Service concepts are leading the industry down a road that results in software componentization as an inevitable response to increasing complexity and demands for adaptability at all levels. In this article we report on a detailed Q&A with the IBM Software Group product development organization in which they share their experience and provide practical insight and guidance.

By David Sprott

Practical Service Specification and Design Part 2a: Refining Service Requirements for Consumers

How do we decide the functionality of a software service? The Service Portfolio Plan has identified the services required, and defined their broad responsibilities. Next, we need to determine the specific requirements for each service: the actual functions (operations) it needs to offer, and any non-standard quality requirements. This part work looks in depth at the needs of Consumers.

By John Dodd

CBDI reported on PolarLake when it was first launched and was impressed by its capabilities. Since then it has matured into a more complete product and is now positioned as an Enterprise Service Bus (ESB). In this report we look at the new features and assess PolarLake’s ability to act as a bridge between new Business Service requirements and the diverse range of resources in use in most organizations.

By Lawrence Wilkes

In this report on the Systinet Business Registry we look at how SOA governance can be improved through the provision of a set of lifecycle services.

By Lawrence Wilkes
Back a few years ago the topic of software components was getting a high level of interest in the technical community. The idea of somewhat coarser grained units of functionality and reuse was and is extremely sensible. But it was pretty clear that software components were a technical level matter and that while development managers were very happy for developers to design and build components, it was hard to persuade anyone of the higher order benefits, and even harder to convince managers that it was desirable to invest to develop generalized components for wider reuse. Generally the cost/benefit equation didn’t produce a sufficiently positive result, and components had no obvious connection to business issues.

Today there is a much higher level of interest in structural matters. The intensive industry debate around SOA has encouraged many architects to address the underlying structural issues that are constraining business change. But as architects understand more and more about SOA they generally come to realize that the notion of federation is equally applicable to the entire functional and non functional stack.

In the beginning many in the industry encouraged a wrapping approach to publishing services, because it was the only way to make progress. But the limitations of this strategy are plain for all to see. The phrase “agility is skin deep” comes to mind. So it was always inevitable that sometime soon the industry and its customers would wake up and ask hard questions about the structure of widely used enterprise applications, as well as the home grown applications and middleware portfolios.

The software component model is pretty well understood by now. The component is a unit of functionality that is implemented with known functional and non functional dependencies and offers its services through formal interfaces. What’s interesting is the convergence of thinking that’s happening right now between service and component architectures. The component, like the service, is a highly generic concept. Because of this generality they are applicable in presumably infinite application and technology use cases. By definition also they can be extremely fine grained or relatively coarse. It all depends on the context.

What’s needed is a clear recognition that we need to view the service and component architectures as part of the same structural set. The service is the external view of the component capabilities, and the component is the implementation perspective. Both views provide different structural characteristics that allow different forms of adaptability and response to change.

This month our lead article reports on the experiences of IBM’s own software group development organization in deploying component and service based architecture. There are many interesting insights in this piece, but the one that will stick in my mind is that they build the components and services with no constraints as to how they will be published and consumed. That is by design, a deployment issue. The components and services at all levels are simply part of the overall SOA.

David Sprott, CBDI, April 2005
IBM’s Experience in Componentization

Service concepts are leading the industry down a road that results in software componentization as an inevitable response to increasing complexity and demands for adaptability at all levels. No surprise therefore that we observe leading edge organizations starting to adopt highly organized and automated component based development strategies. In this article we report on a detailed Q&A with the IBM Software Group product development organization in which they share their experience and provide practical insight and guidance.

Introduction
For the past five years the industry has been very focused on services. First Web Services, then Service Oriented Architecture (SOA) and more recently Service Oriented everything, otherwise known as Service Orientation. These changing interests reflect the progressive maturing of architecture. In the early stages of service thinking, the most common deployment approach was to wrap existing systems and to publish Web Services that solved some form of interop problem. The widespread adoption of SOA shows there is a high level of understanding that creating loose coupling at the transport layer only solves a narrow range of problems, and that to provide business adaptability, loose coupling needs to be effective at many levels.

Some time back we forecast this maturing process would bring us full circle, to the point where software componentization would become recognized as a critical strategy for mature organizations. It was always inevitable that this line of thinking about loose coupling and adaptability would result in genuine componentization at all levels and today we are seeing leading edge organizations embracing component based strategies as part of their overall SOA direction.

Back in late 2003 I discussed this issue with the IBM WebSphere architects. At that time they indicated they had commenced a program to introduce componentized architecture principles to the WebSphere product line. Late last year I met with the same folk again and was really interested to hear they had made considerable progress.

IBM is now engaged in a process to componentize its entire product portfolio. This major undertaking was commenced over a year ago, and is now starting to deliver tangible results in delivered products. In this process IBM architects have learnt valuable lessons in the area of component-based development and SOA, and we can expect these to flow through to all parts of the platform and tools.
IBM's Experience in Componentization continued . .

portfolio and crucially processes for IBM consultants and their customers. CBDI asked IBM to share their experiences and in this report we provide some exceptionally interesting feedback that has been harvested from one of the more challenging projects currently being undertaken.

Editors Note: This report is organized as a Q & A, together with CBDI commentary. IBM verbatim responses are shown in italics.

General Background
Software componentization has been in work now for well over a decade. From fine grained UI components, we have progressed through coarser grained server based components and in the process learnt that there’s more to software components than simply exposing a component based container. Many, including yours truly, found the management activities to be seriously difficult tasks. More recently the development of comprehensive interface protocols for Web Services represented a significant step along the road. For years we have been saying that the service is simply the capability view of the component, but until you have a richer, standards based mechanism to describe this, the term service remains an interesting concept without practical implementations.

One of the difficulties in discussing components and services is that the concepts can be applied in many different ways. A service or component can reasonably span fine grained to somewhat coarse grained sets of functionality. So it is very important to define the context in which a component or set of components is placed.

IBM: IBM has been involved in “course grained” software reuse for quite some time. Such course grained software reuse has focused on the reuse and repackaging of IBM software at the offering (product) level. Software componentization is about IBM’s focus on a finer grained, component-based (as oppose to product-based), software development model. Specifically, software componentization is the establishment of software development model based on the concept of software offerings that are composed from the sharing of reusable software components. Software componentization also entails taking this software development model and building management, infrastructures and governance models to enable development teams to deliver components against this model. Software componentization is part of IBM’s on demand business strategy and part of a broader scoped continuous improvement process of the IBM Software Group (SWG) development engine.

Componentization is in essence a response to the containment and management of complexity. As we start decomposing service offerings the extent and depth of the hierarchy rapidly becomes extremely complex, and there will be many readers that are already be thinking hard about their response to this challenge. Interestingly the IBM need for components directly parallels the needs of their customers.

IBM: Software componentization provides IBM with a mechanism for the progressive enablement of software platform capabilities. New offerings add additional components and capabilities to the IBM software platform that build upon the simpler components and capabilities of previous offerings. This process of “building up” capability progressively and incrementally reveals software platform complexity and avoids overwhelming our customers with the need to integrate and deploy highly complex monolithic products. The focus of our software componentization work is to achieve two primary goals:

1. **Ease-of-Use through Outside-in Design**
   Enable customers to easily, independently, and incrementally acquire and install IBM software platform capabilities through packaged offerings that provide a set of seamless software platform extensions. [Key Thoughts: Incremental & Integrated]

2. **Market Agility**
   Enable IBM and our partners to rapidly adapt to shifting market pressures by delivering packaged offerings and solutions to market through the assembly and reassembly of software platform capabilities into new and flexible configurations. [Key Thoughts: Agile & Composeable]

Structural Matters
CBDI asked IBM about their use of the term software component – what conceptual model and classification systems are they using to cover loose and tightly coupled component concepts?

IBM: We have established a basic three tier classification system for serviceable software assets: offering, assembly, and common component. An offering is an asset that is
delivered to IBM customers. An assembly is an internal asset that contains a set of other internal and or external assets (offerings, assemblies or common components). An assembly is produced when a set of assets are used together often enough that the overhead of producing and managing the assembly is outweighed by the efficiency of being able to test and manage the group as a whole. A common component is a collection of related software, information, data and other assets that provide specific capabilities. A common component is the only one of the three defined software assets that directly provides capability. This classification system, is defined by the IBM Software Platform eXtension (SPX) Component Model Reference Architecture (CMRA), and is depicted in Figure 1.

