Enterprise Business Process Management – Architecture, Technology and Standards

Donald F. Ferguson, Ph.D.[1], Marcia Stockton[2]

[1] IBM Fellow, SWG Chief Architect, IBM Software Group, USA
dff@us.ibm.com
[2] Senior Technical Staff Member, IBM Software Group, USA
mls@us.ibm.com

Abstract. All enterprises' operations require integrating information, and processing information with applications. This has been true for decades, if not centuries. Information and application integration has evolved from completely person centered verbal communication (blacksmith to apprentice), through paper documents-mail-fax, email and Web page interactions. The information and applications control the flow of goods and operations on them. *These are the business processes of the economy*. In some cases, enterprises are almost purely information processing businesses, e.g. insurance.

The past few years have seen explosive growth in direct program-program interaction for application integration. This removes manual steps. The improvements in reliability and efficiency are tremendous.

Controlling the sequence of program interactions and information flow is fundamental to an enterprise's functions. Knowing the status of the flows is equally fundamental. Automating, monitoring and optimizing the flow is the field of business process management. The past two years have seen the emergence of several architectural and standards based innovations.

This paper provides a technical overview of the standards, architecture, programming and runtime models that make modern BPM possible. The paper focuses on the end-to-end model. We also describe concepts that are not widely documented elsewhere, and do not discuss well-documented concepts. For example, we do not describe BPEL4WS or WSDL. We do, however, explain the role of SOA containers and their enablement of automatic process monitoring.

In biology, convergent evolution is a powerful explanatory paradigm. Platypuses and otters are strikingly similar animals despite being extremely unrelated. Their evolutionary paths converged. An analogous phenomenon is occurring in the business world.

A *business architecture* defines the structure and behavior of an enterprise. An IT or application architecture describes how the computing systems are structured. Coming from vastly different starting points, the business architectures' and IT architectures' evolutionary paths are converging. The exposition briefly discusses the co-evolution of enterprise business designs and IT solutions.

1 Model – Assemble – Deploy – Monitor

Figure 1 provides an overview of end-to-end *business process management (BPM).* Business professionals collaborating with IT professionals define a model (architecture) of the business. Often the model is simply a set of "business processes," such as the steps involved in processing a purchase order. Additionally, the model may include business artifacts (purchase order, bill of materials), policies (schedule premier customers ahead of others) and the business components (shipping department, finance and accounting). The business model could include key performance indicators (KPIs). Examples of KPIs include: percentage of purchase orders that complete without manual intervention, or average dollar value of submitted shopping carts. These KPIs directly measure business performance, e.g. profit, customer satisfaction.

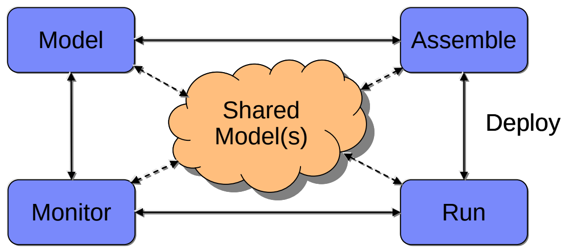

Figure 1. BPM Loop

The fundamental goal of BPM is to iteratively and coherently describe and implement the business model through the development stage, into running systems and monitoring KPIs. This is what distinguishes BPM from more classic approaches to application development and execution. Existing application systems are often vertically integrated with only fragmentary views of the business. The systems typically do not report on satisfaction of business goals, or if they have this capability to any extent, it was added after the fact.

Many recent innovations make the achievement of BPM goals more realizable. These innovations include service oriented architectures, Web services and standard languages for describing business processes, business artifacts, business events and services.

This paper provides an overview of the recent innovations. The breadth of standards and concepts often make seeing the forest difficult, as there are many changing trees. There are, however, an emerging set of architecture models that integrate the many concepts. The models also enable consumable, progressive discovery and application of the concepts. Moreover, the emerging architecture naturally represents the *business architectures and business models.* The business architecture and BPM/SOA architectures' evolution are converging.

1.1 Model

The most popular modeling tools are white boards, followed closely by Microsoft® Office. However, there is an increasing trend to more formal and rigorous modeling. Some organizations use focused tools, e.g. WebSphere® Business Modeler [1] or Intalio [2]. Others use tools that extend spreadsheets, documents or diagrams to incrementally support business process modeling.

