilities of Process Server and how they are related to the SIMM model. Table 1 notes the different solution types that are used at different levels of maturity.

**Table 1. Different solution types**

<table>
<thead>
<tr>
<th>Solution type</th>
<th>Description</th>
<th>Relates to SIMM</th>
<th>Related Process Server components?</th>
</tr>
</thead>
</table>
| Low level connectivity | Native transports, data formats, APIs | Level 2 | • Adapters  
• Export/Import bindings  
• Data handlers |
| Hub-based integration | Improved resilience, consumability and granularity. Hub and spoke. | Level 3 | Mediation flow components |
| Service exposition | Standardize protocols, data formats, service contracts, and policies | Level 4 | • Export/Import bindings  
• Service registry  
• Service gateway |
| Composite services and business process automation | Orchestration and choreography of mature services | Level 5 | Business Process Choreographer (BPC) - for example, the Business Process Execution Language (BPEL) engine in Process Server. |
| Workflow and task-based solutions | Distribution of work via tasks | Various, see detail later. | Human Task and BPC |

Let’s walk through the maturity levels and look at each of these solution types in more detail.

What do the different levels of maturity look like in Process Server solutions?

The next few sections discuss what Process Server solutions might look like when trying to achieve the aims of each of the maturity levels.

**Low level connectivity - SIMM level 2**

These solutions provide connectivity to back end systems directly using additional code or configuration via their native transports, APIs, and data formats. Organizations start building this capability in SIMM level 2. Initially, they write connectivity code into their backend applications. Over time, they make use of packaged adapters. Generally, this area of technology is now well understood, with a combination of APIs provided by packaged systems, productized adapters for common backend platforms, and data handlers for established data formats, such as a COBOL copybook.
However, all the communication is point-to-point as you can see in Figure 2. As more requestors and providers are introduced, exponentially more connectivity code needs to be written, which quickly becomes unmaintainable.

**Hub-based integration (basic) - Lower SIMM level 3**

The simplest possible use of Process Server or WESB is a hub to connect requestors to providers where they do not have an existing common connectivity technology. The presence of a hub marks the beginnings of SIMM level 3.

This can be implemented purely in WESB because it involves only a mediation flow component and WebSphere Adapters as in Figure 3. At this maturity level, you typically see adapters on each side, on the assumption that neither requestor nor provider can implicitly perform connectivity over common standards. This means that the solution is relatively specific to this requestor and provider, and re-use requires, at a minimum, further adapters and maps.

**Figure 3. Solution for a basic integration scenario - a single mediation flow component**

For any given request made on the export on the left, only a single request is made through to the import and on out to the provider on the right. Process Server’s only role here is to provide the low level connectivity such that the requestor can communicate with the provider. The adapters are doing most of the hard integration work.
Figure 4. Inside the mediation flow component of Figure 3

The mediation flow component (Figure 4) performs a mapping of the requestor’s data model to that of the provider, and perhaps performs some further logging for diagnostics or auditing purposes.

Note: Even though we’re only doing one-to-one connectivity here, we can’t describe this as "point-to-point" (for example, SIMM level 2). The fact that there is a hub present at all suggests this is SIMM level 3 since we already have some opportunity to re-use the integration. Pure level 2 is custom code either within or close to the requestor/provider applications.

Hub-based integration (advanced) - Upper SIMM level 3

The simple integration soon becomes insufficient. Additional integration patterns need to be applied to the low level connectivity to make the interface to the backend system generally more resilient, consumable, and suitable granularity for re-use.

The presence of multiple requestors or providers connected via a hub leads to the recognizable hub and spoke architectural pattern (Figure 5), which is typical of more advanced SIMM level 3 solutions. This is primarily performed by mediation flows, although in some cases, BPEL processes may be used. Typically, the requestors and providers are connected via adapters.

Figure 5. Hub and spoke solution

It might be tempting to think that if the protocol used to expose to requestors were a standard protocol, such as HTTP/SOAP or HTTP/REST, we have immediately moved to SIMM level 4 by providing an apparently re-usable exposed service. We will see in the section on service exposition that it is not that simple.