![Diagram of Component Model Reference Architecture](image)

**Figure 1: Component Model Reference Architecture**

This strict division of assets into those that deliver capability and those that assemble capability, combined with the consistent reuse of those assets, enables ease of use and market agility.

When we use the term component without qualification, we mean either an assembly or common component. The logic here being that these components are the primary building blocks for the offerings that we ship to the market.

We acknowledge that some components will be loosely coupled and some tightly coupled. Most of the components we are currently working with are the result of the decomposition of our existing software offerings and as such begin their life tightly coupled. Our intention is to establish a looser coupling between components and the offerings that ship those components over time. We see this process as evolutionary; crawl first, then walk, then run.

**CBDI:** The reference to common does not necessarily mean shareable but rather having the potential to be shared. To clarify, the IBM common component refers to the smallest unit of functionality, or the "atomic" (not decomposable) level, with the potential for being shared. CBDI has found the notion of an atomic level very useful as a basis for definition because it helps to focus upon the lowest common denominating level dependent on the context(s).

The IBM vision is that all software be compliant with the basic component model, organized into common components. Even parts that are not shared, and all offerings are decomposed into common components and assemblies, such that in time all delivered software is well formed irrespective of whether it is intended to be shared or not.

Key here is that there is no implied calling semantic. Components are designed to be elements of offerings and the binding is expected to be variable over time and may include OSGI bundles, JARs, collection of stored procedures or Web Services. The level of loose coupling depends on the type of component and product domain.

**Architectural Framework**

**CBDI:** We then asked what is the architectural context for the portfolio componentization? What form of architectural framework is in use? How does the component architecture integrate with an SOA?

**IBM:** The architectural context for IBM software componentization has evolved naturally from our SOA and on demand strategies. SOA and on demand provide the larger contextual framework in which IBM has defined an on demand Operating Environment (odOE) to structure our middleware development and establish our programming model. Figure 2 depicts the high level architectural structure of the odOE.

The odOE specifies that on demand applications provide business services based upon the capabilities provided by the software services (interfaces) of the software components used to construct those applications. As noted before, in the IBM component-based software development model, components are the providers of capability (either directly as a common component or indirectly through their aggregation into assemblies). This alignment establishes an architectural framework, where the odOE defines the high-level capability model for IBM’s middleware offerings and the SPX CMRA defines the classification system for the construction of these offerings from constituent components. In this way, each offering becomes a purposed and validated set of capabilities provided by IBM that enables the creation of business applications meeting the ease of use and market agility criteria outlined in the two goals stated above.

**Figure 3 depicts the service chain from the creation of components through to the delivery of business services.**

**CBDI:** What form of interface model/specification is used (for different classes of component)?
IBM: IBM is in the process of developing a generalization of the Web Service Definition Language (WSDL) for the specification of component service interfaces and their “wiring” or assembly. Although this work is still in the formative stages, it bears a strong resemblance to WSDL and serves a similar, although much more general and non-web service specific role.

CBDI: We have anticipated WSDL being used in a much broader manner than the current loosely coupled service domain. IBM noted that the generalized version of WSDL has specific support for non-functional characteristics, scalability, performance, e.g., a searching component with multiple methods that provide a differentiated service capability.

Versioning and Release Strategy

CBDI: What approaches are adopted in deployment, versioning and static and rolling upgrade?

IBM: Key to the successful adoption of a component-based software development model is a holistic and flexible approach to modularity, deployment, versioning, and maintenance (upgrades, fixes, etc.). IBM has addressed these issues with three strategic undertakings: a consistent definition and interpretation of version specifications, the creation of development imperatives for version tolerance and proper maintenance procedures, and the creation of a common solution for installation and deployment that respects and implements both of these.

There are several management system controls in place to govern component reuse. For example, we have for some time had a centralized development gear box called the Software Group System House that tracks the publication of milestones and delivery schedules, manages a centralized software asset management system, and oversees the software development standards, imperatives, and compliance for such things as the use of public APIs. We’ve since enhanced the System House models and tools to integrate component-based development.

IBM Solution Install is a key emerging technology for the successful packaging and installation of componentized software. Solution Install respects and implements a well-defined and consistent version specification along with...
appropriate version tolerance rules. The IBM Solution Install is discussed in more detail below in context with platform upgrades.

**CBDI:** IBM advises that there is no easy or simple answer to all versioning issues. They mentioned OSGi RFC79 which is intended to enable enhanced modularity support. However the IBM practices place responsibility on all parties, such that the consumer (solution design) needs to be tolerant and downgrade gracefully and the provider (interface design) needs to take responsibility for subsequent versions. Testing practices should also be managed appropriately; for example never assume that something doesn’t need testing; the more use a component has the more the quality accelerates; make fewer assumptions when reusing in a new context.

We then asked – how does the component architecture impact the overall product development and release strategy?

**IBM:** In order to achieve successful consumption and reuse of software components, IBM has created a formal software componentization organization that supports software component architecture and management, modified existing development organizational boundaries, and has begun to adapt business processes and development processes (e.g., Rational Unified Process or RUP) for component-based development.

The primary boundary changes are associated with the separation of offering development from component development. The primary advantage of separating offering development from component development is the evolution of “slip-stream” software development. Slip-stream development allows offerings and shared components to begin to “detach” or “uncouple” their development cycles. With components not locked into the same development cycles as offerings, offerings-to-offering dependencies decrease. This enables the ship schedule of one offering to no longer be dependent on ship schedule of another offering, thereby moving offerings into the market faster. In order to capitalize on these advantages IBM has had to create a number of new infrastructure and management systems that are optimized for a component-based development model.

These organizational and infrastructure changes are being accompanied by an assessment of related software development processes for corresponding changes. Component-based development requires the adaptation of many aspects of traditional software development processes: requirements management, support, testing/QA, problem determination, globalization, packaging, licensing, etc. For example: requirements management must allow for managing internal requirements across components based on the needs of IBM’s offerings, support processes and procedures must allow for the dispatching and tracking of support issues from offering development teams to component development teams and back again, and licensing processes must enable the creation and management of component license agreements and offering aggregation of those agreements.

**CBDI:** Essentially the slipstream approach provides an asynchronous shipping method, where inter product (program) dependencies are decoupled. They have observable productivity gains and proof points that reduction in product dependency has been achieved.

## Process Matters

### Methodology

**CBDI:** What methodology is used for component specification and design?

**IBM:** IBM is committed to model driven development and software modeling using the Unified Modeling Language (UML) in conjunction with the Rational Unified Process. Consistent with this, IBM has developed an SPX Component Modeling UML Profile that works with the IBM Rational modeling tools to create component models that conform to the SPX Component Model Reference Architecture. The models can then be easily linked downward in a traceable manner to more detailed code-level software models or upward to capability models and business process models.

**CBDI:** Is there a single approach (methodology, conceptual model) which has been adopted across the software division?

**IBM:** To a great extent, there is a single approach across Software Group to componentization; a single approach which allows for the fact that in an organization of some 18,000 employees, the single approach must leave room
for adjusting to individual or unique circumstances. Moving to a component-based development model is not just a technology statement. There are also implications to our processes, management systems, infrastructure and tools. IBM has established a state-of-the-art component-based development model that encompasses all of these aspects in order to create, share and reuse software components as part of a more effective and efficient development engine.

IBM’s primary and common process for software development is a tailored version of the Rational Unified Process called IRUP (IBM Rational Unified Process). IRUP has been specifically tailored to include process guidance for the teams that develop and consume components. IRUP is well integrated with the governance processes discussed below.

In support of these processes IBM is implementing a consistent Software Group-wide software development infrastructure and tools. Well defined processes, including the tasks, roles, organizational responsibilities and structured information that are inputs and outputs of these tasks drive well-defined infrastructure and tools.

To support this ecosystem, we have a Software Group Process Model that defines the tasks, inputs, outputs, roles, and organizational responsibilities, a Software Group information model that defines the data structures of those inputs and outputs and how they are implemented in our infrastructure and tools.