Formal modeling has two major benefits:

1. Precise notation: Formal models can capture information in an unequivocal manner, whereas if you use PowerPoint or white boards, people who were not in the room do not know what your dotted arrows or purple circles mean.

2. Reliable hand-offs: Modeling tools can generate implementation templates and the structure of the supporting applications (SOA services), business processes and artifacts. This is less error-prone than reading documents and guessing the desired application behavior. *Bad things happen when programmers guess.*

Surprisingly, the connection between model and assemble (build) is bidirectional. Organizations often use modeling tools to reverse-engineer existing systems to explain the existing applications to business professionals.

1.2 Assemble

Historically, programmers think of "building" an application, or elements of an application. Just as the designers of electronic hardware seldom design elements from the ground up anymore, but assemble existing components and assemblies of components into larger-scale devices, the trend in software is increasingly moving to a*ssembling* an application, or business process. Packaged and pre-existing applications provide much of the necessary function. Implementing the business model and its changes often simply requires assembling (and configuring) existing applications into a *composite application* that integrates the existing systems. A business process coordinates the integrated composite application. This is often accompanied by some development of new services to complete the business process.

Assembly has two sub-models:

1. Structure: Which services, processes and artifacts comprise the composite application? What are the governing policies? … …
2. Behavior: What are the sequences/control flows of calling the applications? What is the data flow? What are the state transitions? … …

An example of assembly tools is WebSphere Integration Developer [3]. Many other companies have similar capabilities, e.g. SAP [4].

1.3 Deploy

The newly assembled, existing and configured applications run on an IT infrastructure (systems, application servers and middleware, packaged applications, publish-subscribe systems …). The deploy phase maps the application artifacts and configuration information to the systems and software environments. However, the incremental nature of deployment creates enormous challenges. The applications, processes and data formats *change while existing applications and transactions are executing*. Imagine changing the flight control system of a trans-Pacific flight.

Unexpectedly, IT systems and application management itself is evolving to a BPM model. Rolling out patches, upgrading software, etc. are business processes. Products are evolving in this space, e.g. IBM IT Service Management [5]. Standards are evolving in the IT governance space. These include IT governance processes (IT Infrastructure Library [6]), and the use of SOA and Web services to manage applications and systems (Management Using Web Services [1], Web Services Distributed Management [8], WS-Management [9]). The essential realization is that complex IT management functions like call center/trouble ticket or software upgrade approval are essentially business processes. Moreover, many business processes are a mix of IT activities and application activities. IT processes associated with hiring a new employee will update payroll systems, employee profile systems and issue configured PCs, activate LAN ports and VPN user ids.

The IT Service Management solutions and standards:

- Define long running, complex IT management operations using business process modeling. The solutions implement the processes using SOA centered workflow/BPM engines.
- Provide SOA Web service application program interfaces (APIs) for management agents.
- Use portals to integrate manual steps within the service management processes. The portals support UIs to applications, work lists, etc.
- Use Web service-based events to monitor processes and compute KPIs, e.g. average time to resolve a problem ticket.
- Build integrated information models for configuration information. The systems use Web service to perform information integration and federated databases.
- Warehouse and analyze "business process events," and automate reactions to event patterns.

1.4 Monitor

Monitoring is the observation of the executing composite applications. There are two types of monitoring:

1. IT monitoring observes throughput, response, resource utilization, problem events, etc. This enables change, for example allocating more servers, to improve satisfaction of business goals (for example, the time to process a purchase order).
2. Business monitoring observes the execution of the solution. (For example, did a purchase order fail a credit check or require manual intervention? Emit an event when a credit check fails.)

The stream of business events enables intervention and reaction. Assume a credit check fails. Part of the business process definition may be *event filters*, which look for events or event streams matching a pattern. A filter for the credit check failure event may trigger an email or instant message to the account representative. Aggregating events into summaries interesting to employees, and making the summaries available through email or a Web portal, is extremely common.

A common element of business models is *key performance indicators (KPIs).* KPIs provide a way for businesses to measure the success of their business models. Continuing the purchase order example, the business may want to:

1. Achieve a target value for purchase orders. If they fall short of the goal, one possible business action is to modify the personalization and product suggestions on their portal.
2. Reduce cost through eliminating manual approval of purchase orders. If the business is not meeting the goal, they may need to modify and extend the rules in the rule system that automates approval.