This simple looking diagram (Figure 5) hides a number of variants. Let’s consider each separately for a moment:

- **Router**: Each requestor is routed to a specific provider for a given request.
- **Translator**: Multiple requestors are set up to have their requests routed to specific providers for a given request.
- **Low level composition**: Each requestor makes a request, using multiple providers. Note that this is different from the service composition that we will discuss later. Here, we are creating
composing solutions using traditional low level interfaces to backend systems rather than using mature services.

Combinations of the above are also possible and common. Multiple requestors can have their calls translated, then routed to different components that perform an appropriate low level composition to complete the interaction.

Consider how the solution might be broken up differently for each of the above cases. First of all, let's look at the router shown in Figure 6.

**Figure 6. Basic router solution**

![Figure 6. Basic router solution](image)

We have more than one provider, and we can use a mediation flow component to make the choice between them. The mediation flow might use a message filter to perform, for example, content or header based routing as in Figure 7. Once the provider has been chosen, a specific transformation of the data is done to convert to the data model of the provider.

**Figure 7. Basic router - mediation primitives within the mediation flow component of Figure 6**

![Figure 7. Basic router - mediation primitives within the mediation flow component of Figure 6](image)

Now let's contrast that with the translator (Figure 8). This handles requests from two different sources and translates them so that they can call the same backend provider.
Figure 8. Basic translator

Notice that we've used two different mediation flow components for the two different routes. It is possible to do all this in a single mediation flow component. Since the two different inbound routes are often different in character, it is better to keep them separate. This way, they can be developed and maintained independently and their responsibility is clear. As you will see later, if their interaction is more complex, it is even possible that they may deserve their own modules.

Now what's interesting is if you add the translator and router styles together (Figure 9). Remember, each of the requestors has its own data model, and so do all the providers. If we approached this simplistically, then for each requestor, you map the requestor's data model directly to each of the providers. If there are many providers, you end up with an exponentially growing number of maps to create - similar to the mess we were trying to avoid back in SIMM level 2. We're not making the best use of the benefits of having a central hub because we haven't introduced a central or canonical data model.

Figure 9. Combined translator/router solution (see enlarged Figure 9)

In the example above, the inbound mediation flow components translate into our canonical data model, then the router mediation flow component performs re-usable translations to each of the providers' data models. The canonical data model means fewer maps to write, and also if the data model of either a requestor or a provider was to change, it only affects their specific mapping to or from the canonical model.

Let's move on to composition (Figure 10). You need something to gather together several requests to the backend systems.
It is possible to perform some level of aggregation of invocations to providers using a mediation flow component and one or more service invoke primitives, and in simple cases, this may be the right solution. This is sensible if the logic to be performed clearly represents integration logic, such as data enrichment or simple aggregation. Using mediations may have performance advantages if you can keep the data format to XML and gain the benefits of XSLT transformations without parsing the objects. However, even small amounts of complexity introduced into the composition can suddenly make it preferable to turn to the capabilities of BPEL instead. For example, if you require more complex conditional logic, cyclic flow, compensatory capabilities, persistence of state, or in-process human interaction. In these cases, use BPEL for the solution, especially if the process itself is of interest to the business from a monitoring point of view.

If BPEL is used, it is important that BPEL is controlling the high level steps, not being used as a visual coding language to perform low-level logic. This is the reason why we have retained mediation flow components next to each of the adapters in Figure 10. This ensures that all possible integration logic is pulled out of the BPEL process. Remember that one of the benefits of BPEL is its visualization, both at development time and at runtime. If something is going to look complicated if you do it in BPEL, then you probably need to push that detail outside into a mediation, or perhaps out into the provider application.

It is interesting to see if this solution using BPEL is different from the solutions using BPEL in SIMM level 5, where services are more mature. One of the primary issues is that the interfaces to the providers here haven't been matured and exposed properly, so there will be significantly more complicated integration logic to perform. Since this low level integration is complex, it is likely that this solution is more costly and time consuming to implement than an equivalent level 5 solution. This is, of course, the reason you want to mature the services in the architecture in the next stage.