IBM’s approach to componentization starts with a well-defined, but continually evolving, conceptual model defined by the Component Model Reference Architecture (CMRA). This reference architecture is created under the auspices of the Software Group Architecture Board by a senior, cross-divisional team. The component development teams (providers of components) as well as offering development teams (consumers of components) build component models that adhere to the CMRA. They are assisted in this task by tools which extend UML for this purpose, as described above. The component models are the means of capturing and socializing dependencies between and among the various serviceable software assets (offerings, assemblies and common components). The collection of all these individual, linked component models is called the Software Group SPX Component Model. Depending upon the success of this work, IBM is considering the future possibility of taking the CMRA to an appropriate standards body.

**CBDI:** IBM indicates IRUP is a customized version of RUP which adds new states, roles and responsibilities consistent with the component model, supporting the (strictly separated) provider/consumer development approach. We understand there are no specific plans to incorporate this into the mainstream RUP product yet.

**QA and Test**

**CBDI:** What are the QA and testing methods?

IBM has defined the various phases of testing as well as when and how they are to be performed and includes substantial integration with IBM’s software automation strategy and framework. We have created a Software Group-wide team to address critical issues around the QA and testing of component-based development of software. That team is focusing primarily on how a component is tested, the testing of an aggregation of commonly used components, and the productivity increase because of the reuse of the tested component or aggregation of components. The processes defined by this team also address the testing that must be carried out by consumers of components to assure that components work properly within the offerings that use them. By clearly defining the types of testing that must be done for each type of serviceable software asset, an increase in productivity can be gained by eliminating the need for redundant testing.

For example, the IBM Test team (a part of the over-all QA and testing effort) has defined the specific types of testing required for each type of serviceable software asset based on our classification system for such assets. A common component is required to undergo a specific series of tests before being certified as ready for more general consumption. An assembly, a group of serviceable assets that are aggregated together, undergoes additional tests to ensure the grouping works together. Consumers of a common component or an assembly gain an increase in productivity in both development and reduction of further testing by knowing that these assets have already been certified to a specific level of quality.

IBM has a corporate-wide quality management system for its products that is certified to meet the standards of ISO 9001–2000. This quality management system specifically addresses quality and the development of components across all disciplines in the software development lifecycle. A Software-group wide executive management team assures that component development teams adhere to the requirements specified by this quality management system.

**Asset Management**

**CBDI:** How is asset management carried out? How are supplier and consumer coordinated?

Key to the success of component-based development is the creation of well-tuned supporting infrastructure. IBM’s componentization infrastructure currently incorporates: an asset management system that provides a detailed asset catalog (based on the Reusable Asset Specification – RAS – developed by IBM Rational), a common build repository, a clearing house system for managing offering...
and component plans and controls, and a master UML-based component model repository. Each repository stores information in its own native form. We have created the Software Group Information Model (SIM) to provide a consistent core set of component attributes. SIM ties the repositories together so that information in each is consistent, and can be related and combined. Using SIM and a set of EMF models and SDO mediators it is possible to view the various infrastructure repositories as single large partitioned data set with SIM as the “foreign key set” that is common across all repositories. The Software Group component-based development infrastructure is depicted in Figure 4.

**Platform and Tools**

**CBDI:** What tools have been developed to support the component strategy?

**IBM:** The use of the SPX Component Modeling UML Profile, mentioned above, in conjunction with the IBM Rational modeling tools are the primary tools supporting our component strategy. These tools are enabling IBM to create a unified software component model representing the greater portion of IBM’s middleware software assets. This model repository is being linked to the other key software infrastructure tools and repositories such as: software milestone tracking and clearing, software solution install and packaging, software asset management, software support, software build and test, and software plans and schedules. A great deal of effort has gone into enhancing IBM’s software infrastructure and tools to enable reporting upon dependencies (nested and direct) and reuse for the teams to better understand their consumers.

IBM is currently evaluating plans for incorporating our component-based software development processes and tools into our IBM Rational product line.

**CBDI:** Have any changes been necessary to the platform to facilitate component deployment/upgrade?

**IBM:** The most notable change has been the need to shift some of our key technologies bases to more flexible and modular foundation technologies. This can be seen in the recent movement of IBM’s software tooling, client,
and application server strategies to an Eclipse/OSGi foundation.

Additionally, widespread deployment of shared components and management of their lifecycles requires a significant jump in the sophistication of deployment technology, especially as it involves dependency management. The IBM Solution Install architecture, developed as part of IBM’s Autonomic Computing effort and recently submitted and accepted by W3C, addresses these needs. This state-of-the-art software installation architecture, and the infrastructure that implements it, provide the ability to do advanced dependency management that uses knowledge of relationships between installed software and expressions of backward compatibility to make complex decisions about the lifecycle management of shared components. Future releases of IBM’s Tivoli software deployment and Rational software development tools are slated to support the IBM Solution Install architecture.

**Governance**

What governance methodology is in use? Are there metrics that illustrate the benefits of the approach?

**IBM:** It was recognized early on that the our existing governance model for offering (product) development needed some changes to support componentization, primarily because shared components are not being developed for specific marketplace segments, but are in fact, being developed to address the needs of many offerings across IBM. Offerings, in turn, are developed using shared components, to address specific market segment needs. To the extent possible, shared components need to be marketplace neutral. Another way to make this same point is to say that the marketplace for shared components is IBM itself.

To accommodate and enable these changes, Software Group defined and instituted a management team specifically responsible for the management of component development for the “internal market” of Software Group. In many ways the approach leverages the best practices of successful open source efforts. IBM’s internal, component governance model is similar to the Eclipse management organization: it brings together the organization-wide development leaders to establish the roadmap, build the platform and deliver the vision. Similarly, the IBM internal version of the Eclipse requirements council meets fortnightly to evaluate our componentization projects in detail. There are a number of metrics used to evaluate the effectiveness and efficiency of componentization. A few of the most important are:

- Average number of consumers (consumers are IBM offerings) for a shared component.
- Percent of function contributed by “outside committers”. (IBM has implemented a community-based development environment with IBM modeled after Open Source development. In this way, consumers of shared components can contribute to the development of the shared components they consume).
- The increase in the number of offerings with out offering-level dependencies.

In addition to these metrics we are in the final stages of developing component-specific metrics to assess the success of individual components.

The general benefits of componentization have been addressed throughout the responses to the questions above. Some of the specific benefits of component-based development that we have already recognized are detailed in Table 1.

**Concluding Thoughts**

**CBDI:** Is the IBM experience directly applicable to enterprise domain architecture, design and deployment? How might it differ?

**IBM:** It is IBM’s belief that our experience is indeed applicable to enterprise domain architecture, design and deployment. We have reviewed our componentization strategy and approach with several IBM customers and business partners who concur that there are many similarities. They have clearly indicated that they would be interested in seeing our componentization technologies and tools made available as IBM offerings for their use and so that they can participate in the creation of componentized software based on IBM technology.

Software Group is a middleware development organization while a good portion of enterprise architecture, design, and deployment is about solution development. However, this is not universally true, since obviously IBM’s services organizations use our middleware offerings to create customer solutions and many of IBM’s customers are IT organizations that are creating reusable semi-finished offerings built around IBM middleware for organizations in their enterprise.