Business events supply the data that are the basis for KPIs. The solutions store the events in a data warehouse and perform analytics to compute KPIs.

The event and KPI models are *explicit elements of the business process model.* These elements flow through all steps in the Model-Assemble-Deploy-Monitor loop. Business modeling defines the taxonomy and formats for events, the KPIs and the event filters and actions. The assemble phase realizes the model constructs; the deploy and run phases emit the events. Standards emerge in this space, for example the Web Services Distributed Management Event Format.

1.5 Nirvana

The preceding exposition restates the well-known BPM Nirvana. We have been striving for enlightenment for years, if not decades. What is different now? We provided some insight in the preceding sections. The remainder of the paper articulates technical changes that make the goals more achievable.

1.6 A Sample Business Problem

The processing of purchase orders for computer hardware [10] is a surprisingly complex problem[1]. Figure 2 provides a high level overview of the problem. The purchase order (PO) document (artifact) itself is complex and contains many types of differently configured machines (desktops, servers). The PO may also contain requests for other hardware components such as storage devices, networking equipment, etc.

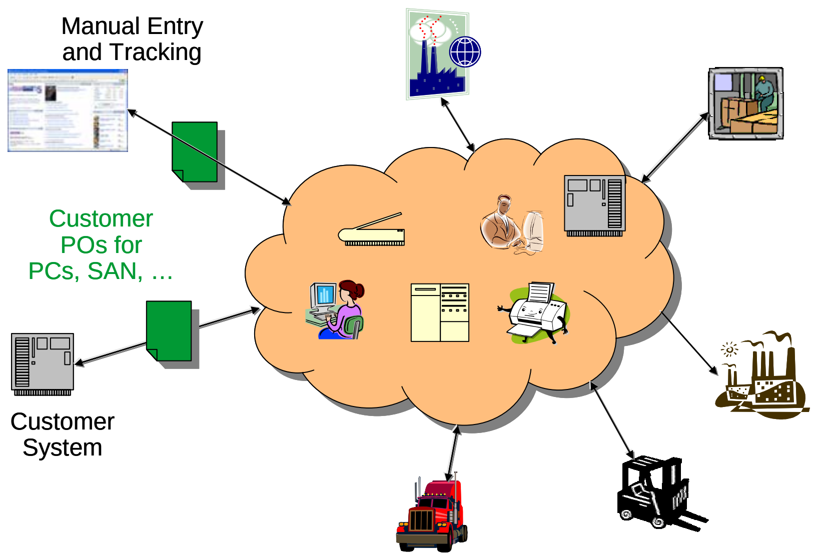

Figure 2. A Typical Business Problem

[1] This sample problem is a generalization and simplification of a subset of IBM's supply change management processes. Readers should not infer anything about as-is or to-be actual solutions.

Customers want a coherent view of their purchase order's progress, a reasonable delivery schedule (the machines should not arrive one-at-a-time), alerts, etc. Provider profitability requires efficient coordination, delivery and assembly of components. Processing of the PO may require business rules to validate configurations or compute volume discounts. The process may call numerical algorithms to optimize production and shipping schedules. In many problems similar to this one, there are many non-integrated applications, different views and forms of business artifacts (PO, product descriptions) and manual tasks. The business process typically involves multiple enterprise sites and business units (for example, server assembly, micro-electronics, financing), and external partners.

Improving this solution has several aspects or sub-elements. These concepts are the elements that make up the business process management solutions. The key elements are:

- Enabling (adapting) existing systems for SOA and Web services, and incrementally reengineering the systems. This enables BPM. It also allows flexible and customizable connectivity, which allows reengineering. Web services and XML provide a set of standards that simplify application integration by having a common protocol and message type space. The common model is a *lingua franca* that eliminates N-squared complexity of connecting different protocols and formats for N systems.
- Externalizing the integrating process flows from the individual packaged applications and existing systems. Modeling and deploying the process in a business (service) choreography engine. This allows flexibility, for example tailoring a process for a specific customer, and visibility (ability to respond to an inquiry like, "What is the status of my purchase order?").
- Enabling the exploitation of useful SOA programming model abstractions such as business rule sets, and the computational optimization of process flows
- Externalizing volatile business logic from services, relying on configuration files/interfaces or strategy patterns [11]. This enables flexibility.
- Defining key performance indicators, and adding instrumentation and monitoring probes to gather data.