**Note:** We've been discussing service maturity, and yet we've hardly mentioned the term *service*. This is important and completely deliberate. There is much ground work to do in improving integration and also in understanding what are the core re-usable functions of the business before we can start talking about exposing services.

**Service exposition - SIMM level 4**

As an SOA reaches SIMM level 4, a selection of key business and technical functions surfaces as candidates for re-use. Initially, they may be re-usable within the IT department, but some may have wider opportunity at the enterprise level, or even on the Internet. SIMM level 4 is all about how you find these services (which is out of scope for this article - see *Service-oriented modeling and architecture*), and technically how to expose the service, which is what this section focuses on.
By *service exposition*, we mean exposing the interfaces using standard protocols, data formats, service contracts, and policies to enable significant re-use by diverse consumer applications.

So what's so challenging about exposing a service for re-use? We've said already that choosing an interoperable standard protocol, such as SOAP/HTTP or REST/HTTP, is only a fraction of what we mean by "exposing" a service. What else is there to do?

To expose a service usefully, it must at least be: valuable, robust, reliable, performant, usable, monitorable, maintainable, and secure. We'll look at these qualities in much more detail in future articles. For now, you need to recognize that most interfaces prior to this stage were developed for a specific purpose and for specific requestors, and so we provide only a small selection of the above qualities. They address the qualities required by a specific solution rather than considering all possible consumers of the service. It is the standardization of these qualities that is the key to service exposition, and therefore, to SIMM level 4.

Let's look briefly at how and where these extra capabilities can be designed into a solution.

**Figure 11. Service exposition**

Four things have changed as we progressed from the previous hub and spoke architecture to service exposition, three of which you can see by comparing the previous figures with Figure 11.

1. **Standardized exposure**: Requestors no longer require a connector to talk to the hub since we have agreed on standard interoperable protocols and transports, such as SOAP/HTTP or REST/HTTP.

2. **Standardized request handling**: We have dedicated part of the hub, the ESB Gateway, which performs the aspect oriented capabilities of virtualization, security, visibility, and traffic management required for mature exposition of services.

3. **Service registry**: A service registry has been introduced since you need a publishing service so that requestors can find it. The dotted line represents the registry that may initially only be used at development time rather than at runtime.

4. **Operation specific integration to improve consumability**: The fourth change isn’t visible in Figure 11 because it’s a change in what the hub is doing, but the hub was already there. The "non-ESB Gateway" part of the hub was, and still is doing, the deeper more operation specific, integration logic. Where before it only performed the minimal logic required by the known requestors, it now performs any reasonable logic that makes this
a more consumable service for any requestor. These might include the granularity of the error handling, management of duplicate requests, handling re-tries, scheduled downtimes, subtleties of connection sharing, and many more. These are the same integration patterns we were using in SIMM level 3. The point is that you are catering to more consumers, so you have more requirements, and hence there is more integration work to do.

Let’s see how that is implemented in a Process Server solution.

**Figure 12. Solution diagram for interfaces exposed via a service gateway (see enlarged Figure 12)**

So what's different about Figure 12? Each part of the solution is internally different, so let's go through it in detail from left to right.

- **Standard protocol/transport for exposition:** We’ve chosen a standardized way of calling the service. In this case, *ServiceExport1* is a Web service export. Most modern systems can call over Web services, and WSDL is a well-established way to share the definition, either in a registry, or directly. The service is easy to find and call.

- **Standardized handling of requests:** We’ve introduced a special mediation flow component called a service gateway. This is similar to a normal mediation flow, but it allows you to define integration logic that will be performed on all requests, regardless of the fact that they may be different operations. This makes it easier to perform the “aspect oriented” capabilities that you need for the services to behave appropriately as part of a governed SOA. This component is responsible for:
  - Exposing the service securely (encryption, identity management, authorization, and so on)
  - Making it visible (logging, auditing, monitoring, and so on)
  - Making it maintainable (virtualized, versioned, configurable, and so on)
  - Managing traffic (throttling, load balancing, routing, and so on)

Part 2 of the article series will look more closely at how these aspect-oriented capabilities are achieved.