Nonetheless, we firmly believe that the software development industry today has reached a crossroads and IBM intends to help lead the way through this change. Software components and their capabilities are the key technologies for addressing the increasing rate of change and need for speed, flexibility and adaptability demanded of in today’s software development organizations. This major undertaking is starting to deliver tangible results and, as CBDI has noted, we have learned valuable lessons in component-based development and SOA. We expect
<table>
<thead>
<tr>
<th>Introduce consistent behavior and increased simplicity through common components</th>
<th>Autonomic computing components (common install, common console, common infrastructure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduce Redundant Development &amp; Redundant Processes</td>
<td>Reuse of the best function rather than reinvention of similar (but different) function</td>
</tr>
<tr>
<td>Reduced maintenance &amp; service costs</td>
<td>Common parts are driven to higher quality sooner</td>
</tr>
<tr>
<td>Accelerate Higher Quality</td>
<td>Reuse drives higher initial quality of new marketplace offerings</td>
</tr>
<tr>
<td>Deliver to Market Faster</td>
<td>Leveraging inventory of shared components will reduce time to market</td>
</tr>
<tr>
<td>Rapidly Adjust to Changing Marketplace</td>
<td>Ability to compose solutions more directly targeted at specific or unique market opportunities. Allow new offerings to be built quicker by leveraging existing inventory. Enable an ecosystem</td>
</tr>
<tr>
<td>Reduce Redundant Development &amp; Redundant Processes</td>
<td>Reuse of Common Criteria certifications reduced efforts and cost savings for consumers.</td>
</tr>
<tr>
<td>Eclipse is at the core of Rational/WebSphere tools, IBM Workplace and DB2 Tool Strategy</td>
<td></td>
</tr>
<tr>
<td>Common architecture will enable better problem resolution. For example problem determination (PD) gets easier through the use of a consistent PD approach</td>
<td></td>
</tr>
<tr>
<td>Reuse of a WAS self-certification test suite for Operating Systems changes reduced test cycle from 20PM to 9PM saving $183K per platform for three-plus platforms.</td>
<td></td>
</tr>
<tr>
<td>Reuse of common tooling and components across WAS offerings reduced delivery cycle by 18 months.</td>
<td></td>
</tr>
<tr>
<td>Reuse of open standard components (e.g., Eclipse, OSGi Runtime, and BPEL/JSR-170) provides a solid and flexible industry standard foundation for growth across IBM offerings and with our partners.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1:** Specific Benefits of Component Based Development

these lessons to flow through to all parts of IBM’s platform and tools portfolio as well as crucial processes for IBM consultants and our customers.

**CBDI:** We were also interested to hear IBM’s views on open source and how they use it themselves. In the tools development area they use open source extensively. However in other areas such as large scale, run time environments they find it less effective. This is directly related to the constituency—there are many developers that are extremely knowledgeable about tools and can contribute to a significant extent. Whereas it is much harder for the average developer to get experience of a more specialized environment.

IBM has indicated that this is a major transformation for their organization, and it will take a considerable time and effort. They have wisely commenced with the necessary component reference model that gives them the firm foundation on which they can build. However when the discussion turns to automation, it becomes very clear that this is the real enabler, not just because this is a tools company, but because it’s absolutely essential to manage the vast number of moving parts and process steps.

Over the years we have seen many comparisons between software components and other engineering disciplines, and they generally approach the subject from the structural dimension. However we think that IBM’s experience suggests that the really useful analogy is with large scale production engineering, for example the automotive or aerospace environments, where the primary task is a product line development activity in which managing tens of thousands of common components and assemblies through multiple concurrent versions requires an extremely
high level of discipline together with the highest level of automation.

Acknowledgements
Thank you to IBM and in particular:
Ciarán DellaFera, IBM Distinguished Engineer, Chief Architect SWG Componentization (ciaran_dellafera@us.ibm.com)
Patrick Gibney, Director SWG Componentization (pgibney@us.ibm.com)
Jeff Reser, Manager SWG Componentization (reserj@us.ibm.com)
Carl Kessler, SWG VP WW Development & Quality (csk@us.ibm.com)
Practical Service Specification and Design

Part 2a: Refining Service Requirements for Consumers

How do we decide the functionality of a software service? The Service Portfolio Plan has identified the services required, and defined their broad responsibilities. Next, we need to determine the specific requirements for each service: the actual functions (operations) it needs to offer, and any non-standard quality requirements. This report introduces techniques for formulating these requirements, which are initially derived from the immediate needs of the software solution projects that are underway. What gets built (or acquired from external sources) has to serve the needs of these projects. At the same time, the Service Provisioners must aim to address the needs of future Service Consumers and must deliver the flexibility needed to cope with the inevitable changes to the prevailing business environment. If necessary, each service’s functionality is delivered incrementally; the service content is then extended in future releases.

Introduction

This is the second article in a five part series that provides guidance on how to develop software services for long-term value while addressing the immediate needs of current projects.

- Part 1 covered Service Planning—identifying the software services an enterprise should acquire in the long-term.
- Part 2 covers Service Refinement—in which the specific functional requirements for services are uncovered:
  - Part 2a describes techniques for driving the service details from underway Software Solution projects.
  - Part 2b will consider situations in which the service providers take the lead in defining requirements.
  - Part 3 will address Service Specification—preparing a thorough definition of service function and quality, to be used by implementers, testers, consumers and operations staff.
  - Part 4 will concern Service Delivery—the activities required to take a service from specification to deployment.
**Provisioning Services**

Our practical approach to service design advocates that a service is provisioned only when a Software Solution Delivery (SSD) project is underway and has a definite need for it. This statement, however, needs qualifying in several ways:

- The term provision is intended to embrace all the ways a service can be sourced. For example: Hosted services used on a commercial basis from external providers, external services provided by trusted partners, internal services with the automation unit developed in-house from scratch or services built by constructing wrappers around existing in-house systems.

- The SSD project should originate from a desire to improve a business process. A Business Process Improvement (BPI) project designs a new process, to make that process faster, better and cheaper, and to improve compliance with regulations. This then spawns one or more SSD projects.

- SSD projects are obliged to use existing services as far as possible; not every service-consuming Solution will require further operations to be provisioned. As the BSB and the rest of the service portfolio expands, this should become an increasingly common situation.

- Some companies will prefer to predict the service requirements of their future projects and seek to acquire services in readiness of those needs, and this is certainly a legitimate tactic. But it is risky too, since the future projects may never take place, and significant business changes may occur between acquisition and use. Past experience indicates that provisioning IT functionality well ahead of usage can prove to be expensive, wasted effort.

- This article focuses on gathering the Service Consumers’ requirements for services. The next article will cover situations where it is appropriate for service providers to formulate the detailed service requirements.

This article will recommend a group of techniques that can be used to identify and detail the operations, and hence refine the definition of the services first proposed within the Service Portfolio Plan. UML diagrams and terminology will be used, so the techniques should already be familiar to many readers, though there is now a service-oriented twist to them.

Before introducing these techniques however, it is useful to learn something about the context in which we expect them to be used.

**Project Context for Service Refinement**

Figure 1 presents the project context for service refinement, in the form of a UML activity diagram. The three vertical “swim lanes” represent distinct projects, performed by separate teams.

- On the left, the Business Process Improvement project is focused on making a business process run better or faster or cheaper. The process in question could extend across organization boundaries, and involve cooperation between several companies. The redesigned business process typically requires one or more Software Solution Delivery projects to be organized—to amend existing systems, or to create new applications. This can be done more rapidly once your organization has built up the content of its Software Service Portfolio. The BPI project is also responsible for the progressive rollout of the redesigned process to the business workers.

- The central swim lane, Software Solution Delivery, is responsible for building the software to support the redesigned business process, but not the software services—apart from those that are part of the internal Solution Software design, or those exposed by the solution itself as will be discussed later. The designers prepare a description of each service they require, including all the operations that are needed. These descriptions should comply with the policies set out by the planners. Using a registry of published services that already exist, they then aim to identify suitable pre-existing services for use within the solution. In some circumstances, Service Planning will have already identified core Services that pertain to the business domain that must be used by the solution project. To use these services, it may be necessary to make adjustments to the proposed Solution Design. The requirements for any services not found in the registry (and are not internal to the Solution Design) are passed to the Service Provisioning team.

- The right hand swim lane is used to show the two supply-side projects that the Solution Designers interact with. We will expand on these in Part 2b.

The refinement techniques introduced below reflect a business process-led approach to service-orientation. Services are initially provisioned when they are needed by a Solution that improves a specific business process. As more services are provided, it should become quicker to introduce new business processes, and easier to improve the quality of existing business processes, so long as the services have been built to be independent from specific business processes and user interface designs. We will
consider these factors again when looking at the supply-side requirements in Part 2b.

Some writers have suggested that process flow diagrams are an inappropriate way to view and redesign a service-centric business (even though services can be represented in such diagrams), since flow diagrams encourage businesses to be viewed as consisting of chains of activities, rather than being viewed as organizations which prosper by using and consuming physical services. New techniques may emerge that are more suited to the latter view. In the meantime, our practical approach adopts some well-established modeling techniques and places the emphasis on evolving a Software Services Portfolio and service-centric systems, rather than promoting the notion that the business itself should be structured in a service-oriented fashion.