Much of the remainder of this paper describes the architecture and standards for applying these design aspects to solve the business problem. Section 3 covers the architecture and technology in some detail. Before these details, we outline a broader picture. Much of the benefits of service oriented *architectures* and *components* derive from the similarity to evolving business *architecture* and *business components.* Section 2 provides a brief overview of this trend.

SOA solutions and business processes span multiple lines of business, geographies and enterprises. Standards play a key role. In Section 4, we provide an overview of important standards and their role. Finally in Section 5, we close with some interesting research and exploratory challenges.

2 Business Architecture and IT Architecture

2.1 Componentizing the Business

Businesses have components: factories, departments, teams. Unfortunately, businesses are very rarely designed from a top-town architectural perspective. Poor or haphazard business designs are much more common than well-conceived ones. There may be duplicate components or subcomponents. Different lines of business within an enterprise may have disjoint, duplicate shipping departments or administrative functions. Interfaces between components are not well-defined. In many cases, there is no clear understanding of components' behavior. This is not deliberate, but because of how the design arose through a process of organic growth and change, with infrequent rationalizations to identify and merge redundant elements.[2]

Convergent evolution is a powerful, descriptive and prescriptive theory. Kangaroos and Kangaroo Rats are not closely related, but have similar locomotion. Platypuses and otters are not closely related, but are similar. In each case, the pair of similar animals independently acquired similar solutions to certain

[2] Nott et al., "Choose an ESB topology to fit your business model", ibm.com/developerworks/library/ws-soa-esbtop/, describes IT implementation structures to support various governance models, arising from different business designs.

problems that it faced in its environment. Convergent evolution explains the similarity between the evolution of enterprise *business architectures* and SOA: both evolved in parallel to address the same kinds of problems, and not-so-coincidentally arrived at very similar approaches.

Enterprises are evolving toward having well-factored and well-defined *business components* [12]. Business components offer well-defined services with standard business object formats. The business architecture detects redundant components and subcomponents. Once detected, the enterprise can consolidate or divest the redundant components to whatever extent their vision for a streamlined business model dictates.[3]

The new business architecture enables selective externalization and flexible placement of components. In fact, many enterprises "own" almost none of the business components: their business model is the composite, aggregate solution.

There is large body of literature on the benefits of the new business architecture which we need not repeat here (please see the references for case studies and citations). Our key observation, supported by the cited literature, is the close alignment of the *business architecture* and the *SOA technical model* in an apparent example of convergent evolution. It is fundamentally this parallelism which enables the IT realization of an enterprise to match the business architecture.

There is a second, subtler observation. Historically, the industry focused on changing IT to make it more responsive to the business needs. Going forward, we foresee a slight change in emphasis. Modern, SOA-based IT solutions have evolved to an effective architecture. *It seems likely that similar pressures will cause the business to become more like IT.*

2.2 Service Oriented Architecture and Components

There are many definitions of "service" and service oriented architecture. We will use the following definition [13]:

> *"A Service is a set of functionality provided by one entity for the use of others. … Opacity is a core component [i.e., trait] of services.*
>
> *Each Service has a Service Description. A Service Description is a set of metadata declaring all aspects of a service necessary for a Service Consumer to understand the service's externally inspectable aspects."*

These definitions are correct, but are not sufficient to provide a SOA realization of a business architecture that supports BPM. For example, business components provide services to other business components, but also *use* services. The Service Component Architecture [14] is an evolving set of specifications to define a *SOA component model*. Many of the elements of SCA directly support BPM.