- **Operation specific integration to improve consumability:** There are specific mediation flow components for each of the operations to backend providers. In these, you implement the operation specific integration logic needed to resolve mismatches between the characteristics of the providers and those of the desired exposed service.
**Note:** Definitions of the purpose and boundaries of an ESB are notoriously hard to agree upon. An ESB is an architectural concept, and thereby doesn't live independently within any concrete components in the diagram shown in Figure 12. A concrete capability is the *service gateway* component, which is critical to the ESB pattern. It is the service gateway that makes it possible to expose mature enterprise services in a standardized and governed fashion. It is one of the main things that extends and differentiates *service exposition* (SIMM level 4) from the more traditional integration, such as *hub and spoke* (SIMM level 3).

**Composite services and business process automation - SIMM level 5**

From the previous stage, you now have some fully matured and consumable services (Figure 13). The next logical step is to see where these are often used in common patterns in your day-to-day processes. If so, you have an opportunity to wrap them up into new *composite services* that orchestrate or choreograph the lower level services. This further improves the maturity of the SOA, providing more powerful services to consumers.

**Figure 13. Composite-based solutions**

Simplistically, for now, we will break composite services into two flavors:

- A *basic composite service* is nothing more than a commonly used set of service requests wrapped up into a single service request. All the service requests are probably synchronous and complete in a reasonable span of time (for example, seconds or less). These are typically implemented using a short running BPEL process.
- Some composite service are inevitably created to manage processes that stretch over a much longer time span of perhaps days or even weeks. It is likely in these cases that visibility of where you are within the steps of the request is important to the business. These are, therefore, more clearly recognizable under the label of *business processes*. These are typically implemented in long running BPEL process, and may involve human interaction in the form of Human Tasks in the process.

Regardless of the type, the solution looks similar to Figure 14.
There's more to say about the different types of process solutions. In fact, we'll save that for a future article on process implementation types.

Let's take a look at a more advanced Process Server solutions that involve a few operations that are properly exposed through a service gateway and are, in themselves, composite calls to backend systems.

Figure 15. Solution diagram involving exposed composite service operations (see enlarged Figure 15)

In Figure 15, you can see that the two separate service operations are exposed via a service gateway. The gateway performs all of the aspect oriented capabilities noted above to expose the service operations, then routes the requests based on the operation called to separate modules. These process-based modules perform the composite services, choreographing multiple calls to potentially multiple backend systems using BPEL. You'll notice that the imports on the right of the process modules are not adapters anymore, since we are calling mature services, probably over Web services. You'll also notice the absence of mediation flow components in the process modules. That's not because you can't have them in the same module. Starting in WebSphere Process Server v6.2, you can have processes and mediation flow components in the same module. The reason is, again, because we're assuming that you are choreographing mature services and there is no need for any complex integration logic. We have a simple interface map with underlying business object maps to translate between the data model used in the process, and the data model of the services we're choreographing.
So ideally, only mature services are choreographed to avoid integration logic cluttering and constraining the business processes. That is where the greatest benefits are drawn from process automation and why it is ideally targeted at SIMM level 5 to build on the matured services of level 4. Note that it is rare for all of the interactions required by a process to be mature, well exposed services when we are only just reaching this level of maturity. Therefore, it is common to see processes performing a mixture of requests to mature services and traditional interfaces. Care needs to be taken to ensure that these traditional interfaces are suitably decoupled from the business process to improve maintenance and re-usability.

**Workflow and task-based solutions**

Workflow is a special category when discussing service integration maturity because they are not innately linked to integration. Workflow refers to systems that allow efficient distribution of work between teams of people by breaking the work into tasks and automating the distribution and progression of those tasks (Figure 16).

**Figure 16. Tasks only solution - no integration requirements**

This stands out as something that will not fit onto the SIMM. For a start, it does not necessarily require any direct integration with backend systems at all, since the user task may simply involve making a phone call or posting out some paperwork. Backend systems are often in use for some of the tasks, but the workflow system doesn't necessarily need to integrate to those backends. The user may switch from their desktop to a mainframe terminal to perform the task, then switch back again. Products, such as IBM MQ Workflow and its predecessors, have been around as standalone systems for decades when companies were still at SIMM level 1, and have grown to significant sophistication in parallel with architectural concepts like SOA.