The Techniques

The following sections introduce the techniques that are used to identify service operations and refine our understanding of service requirements. *Italics are used to indicate where the technique is applied within the project context.*

![Diagram](image)

Figure 1: Context for Refining Service Requirements, expressed as an Activity Diagram

Process Flow Diagram

*Used in Design Improved Business Process*

This diagramming technique is used to explore and document both as-is and to-be models of a business process. Each sub-process of the business process is allocated to a “swim lane” on the diagram. A swim lane can represent an external organization or customer, or an internal department of your own company. The example given in Figure 2 is drawn using the UML’s Activity Diagram notation. The BPMN1 provides an alternative notation. As business processes are increasingly defined to cross enterprise boundaries, we can expect these diagrams to depict the organizational units within several enterprises, and both BPMN and UML2 provide notations to support this.

The flow lines in these activity diagrams are effectively object flows. But rather than depicting object boxes or UML2 “pins”, which would make the diagram rather cluttered, the lines are simply named according to the principal object that flows, typically qualified by the object’s state. The square brackets signify the guard conditions for decision nodes.

Process Flow Diagrams are not special to service-orientation, but we shall see how they lead to the discovery of a service’s operations.

1See [http://www.bpmn.org/](http://www.bpmn.org/) BPMN refers to the sub-processes as “activities” while UML calls them “actions”. UML2 Activity Diagram notation is much richer than that defined in UML1.
Observe that this example does not refer to computer systems: the sub-processes shown may or may not get IT support. The business analysts should aim to decompose the business process until the sub-processes on the diagram are all elementary.2

This diagram can also be used to consider and depict physical services3 used by the company.

- A physical service used within the process can be depicted as a sub-process adorned with an appropriate stereotype. It may be helpful to create a separate swim lane representing the service, where each sub-process box shown in the swim lane represents a different function provided by the service.

- The internals of a physical service (especially one supplied by our company) can be depicted in a separate process flow diagram.

- The process flow diagram can be studied to identify potential services, to be bought in, or offered for sale.

A more advanced treatment of business modeling for service-orientation will be found in CBDI Journal articles written by Richard Veryard.

---

2 An elementary process is one that should not be decomposed into smaller processes. It should have a triggering business event, and not require further business events to occur in order to complete a process execution. Generally, it’s a task that one person can do in one location, continuously until done. Management assigns resources and measures performance at elementary process level or higher, not at sub-elementary process level.

3 The term physical service is used here so the reader is clear we are not referring to software services. A physical service could make use of IT, but in that case human intervention is usually needed to initiate an IT capability or to accept a device’s outputs.
Figure 4 also includes some further possibilities for service usage. This diagram proposes that the Conference Service of the BSB is offered to customer organizations, so they can build their own “Event and Travel Booking” applications (say), which allow their employees to book places on any conferences run by trusted suppliers. This is documented by the Directly Register for Conference use case in the Customers’ Systems swim lane, which calls the createBooking operation of the Conference Service hosted by the Conference Company.

We also see that the Conference Company calls an Employee Diary service exposed by the Customer, which enables the Conference Company to provide the booking confirmation by directly updating the online diary of the
booking employee. This illustrates that in many processes, service requests may occur in both directions between participants, regardless of the actual flow of the business activity or transaction.

Software services bring new possibilities for business-to-business interaction. As well as delivering the software solution as a portal for use by employees, or for customer self-service, the Solution could also expose services for direct consumption by customers and business partners to further optimize the end-to-end process through automation. The business analysts need to evaluate these options while designing the business process improvements. These process flow diagrams can only reflect the broad service operation requirements however, and additional techniques, as described below, will be used to drive out a more thorough set of service requirements.

**Use Cases from Process Flow Diagrams**

*Used in Design Software Solution*

Once the process redesign has been approved, the software requirements of the Business Process Improvement project are organized into one or more Software Solution Design projects.

A use case diagram of requirements is prepared for each SSD project, showing the system use cases that were identified in the process flow diagrams. The initiating use case actor is probably a role within the organization unit (swim lane) which is responsible for the use case. This role can be generalized to reflect other potential use case initiators. But some use cases will be initiated by operations staff (especially batch jobs) or by “the system” (where automated triggering without human intervention can be achieved).

It is useful to provide use case adornments to indicate whether the use case is new, changed or existing-unchanged. Existing-unchanged use cases can be omitted altogether unless they are judged helpful for intra-project communication. Our simple example appears in figure 5. Note that we have also chosen to include the Create Booking operation of the Conference Service as a use case, since it will be possible for Customers’ Systems to activate it.

**Use Cases Refinement Maps**

*Used in Design Software Solution*

We suggest that each new and each changed use case can be subdivided into one or more logical steps, and the service operations called by each step identified, using...
use case diagram notation. An example of the resulting “refinement map” is shown in figure 6.

This technique helps to drive out potential operation requirements per use case, but is not as thorough as the interaction diagramming technique that follows. On the other hand we investigate the service requirements without regard to the user interface and dialog solution, so it offers an opportunity to seek out the logically-essential, key operations, which could fit alternative end-user solutions.

Nonfunctional Requirements
Used in Design Software Solution
The nonfunctional requirements should be defined for the Software Solution as a whole, but an individual use case may have differing nonfunctional requirements. Either way, these requirements must then feed through to the quality of service (QoS) demanded from the software services consumed by this use case. The service specification can then include any overrides to the default QoS required for services (defined as a part of the Software Services Portfolio⁴), with the consuming use cases providing inputs to this specification activity.

Generalizing Service Requirements
Used in Specify and Provision Service
The techniques described above drive out service and operation requirements which are specific to one Software Solution. The service provisioners will need to generalize the requirements summarized in the interaction diagrams and nonfunctional requirements, so that the provided services will have wider applicability and longer life than that demanded by the requesting Solution. Generalization techniques will be covered in the Part 2 article.

What Comes Next?
This Part 2a report has primarily considered service requirements from a consumer perspective. Part 2b will discuss requirements from the provider perspective, and these tend to be driven by the need to ensure domain

⁴See Practical Service Specification and Design Part 1: Planning the Services.
consistency and provide reuse across the enterprise, by external pressures such as compliance with industry standards, by the needs of business partners, and by existing services offered by packaged applications.

Either way, having described the services that are needed, and the operations that currently need to be supported, a rigorous service specification should be prepared. This can be used to instruct implementers, or to evaluate services offered by third parties. It continues to be useful after service acquisition, since designers and developers need a definitive definition of service functionality and quality to work from. This will be covered in Part 3.

Acknowledgements
This article includes ideas originated by Paul Allen of Computer Associates and Lawrence Wilkes of the CBDI Forum.
Product Report

Polarlake Integration Suite

CBDI reported on PolarLake when it was first launched and was impressed by its capabilities. Since then it has matured into a more complete product and is now positioned as an Enterprise Service Bus (ESB). In this report we look at the new features and assess PolarLake’s ability to act as a bridge between new Business Service requirements and the diverse range of resources in use in most organizations.

Introduction

We first looked at PolarLake at the beginning of 2003. At the time we concluded it was an extremely polished and well produced product that stood out from the then current clutch of Web Services software products in the way it was focused on the real-world customer requirements of XML and non-XML content processing rather than specializing just on the Web Services paradigm that was emerging at the time.

Since then, the PolarLake Integration Suite has expanded to include significant new functionality that positions it in the Enterprise Service Bus (ESB) product marketplace. Wisely in our opinion, PolarLake does not have its own Message Oriented Middleware (MOM) sub-system, relying instead on interoperability via open standards such as JMS or directly with MOM products such as IBM MQSeries, Tibco and others. Instead it focuses on core ESB functions of Service orchestration, and mediation (message broking and transformation), whilst leaving transport to the underlying infrastructure that most organizations will already have in place.

PolarLake has a similar approach to Service Management (SM). Whilst it includes a useful Business Activity Monitor (more later) and can be configured to generate relevant SM information, it does not have an SM dashboard or specific SM functionality, preferring to delegate that role to Systems Management products such as HP Openview via SNMP.

Enabling Business Services

A key principle of SOA is that to improve agility Business Services should be loosely coupled with the implementing resources. In addition to the loosely coupled transport and contract provided by the Web Services layer, it is highly desirable that the behavior of the Service and its messages should be further abstracted to hide the implementation specific detail that is often exposed via the APIs of existing resources.

Business Services should reflect a meaningful business activity, and to enable this the published Service will often need to consume and aggregate other Services to deliver the functionality required by the Service Consumer.