The definition of service description identifies the needs for metadata. SCA brings together several standards and patterns for formalizing this metadata. The standards are:

- XML Schema Definition (XSD)
- Web Services Description Languages (WSDL)
- WS-Policy
- BPEL4WS
- Elements of the Unified Modeling Language

Finally, SOA relies on loose coupling and flexible binding. If service **A** needs to invoke an operation ***O*****,** the choice of an operation**-*O*-**capable service (component) may be made at runtime. Continuing our purchase order example, the specific shipping service to use may depend on particulars of the purchase order, current business contracts in effect between the parties, etc. Loose coupling and flexible binding are accepted concepts in SOA. The technical architecture we describe in Section 3 identifies the architecture models for *mediation* and integrating *event-driven processing* with SOA. These concepts offer greater flexibility than

[3] Even during a rationalization process, a redundant function is not necessarily an automatic candidate for divesture or rationalization. There are good reasons why a business might or might not want to do this. A multinational may need to allow local autonomy of its country operating units, for example, in regard to conduct affected by local taxation, laws, or employment regulations. Or a company may be a loose federation of unrelated product divisions that remain autonomous and each conduct their own advertising and brand image campaigns. The structure of IT services should be aligned with the business design.

dynamic binding. The message transport mechanism, the *enterprise service bus (ESB),* contains active logic within its message-processing components. This logic can, for example, transform messages into a format compatible with the message receiver, or modify the routing of a message based on the data payload, enabling the ultimate in SOA runtime flexibility.

We can see how integrating a set of standards provides a more complete, coherent model. The model facilitates the business process modeling and assembly phases. It also provides a more complete, concrete rendering of business concepts in an IT model. Extensions to the architecture, which are patterns for services/components, improve the fidelity for representing business architectures. The ESB allows dynamic connections between business processes and the services it choreographs.

An additional element of SCA is an increasingly portable programming model for service components. This model, which extends beyond language-specific approaches like J2EE, offers many benefits to both programmers and CIOs, most notably avoiding vendor lock-in to specific products. SOA flexibility can be equally as valuable to business architectures and business process management as it is to IT implementation. As an example, consider that the evolution of a business component architecture may dictate migration of “programs” to partners, service providers, etc. A portable approach to implementing services and components facilitates this flexibility.

3 Architecture Overview

This section refines the concepts introduced in 2.2. Much of the application architecture focus is on ensuring that the *application (SOA) architecture* supports the *business architecture.* There is a special emphasis on coherence of the business process models and the SOA models. This coherence helps integrate the modeling phase of a solution with the assembly, run and monitor phases. So, the first element of the application architecture we describe is the *process model.* A process choreographs a dynamic set of services. We follow the process concepts with modeling and assembly of dynamic sets of services.

A second tenet of the application architecture is well-defined services, metadata and flexible binding. These are key requirements for mapping the business architecture to the application architecture. To ensure flexibility and responsiveness to changing business designs, the portfolio of services and the processes must be able to evolve independently. We explain a general set of approaches to service description and metadata, and an evolving standard architecture (the Service Component Architecture).

To be useful, a service must be executable. Runtime products that support the *monitor* phase of the BPM lifecycle play an extremely important role in SOA and SCA. Section 3.5 describes the integration of monitoring/management with SOA and SCA.

Existing systems, such as CICS® and IMS, offer robust support for change management. Programmers for these systems find them intuitive and easy to program. Many of these benefits derive from applying a set of well-known best practices or design patterns. Similar patterns are emerging for SOA, and we briefly enumerate the core patterns.

3.1 Application Architecture

Each of the following subsections is devoted to one aspect of the application architecture: composition, components/artifacts, policy, bindings, Enterprise Service Bus, and configuration/customization.

3.1.1 Composition – Processes and Structure

3.1.1.1 Business Processes – Motivation. Figure 3 illustrates two contrasting approaches to building SOA business solutions.