However, the advance toward service-oriented architectures brought new possibilities to the workflow, and many have inched across toward process automation by allowing some of the tasks to be "system" tasks (services) rather than "human" tasks. Describing it differently, you can see a workflow as a process that makes a number of requests to services, which are performed by a mixture of people and backend systems. Initially, most of those tasks are done by people, but over time some of them are replaced by calls to services, and ultimately to a fully automated process.
Figure 17 compares a completely human task based process with one which is mostly automated but for rare exception cases.

**Figure 17. Tasks only solution - no integration requirements** *(see enlarged Figure 17)*

![Diagram comparing a human task based process with one that is mostly automated for rare exception cases.]

Process Server allows human tasks to be interspersed within and around BPEL processes so you often can't automate every step in a process on day one. However, do not limit the use of human tasks to these occasional steps in integration-based processes, and do not ignore the wealth of knowledge built up on workflow systems over the years. You must also recognize that many processes are fundamentally people oriented and will remain so for some time to come.

Process Server offers a significant number of ways to build human task-based processes. These processes will also be discussed in more detail in a future article on *process implementation types.*

**Conclusion**

This article looked at what typical high level solutions might look like for process and integration problems that are commonly solved using WebSphere Process Server and WebSphere Enterprise Service Bus.

In this article series, we will be drilling down to look at what integration and process solutions look like in detail. We will also describe how to capture and translate key solution characteristics and translate them into the patterns within the design. In each article, a key new feature recently added into the product will be introduced. Here’s the content summary of upcoming articles:

- **What do solutions look like in WebSphere Process Server** *(this article)* describes using *solution views* to consider designs from a high level.
- **Integration characteristics, patterns, and the Enterprise Service Bus** will introduce the *new service gateway pattern.*
- **Process implementation types** will explore the broadening capabilities of process and task based solutions, such as *collaborative flow.*
• Building flexibility into solutions with versioning and dynamicity will take a look at existing and new options for process and module versioning and options for making processes flexible.
• Using patterns in solutions will look at how JET2 can be used to increase productivity and improve consistency.

Acknowledgements

The conclusions in this article are gathered from our discussions with people on design topics, project experiences, and also from our conversations with people involved in the creation of the product. Our thanks to at least the following people: Andy Garratt, Bobby Woolf, Eric Herness, Geoff Hambrick, Greg Flurty, Helen M Wylie, Jonathan Adams, Joseph (Lin) Sharpe, Rob Phippen, Stephen Cocks, Paul Verschueren, Werner Fuehrich, and in reality, many more.
Related topics

- The Information Center describes the solution view used to create many of the diagrams in this article and also the service gateway capabilities important to service exposition.
- **Enterprise Integration Patterns** by Gregor Hohpe and Bobby Woolf defines the base integration patterns, most of which have progressively been productized in products, such as WebSphere Enterprise Service Bus, WebSphere Message Broker, and WebSphere DataPower.
- **Enterprise Connectivity Patterns** by Peter Lambros and Helen M Wylie puts some important detail behind the broad range of architectural integration patterns.
- The developerworks article series **Exploring the Enterprise Service Bus** by Greg Flurry provides core definitions for the term Enterprise Service Bus and practical advice, both architecturally and in terms of IBM products.
- **Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB** is a product-focused look at how key IBM integration products are used within an SOA. It also includes notes on DataPower, WebSphere Service Registry and Repository, and WebSphere Transformation Extender.
- Bobby Woolf's article **ESB-oriented architecture: The wrong approach to adopting SOA** is a valuable wake-up call that an ESB is only part of a broader picture.
- For a practical approach on defining processes within an SOA, take a look at **Process oriented modeling for SOA** by Ruud Schoonderwoerd.
- **Increase flexibility with the Service Integration Maturity Model (SIMM)** by Ali Arsanjani, and Kerrie Holley provides a basic description of the SIMM, which has since been adopted by the open group.
- **Service-oriented modeling and architecture (SOMA)** was first discussed back in 2004 and has now matured into a fully fledged methodology with tooling available to accelerate the learning process.

© Copyright IBM Corporation 2009
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)