Messages may have to be routed to different resources based on their content and business rules – for example sales orders may have to be routed and transformed to various product specific systems that have been built or acquired as silos within a large organization over the decades. This requires a Service transformation layer as illustrated in Figure 1 that can expose the Business Service, process the messages, and route them to the various back-end resources regardless of their APIs, or in turn request other Web Services. Even where the existing resource supports Web Services it is likely that the message will require some level of transformation between incompatible schemas, or aggregation and orchestration of multiple Services.

There are a number of ways in which organizations might implement this layer. They could build process components or Service Facades\(^3\) over the existing systems, but typically these will be specific to the particular implementation.

There is also the additional requirement of Web Service enablement of existing APIs. This might be done natively by the App Server, or require a façade or adaptor.

A more cohesive approach may be appropriate using an integrated product, that supports transformation, orchestration and ESB capabilities such as PolarLake.

**XML Circuits**

Central to PolarLake is the concept of XML Circuits. To recap on our original report, a circuit enables the processing of an XML document to be modeled as a pipeline of discrete steps assembled from a rich palette of components using a graphical, declarative approach that requires no additional coding which should improve responsiveness to change (though XSL, BeanShell or XQuery scripts can also be included).

**Data Circuits**

This original document processing functionality is now referred to as a Data Circuit. The component library has expanded to some 70 standard operations which as well as providing various ways to process and manipulate the documents also includes:

- Event scheduling
- Initiation of Service requests – e.g. use content from a document to request a Service to return further information that is used to validate, or enrich the document.
- Monitor databases and queues
- Milestones for Business Activity Monitoring (see later)

**Process Circuit**

The new functionality added since our last report is the addition of BPEL-compliant Process Circuits as shown in Figure 2. This visual specification approach provides integrated support for both orchestration of Web Services and PolarLake Data Circuits. This is rather useful as the Data Circuits can access non-XML documents, and invoke other “services” such as reading a message queue, permitting them to be orchestrated within the BPEL process. Whilst BPEL provides the overall standards-based orchestration and correlation capability, Data Circuits provides for a much greater range of processing options.

This is a very powerful combination as multiple Process and Data Circuits can be used together in a “circuit hierarchy” to deliver quite complex behaviour, as well as facilitating reuse of individual Circuits.

---
In the simple “Zip Code Aggregator” example I constructed in Figure 2, the process invokes two Web Services in parallel (in this case provided by StrikeIron) to return different types of information about a US city based on a zip code. The Process Circuit then invokes a third service – an XML Data Circuit, which joins together both the documents returned by the initial Zip Code Web Services.

Once you are familiar with PolarLake and understand the behaviour of the Web Services and the process being orchestrated, an activity such as this would take just minutes to construct and deploy.

It is also worth noting that PolarLake is also compliant with WS-Addressing which enables the dynamic routing of messages.

Adaptors

Some ESB products have a heritage in Enterprise Application Integration (EAI) space which typically use a common engine into which various adaptors can be plugged in for each different application that needs to be integrated.

PolarLake have not attempted to reinvent this wheel and instead partner with Netmanage and license their Librados adaptors. These provide access to packaged applications such as Oracle E-Business, PeopleSoft, JDEdwards, SAP R/3, etc.

PolarLake also includes inbuilt adapters to manage integration with:

- Application containers such as IBM WebSphere, IMS and CICS (through partnership with GT Software), WebLogic, JBoss
- Microsoft COM and .NET, and J2EE
- Databases including Oracle, SQL Server, DB2 (Unix, AS/400), MySQL, HSQLDB
- MOM Connectors including openjms, IBM MQ-series, Tibco, Spirit Wave, SonicMQ
- JMS Topic Connectors including openjms, IBM MQ-series, Tibco, Spirit Wave, SonicMQ
- FTP, HTTP, SMTP, file system

Given that customers may already have an investment in an EAI product, nothing prevents PolarLake from also working alongside these by treating their messages as another data source in a XML Data Circuit.

Mapper

Though the use of an additional transformation and mediation layer in the SOA and the use of XML can greatly improve flexibility, technologists are nevertheless often reluctant to use such approaches if they are not performant. As we noted in our original report, PolarLake address this issue through a highly optimized architecture that streams and multithreads XML documents through its processing pipeline, which also ensures a document is only scanned once.

In addition, whilst the transformation of XML document can be done with XSLT, PolarLake also provides its own Mapper that deals more effectively with complexity and provides a richer set of conversion options.

Business Activity Monitor

There is further new functionality in the form of a Business Activity Monitor (BAM). Multiple milestones components can be placed in a XML Circuit to track business activity, for example recording the progress of a document through a long-running process. This can be monitored with

Figure 2: PolarLake Process Circuit

PolarLake Integration Suite continued . . .

PolarLake’s own lightweight BAM browser application or a third party Business Intelligence (BI) product. As well as business activities, this could also be used to monitor SLA and other non-functional aspects in much the same way as many Web Service Management (WSM) products do.

PolarLake as a Business Service Enabler
To summarize these features, PolarLake can be seen as operating on three levels

- The combination of BPEL-compliant XML Process Circuits and XML Data Circuits provides a productive capability for the orchestration and mediation of Business Services, and their aggregation and abstraction from other underlying Services.
- The XML Data Circuits and EAI adaptors can provide a Service Enablement layer as shown in Figure 1 to coordinate and integrate a wide variety of existing resources.
- Straightforward CML document processing.
- PolarLake has published case studies that document the experience of many organizations using the PolarLake product for this purpose and more traditional EAI tasks, or even to build client/server style applications, regardless of any requirement for SOA or the use of Web Services.

The capability of PolarLake to combine Web Services together with other data sources allows Service Orchestration and Service Enablement to be combined in a single process. Though the XML Circuit architecture provides for a high degree of flexibility and reuse for a large scale solution it does require planning as to how it will be decomposed into the individual fine grained steps. Whilst PolarLake may not require low level coding, it nevertheless still requires a good understanding of the design activity. As with any programming environment there can be more than one way to achieve the same outcome, some more effective than others.

Case Studies
Partly reflecting a core customer base in the financial sector, PolarLake have expanded from Dublin into offices in London, New York and Tokyo. They also have new channel partnerships with Accenture, Hitachi Japan and Sun. A brief look at some of their recent case studies gives a good indication of how PolarLake is being used and the benefits customers are seeing.

Criminal Justice IT (CJIT)
CJIT is responsible for providing support for the UK Criminal Justice System (CJS). They wanted to improve the way information from multiple participants such as the police, the Crown Prosecution and the courts, is provided to victims and witnesses, so they better understood the status of a case and their own obligations – such as the requirement for them to appear in court. PolarLake is used to create a message exchange that enables case information to be shared between these agencies and delivers information to a portal that provides single point of contact for the victims and witnesses.

The message exchange uses XML and Web Services to provide a loosely coupled connection removing the hard coding of previous solutions, which were inflexible and difficult to maintain. This will also make it easier to plug additional agencies into the exchange in future. The use of XML Process Circuits has also reduced the effort to implement workflows across the multiple agency systems and adapt them to changing needs. The approach of retrieving the information “on demand” rather than replicating it into a data warehouse to support the portal was also important from a data protection perspective.

KDDI
KDDI, a major Japanese telco, wanted to offer services to its customers such as hotel booking or order foreign exchange, to be accessed either by voice or handset. The original solution was based on iMode, the Japanese equivalent of WAP. However they found this expensive to run, and because of the way it worked, it required all the information to be replicated and stored in the iMode servers. In addition, it was inflexible – preventing rapid change of service providers, to respond to changes in competitiveness or improved service levels from the different service providers in the market.

To reduce these operational costs for KDDI, the solution uses Web Services to retrieve the information “on demand” from Service Providers, whilst a UDDI registry is used so that new Service Providers can be easily included. PolarLake is used as a broker to route requests to the appropriate provider and to transform the request to their particular schema.
Ricoh

As a result of mergers and acquisitions, Ricoh, a print engine manufacturer, found itself in the classic situation of running and maintaining multiple order, production logistics and sales management systems that were aligned with different products. Each sales order line was manually rekeyed into the appropriate system.