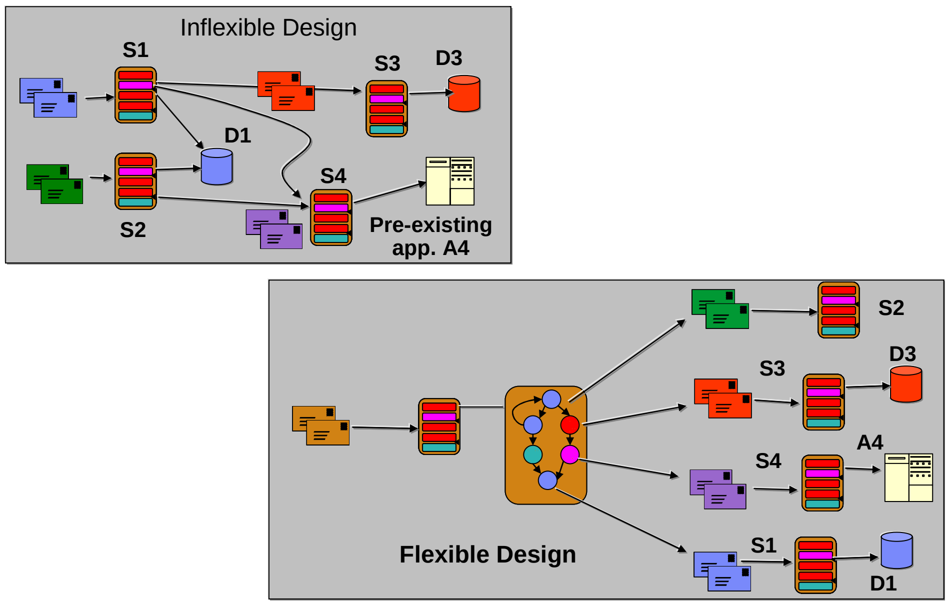

Figure 3. Well-designed Business Processes

The first approach (the top diagram in Figure 3), labeled inflexible, is just that. The solution is inflexible and violates two SOA design principles. Two services access the same data (S1 and S2 access D2). This violates the principle of well-defined interfaces. The two services effectively communicate through private data signals. The services' implementations (e.g. S1) also orchestrate or choreograph other services within their implementations.

How do these violations cause inflexibility? Three problems are salient:

1. The ability to independently evolve and refine services provides flexibility. Consider the PO example of Section 1.6. The overall business solution has sub-services that compute PO discounts and validate the hardware configurations. The sharing of data by services makes it difficult, costly or impossible to modify the implementations of the services that read/write shared data (e.g. hardware configurations in POs)).

2. Embedding sequences and control flows for service invocation within the code of a service makes change difficult. In our example, what if the enterprise wants to add an additional hardware validation service? Making this change to the inflexible design requires finding and changing Java or C code that embeds the calls to the existing validation service, and replacing them with calls to a new validation service that ensures compatible operating system configurations and device drivers for POs that contain mixed hardware and software line items.

3. Embedded signaling through data and encapsulated (implicit) service choreography make flexible modeling and monitoring more difficult. Detecting signals through shared data and emitting a BPM event requires source code modification. The same may apply to embedded choreography. Process steps and transitions are not explicit.

The inflexible example is characterized by architecture, data, message flows and logic that are all implicit, making the solution flat, ad hoc and two-dimensional.

Figure 3 also illustrates a more flexible design in which services are atomic4 elements. The implementation of the atomic services may access data, call pre-existing applications (e.g. A4 in the diagram), use libraries, etc. A key design principle is that the sharing of data between services is prohibited; services may not even

[4] Our use of *atomic* here is different from the ACID property of transaction models. Here, *atomic* refers to the smallest indivisible elements of functionality.

call applications that share data. Another principle is that all aggregation and control flow of service operations must be external to the atomic services, and reside in a well-defined business process.

The flexible example makes all of these design elements explicit: this solution has a three-dimensional architecture that arches over the implementation parts, tying everything together by plan. Similar design elements can be found at several levels of scale and detail. This makes it just plain easier to understand, too.

What motivates these goals?

- Business process monitoring is simpler and easier to change if services communicate solely through their interfaces. Most systems support dynamic monitoring of the "ins" and "outs" of service invocations. Business performance monitoring becomes dynamic, and does not require application modification[5].
- The business process definition is typically a *text document* in a well-defined language. The workflow engine reads and interprets the document, and executes the activities. It is typically easier to find, modify and save a document than to perform the same for Java or C code.
- It is simpler to understand the relationship between process elements in the business model and code that implement them. There is a large gulf between high level business constructs, e.g. "this step checks for existence of the account," and the corresponding Java or C code that implements the step and calls other Java code. An intermediate level of abstraction that more closely mirrors the process diminishes the gulf.
- Making the flow of logic explicit by placing it into an easily modified text document improves flexibility. By contrast, consider the inflexible alternative: the implementation of common sub-processes in programming languages. The drawbacks of this approach are evident when you consider our example of modifying the set of validation services in the processing of a PO, which might require finding and modifying several instances of the code that sequences calls to services. Even the state of the art prior to a BPM-style implementation, a common library, would help eliminate redundancy, but nevertheless problems remain:
    - Application sub-elements in complex business solutions are often multi-language, e.g. C, Java, COBOL. The sub-elements run in different platforms models, e.g. CICS, J2EE™, batch jobs.
    - Modifying libraries often requires stopping processes and servers, re-linking programs and restarting servers.