Ricoh decided to rationalize the back-end logistics, production and management systems, while retaining the sales systems so there was minimal impact on customers and the channel. PolarLake's role was to act as a mediation layer accepting sales orders in multiple formats such as XML, Excel and CSV plus transports including MQ Series, FTP and SMTP, and transforming them ready for the new back-end.

As well reducing the errors and inefficiencies in the manual processing this provides a more agile solution for new product lines and further M&A activity. Ricoh also say it has reduced development and maintenance costs by 66%.

Nissay Dowa

Nissay Dowa, a major insurance company serving the consumer market in Japan, had multiple product lines that it wanted to present via a self-service portal to save call centre costs, whilst also introducing a CRM system to provide a single view of the customer. In this case PolarLake is used to provide a flexible solution for the future. The ability to deploy the Polarlake runtime across a distributed system has enabled them to ensure the solution is highly performant.

Nissay Dowa have seen significant cost reductions associated with future changes to the Portal screens and the costs associated with changes in the application itself.

Summary

With these new capabilities, PolarLake provides a integrated capability for delivery and orchestration of Business Services. The XML Circuits concept provides for high levels of flexibility and reuse. PolarLake Integration Suite also provides a critical bridge between Web Services and non-service based data sources, making it effective as a transitional tool on the SOA Roadmap.

ESB is rapidly becoming an overcrowded marketplace. Peer referral is therefore vital and hence common benefits cited by PolarLake customers are important, such as:

- Implementing SOA with PolarLake is delivering business flexibility
- The declarative approach and ease of use is providing development productivity
- The deployed systems are highly performant

PolarLake occupies a slightly different space to many other ESB products on the market. While many other ESB products focus on comprehensive support for messaging and other transformational capabilities, PolarLake has stronger support to the integrated process of specifying and managing the business document and process, together with adequate service bus capabilities. In many situations this will be a strength as organizations will already have a messaging infrastructure in place and technology support for Web Services will be delivered more effectively as part of the existing middleware platform.

Organizations assembling their SOA toolkit should consider PolarLake Integration Suite as a best of breed component that helps them build solutions rather than accumulating yet more middleware.

Links

PolarLake – http://www.polarlake.com
Improving SOA Governance with the Systinet Business Registry

The UDDI Discovery Standard was one of the original trio of Web Service protocols along with SOAP and WSDL. Now in Version 3, the UDDI specification might be expected to be relatively mature, however, there are still plenty of opportunities for a software vendor to integrate directory functionality into a more complete product that supports the requirements for an organization’s own Service Registry. In this report on the Systinet Business Services Registry we look particularly at how SOA governance can be improved through the provision of a set of lifecycle applications and services.

Introduction

The UDDI (Universal, Description, Discovery and Integration) data model and protocol is one of the three original core Web Service specifications. It defines the implementation of a standards-based registry by which Service Providers can publish their Services and Consumers can discover them. Because UDDI APIs are based on Web Services, it is straightforward to integrate the registry into tools or portals. Today UDDI based registries and directories are widely used in private, in-house environments. The original vision of UDDI envisaged more dynamic business systems that could autonomically discover and consume new Services, and while little use is made of this capability today, we fully expect that as SOA becomes the de facto approach to systems deployment, dynamic discovery services will become increasingly important.

However, before UDDI is used in any revolutionary fashion in the future it must first overcome some challenges of today. These are largely issues of governance. For example,

- Quality of information. The publicly hosted UDDI registry – the UDDI Business Registry (UBR) – jointly operated by IBM, Microsoft, NTT Communications and SAP are exactly that – public. They are also unregulated, and as such there is little guarantee that any of the entries are valid, and a large amount of the information is out of date or contains broken links.

- Privacy and security. Many businesses have no desire to make their Services public. In addition, many have no real need at present to “discover” Services, as the Provider and their Services are already known to them via other routes that are part of existing commercial
arrangements. Hence they don’t need to make the use of the public UDDI Business Registry.

- Lifecycle. There's no real lifecycle support built into the UDDI specification to manage the submission, approval and publication process.

- Non-functional requirements. Discovering Services by functionality leaves many questions unanswered, such as Quality of Service (QoS) or the commercial basis on which the Service is offered.

One way round this is to use an intermediary who hosts their own UDDI-based registry and exercises much greater control over the contents, or who front-ends the public version and filters the information more rigorously, such as BindingPoint® or Sal Central®. Sometimes this is facilitated on a commercial basis between the Service provider and the intermediary who effectively acts as a commercial publisher.

Another alternative that is increasingly common is for organizations to implement their own private registry. This could be used for:

- The internal publication and discovery of Internal Services within the organization
- Internal publication of a filtered set of approved External Services
- With the right security in place, the sharing of the private UDDI registry across an ecosystem of business partners

Many find that the ad hoc Service Provision and Consumption within their organization leads to similar problems of governance. In a recent CBDI survey® most respondents admitted they had little visibility of service-based activity across organizations. A registry by itself does not overcome this problem, however it can provide a focal point via which organizations can exercise greater governance over their SOA. To improve the overall scenario it would be useful to complement a UDDI-based registry with:

- Service Management tools from Web Service Management (WSM) or traditional Systems Management vendors that can sniff out Web Service activity on the network, so that their usage can start to be governed
- Service Management tools could also provide feedback to a UDDI registry on the historical service level of production Services.

- Asset Management tools. The visibility of internal assets across an enterprise is often weak, and hence compromises the ability of developers to discover and reuse those assets. Having a Service registry that could be linked into their tools and process, and categorized in ways that are meaningful to them provides a central controlling or coordination point. Linking a UDDI-based registry to asset management tools would provide a broader solution for the organization’s overall asset management needs

- Integration with security and identity solutions, such as LDAP to enable permission based access and usage.

Systinet Business Services Registry

UDDI is a specification and whilst standards compliance drives a base of common functionality this still allows software vendors to provide their own unique implementation. Systinet is a vendor focused on Web Services and SOA governance, who are involved in the UDDI standards initiative and the OASIS UDDI Technical Committee®. Systinet provide a suite of products including

- Systinet Server for Java (formerly know as WASP Server) and Systinet Server for C++
- Systinet Developer for Eclipse
- Systinet Business Registry

Now at version 5.5, the Systinet Business Service Registry provides a platform independent implementation of UDDI V3 (and UDDI v2) as well as a number of value added capabilities. In terms of governance the key extension provided by the Systinet Business Services Registry is a set of life cycle applications and services. As illustrated in Figure 1, this is a useful approach as a registry can be active at many stages across the lifecycle, and does not necessarily play just a simple publish/discover role.

Classification

The Services registry provides a logically centralized source of information about services that is available to all participants in the life cycle. At the core of the registry implementation is the requirement to establish classification systems for the data types that provide a stable and consistent description system.

Systinet has recently announced the Governance Interoperability Framework (GIF). This initiative, supported

1http://www.bindingpoint.com/
2http://www.salcentral.com/
4http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec
by vendors such as HP, Amberpoint and Actional, is intended to provide a common approach to publishing and discovery of services metadata, with interoperability between the registry and other components of the SOA infrastructure such as Service Management, Security and Integration.

Based on WS-* standards, (WSDL, BPEL, WSRP, WS-Policy and WSDM) this initiative aims to establish interoperability specifications as well as use cases for mapping business service metadata and policies to the Business Service Registry. This should provide classification guidelines and recommended schemas that provide common approaches for the publication and discovery of Services based on the examples shown in Table 1.

<table>
<thead>
<tr>
<th>Taxonomies</th>
<th>Service Type</th>
<th>Domain Business Area</th>
<th>QoS Response Times</th>
<th>Mapping Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policies Capabilities and Constraints</td>
<td>Technical</td>
<td>SLA</td>
<td>Regulatory</td>
<td>Corporate</td>
</tr>
<tr>
<td></td>
<td>WS-I Security</td>
<td>Availability</td>
<td>FDA</td>
<td>SLA Governance</td>
</tr>
<tr>
<td>Specifications</td>
<td>Transports</td>
<td>Security</td>
<td>Interfaces</td>
<td>Documents</td>
</tr>
<tr>
<td></td>
<td>HTTP</td>
<td>HTTP Digest</td>
<td>WSDL</td>
<td>Functional Specification</td>
</tr>
<tr>
<td></td>
<td>JMS</td>
<td>X.509</td>
<td>XML Schema</td>
<td>API reference</td>
</tr>
<tr>
<td></td>
<td>IOP</td>
<td>Kerberos</td>
<td></td>
<td>Examples</td>
</tr>
<tr>
<td></td>
<td>SMTP/POP</td>
<td>XML Sign</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Business Service Registry Classification and Metadata Examples (Source Systinet)

5GIF Participants include AboveAll, Actional, AmberPoint, Composite Software, DataPower, Hewlett-Packard, Layer 7, MetaMatrix, Reactivity, Systinet, and Service Integrity
A key benefit of this approach is that when combined with the lifecycle services provided, the Business Services Registry can provide the core audit of Service activity.

Approval and Publication
To ensure the validity of registry entries and overall quality of the metadata to promote reuse among other usages, the Systinet Business Service Registry provides an approval process. This is exposed both as self service via a browser and as a Web Service so that it can be integrated into tools and portals. The approval process can have various behaviors. For example the process could be:

- An automated one whereby the Service is validated and approved automatically against various criteria and policies.
- Or submitted via a role-based approval process, with submissions processed by someone designated with the role of approver.

The process provides a number of extensible validation steps, so users can add their own scripts, or invoke external services – which could for example be long running operations that run a battery of tests. Validation or certification could include policy compliance, protocol usage, or validation and enforcement of classification including customer-defined taxonomies. One test not included by default at present is WS-I compliance, but this can be integrated into the process via the API.

To support approval and publication, Systinet provides both a staging and production registry with the approval process moving entries from one state to the other.

Discovery
Discovery takes place at many stages of the lifecycle, for example:

- Design-time Discovery. Early in the development process analysts and designers might search for candidate Services for possible use in their projects.
- Applications might search dynamically for new Services to use at run time in response to business change, or version changes.
- The runtime infrastructure might search for Services or Service locations for reasons of routing, scalability or backup.
- Service Management might look to discover Services to manage.

Figure 2: Business Service Console Showing Certification Status
Whilst UDDI includes extensible categorization systems to aid Service Discovery, Web Service protocols such as WSDL are not rich enough to enable Service Consumers to fully understand the function and complete specification of a Service. This is perhaps more a requirement for analysts and designers rather than the runtime discovery of alternate Service locations. One of the useful roles of UDDI is therefore to provide links to other sources of information that provide further more comprehensive specifications.

Service Discovery is made against the production registry. The Systinet Business Service Registry provides rich support for business service classification, with configurable taxonomy support and a range of standard classifications systems built in such as North American Industry Classification System (NAICS), Standard Industrial Classification (SIC), Universal Standard Products and Services Codes (UNSPSC), Universal Standard Products and Services Codes (UNSPSC), ISO 3166 Geographic Taxonomy.

**Visibility**

A configurable Business Services Console (BSC) and extended set of business service abstractions enables users to discover and understand Services from several perspectives, such as: business, technical, and organizational taxonomy, as illustrated in Figure 2. Standard views include taxonomy, whether a Service is managed (i.e recognized by the Service Management infrastructure), QoS, or lifecycle status such as development, QA, certified, production, deprecated. As well as supporting the discovery of Business Services based on business and technical classification the BSC enables users to look at published models such as WSDL, XML Schema and XSLT without having to understand how these are stored in the UDDI data structure, and provides links to further information sources that aid understanding such as UML models or test cases.

**Feedback**

An important factor in the Service discovery and consumption process is selecting Services not just on the basis of functionality, but also on non-functional issues such as QoS and other governance data. It is good practice for organizations to classify Services by Service Level, in terms of the agreement level under which they are offered. In addition it is useful to have actual QoS information based on actual performance.

With Service Management tools already collecting the relevant information required such as transaction volumes, average response times or availability, it seems natural to use this to provide a feedback loop by which production Services can be observed and measured.

Systinet have worked with AmberPoint and HP to feed this information back into the Business Registry, enabling Service Consumers to discover Services by SLA and QoS. Work is also ongoing between the UDDI and WSDM Technical Committees (TC) at OASIS to publish a technical note for mapping of WSDM metrics into UDDI and for the registration of management endpoints. Note the comments on GIF in the classification section above.

**Versioning and Change Notification**

Web Services have no explicit versioning mechanism. Common practice is to assume that changes to a Service interface results in the publication of a new Service, so that existing Service Consumers are not impacted by the change. UDDI V3 implements a subscription mechanism supported by the Systinet Business Registry whereby Service Consumers can receive change notifications by registering their interest. This requires them to register the areas in which they are interested in receiving notifications of changes or new entries, and also implementing a Web Service by which they will then receive any notifications.

**Federation**

Federation isn’t a lifecycle stage as such, but it is useful to reflect how the Systinet Business Services Registry supports usage within a closed ecosystem of business partners, or perhaps just autonomous divisions of a global company that each independently maintain their own registry. UDDI was of course designed from the ground up to support federation recognizing the potential organizational separation of Service Providers and Consumers.

To keep multiple registries in synch, the Systinet Business Services Registry supports replication using the UDDI v3 Subscription mechanism rather than the use of proprietary database replication. Multiple registries can be federated by replicating information between registries, either using a top-down one way replication strategy when federating a hierarchy of registries, or a two-way replication between peer registries, such as between business partners or between divisions in the absence of an enterprise root registry.

The approval process can be put in place across any of the federated registries. For example, one participant could
publish Business Services to their own registry, which are then submitted to the approval process and only replicated into other participant’s registries upon approval by them.

Conclusions
Systinet provide a leading-edge registry capability based on UDDI V3. This is perhaps not surprising given their level of participation in the UDDI initiative. At the same time, the Systinet Business Services Registry provides sufficient added value capabilities to differentiate it from the functionality common to other UDDI implementations.

With many other UDDI implementations coming from platform vendors, having a platform independent registry can be useful to a heterogeneous enterprise or ecosystem. Registry independence is also useful to ensure Service governance is independent of individual platforms and development tools. Whilst these might each have their own standard UDDI implementation, at the same time they may also have incompatible approaches and lifecycle processes, which in turn could lead to inconsistent validation, certification and approvals.

With the use of Web Services and the transition to SOA growing steadily, at the same time so too does the pressure for improved business and IT governance. Introducing a more formal lifecycle approach to SOA therefore becomes increasingly essential, and the lifecycle services provided by the Systinet Business Services Registry can play a pivotal role in achieving this.

Links
Systinet – http://www.systinet.com
UDDI – http://www.uddi.org
CBDI Raison d’etre
We aim to provide unique insight on component and service oriented technologies and processes for the software industry and its customers. To provide high quality analysis and other information resources on best practice in business software creation, reuse and management. To maintain the highest level of independence.

ModusOperandi
The CBDI Forum has a number of channels:

- Subscription services – provision of continuous commentary and information.
- Workshops and seminars – providing indepth briefing and guidance on advanced architectures, processes and practices.
- Consulting – including related product management and marketing advice, application audit and guidance, technical and business evaluation for investment

How we compare with others
We are not a mass market, media oriented organization. All material published by the forum is unique. The majority of material is published by our own analysts, or commissioned from others, and under strict editorial control to ensure accuracy. We rigorously exclude spurious marketing.

We provide depth insight which covers a narrow topic footprint in a deeper way than the other analysts, and in particular cover not just the technology, but also the architectures, usage, practices and processes.

Also we are unusual as analysts; we do not simply echo what the vendors say, we are a think tank, identifying new ideas, opportunities and providing stimulus for thinking. We are thought leaders providing ideas generation and a rich source of conceptual thinking based on practical, real world feedback.

Who Reads the CBDI Journal
Technical leaders including Technical and Application Architects, Business Analysts, Consultants, CTO’s, Designers, Product Strategists, Senior Developers, Project Managers, CIO’s etc. Subscription is split roughly 40% USA and 50% Europe.

Contact Us
For further information on any of our services contact us at: info@cbdiforum.com or +353 28 38073 (international)

IMPORTANT NOTICE: The information available in CBDI publications and services, irrespective of delivery channel or media is given in good faith and is believed to be reliable. CBDI Forum Limited expressly excludes any representation or warranty (express or implied) about the suitability of materials for any particular purpose and excludes to the fullest extent possible any liability in contract, tort or howsoever for implementation of, or reliance upon, the information provided. All trademarks and copyrights are recognised and acknowledged.