    Externalization of business processes that coordinate well-defined services will mitigate, and eventually eliminate, the inflexibility. They key technologies that enable this externalization, are:
        - The enterprise service bus (Section 3.1.5)
        - The use of external policy and rules (Sections 3.1.3, 3.1.6), and
        - Dynamic structure composition (Section 0).

3.1.1.2 Behavioral Composition. Experience with products and solutions reveals three fundamental approaches to composing the behavior of services to form business processes and composite applications.

- A control flow and composite activity model – The standard approach to control flows and composite activities is BPEL4WS. There are many documents on BPEL4WS, and we will not discuss it here.
- A state machine model – State machines are also widely understood, and we will not discuss them.
- A message flow/event model – We discuss the message flow/event model in Section 3.1.5.

SCA takes an interesting approach to behavioral composition. In SCA, BPEL4WS processes and state machines are SCA components. Externally, these component types look like any other. This approach may appear to violate some of the principles we articulated in Section 3.1.1, e.g. embedding control flow in components. We clarify these issue later in this paper.

[5] Support for transparent, dynamic monitoring is one capability that containers provide (see 3.4)

3.1.1.3 Business Processes – Structural Composition. A service component in SCA documents the interfaces it supports and the interfaces it requires. We explain the details of the approach to documenting the interface in Section 3.1.2.

Figure 4 illustrates the principles involved in structural composition. The model "wires" required interfaces to implemented interfaces. Interface compatibility can consider all aspects of the metadata describing the interface. Programmers wire service components together to create assemblies called *modules.* The model is recursive in that modules may, in turn, export interfaces and require interfaces. Assemblers can then wire modules together to build subsystems and solutions.

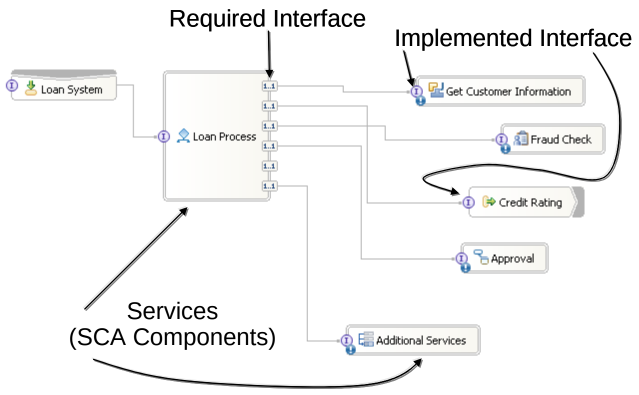

Figure 4. Structural Assembly

The wiring approach simplifies the process of assembling services into new solutions, and modifying the services that comprise a solution. The key benefit is that all service components have the same external model. Programmers can perform assembly without being aware of a component's implementation.

Finally, assembly can be dynamic. The model is not static. Because of loose coupling and the enterprise service bus functions, modules and solutions can change rapidly.

3.1.2 Service Components, Service Description and Business Artifacts

A component implements one or more interfaces, which it defines in WSDL[6]. The component also identifies one or more *required interfaces.* In our example, the PO management process declares a requirement for the shipping service. The operations on an implemented or required interface may be in either direction, e.g. the required shipping service may call the PO service.

Another key element, not yet explicitly part of SCA, is a *business artifact.* Services and their operations are the *verbs* that implement an enterprise's business model: *create purchase order*, *approve purchase order*. Business artifacts are what the verbs manipulate, e.g. account, purchase order. WS-ResourceFramework [15] is an evolving standard for integrating Web services and business artifacts. WebSphere Process Server and WebSphere Integration Developer support *business state machines (BSMs).*

Most literature on SOA and Web services state that Web services are stateless. This is true only in a narrow sense. Most Web services manage and manipulate state data, i.e. the business artifacts. The opacity requirement for services implies that business artifacts not be externally visible, and a service's operations completely describe its behavior. There are three ways in which a business artifact becomes externally visible, however.

[6] Many SCA implementations will support defining interfaces in native languages, e.g. Java, COBOL, to simplify the developer role. Tools generate the WSDL when necessary.

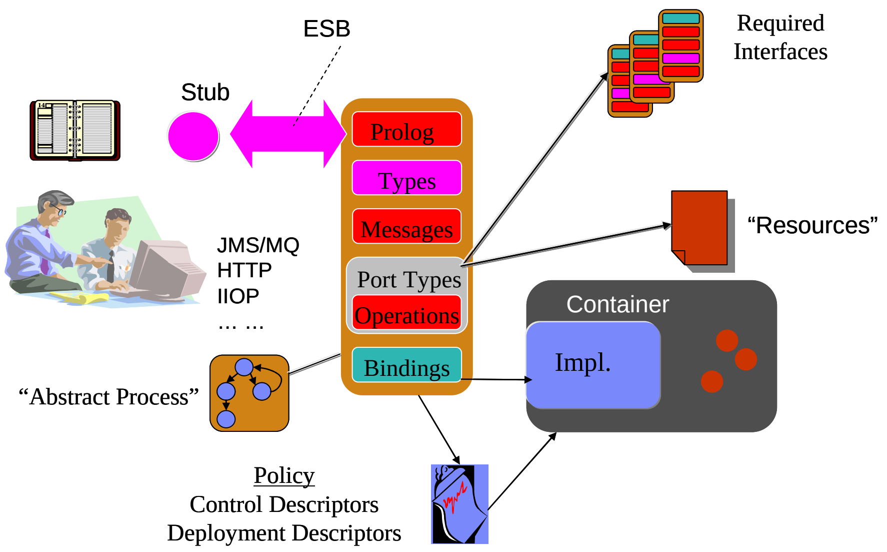

Figure 5. Components and Interfaces

1. A service's operations send and receive messages. Typically, the messages are projections from an underlying data/information model.
2. The concept of a *service description* allows a service to document metadata that enables consumers and clients. Documenting the business artifacts is extremely useful for many scenarios. Moreover, UML and other disciplines demonstrate the value of state machine models for artifact lifecycle. The state machine documents how operations transition an artifact's state, and which operations are valid for which states. The state model is also intuitive for process monitoring. In our example from 1.6, "What is the state of my purchase order?" is a natural question.
3. Business artifacts, lifecycle states and associated operations and messages provide additional metadata to aid programmers using a service. The constructs also aid in service binding and discovery. The metadata enables a consumer and provider to ensure that they have a common understanding of business artifacts' behavior.

Business artifacts and state machines have limited ability to express behavior. There may be aggregate behavior that a service wants to describe that spans multiple operations, interfaces and artifacts. In our example, what behavior (operations, state/artifacts) does the service expect from a set of consumers, or services that it uses? BPEL4WS introduces the notion of an *abstract process*. A service connecting to (or connected to) in a specific *role* in the collaboration can obtain additional understanding of the component's behavior from the abstract process. In some sense, the abstract process projects or subsets a service component's "business process" from the perspective of a consuming service or providing service.

The evolving Service Component Architecture provides a rich set of constructs for describing the interface syntax and behavior of services. We must, however, make two important observations:

1. The model is *consumable* and *incremental*. This section has briefly explained many concepts. Programmers and solutions can start with simple interfaces and gradually increase sophistication.
2. We have explained the concepts for describing a *logical external model* of the component. The innards may be, and typically will be, completely opaque.

3.1.3 Policy

The preceding service description concepts focused on documenting *what* a service/component does. In a purchasing application, a service may *create*, *approve*, and *cancel* purchase orders. Equally important is *how* the component implements its functions and what it expects of callers or services that it calls. A component may have policy annotations on the interfaces and other elements. The policy annotations typically describe quality-of-service related concepts. For example,

- Does the component require that callers sign or encrypt messages? What certificate authorities does the component trust?
- The component may require reliable messaging, ensuring that there are no duplicate or out-of-order messages.

Web service standards such as WS-Transactions [16], WS-ReliableMessaging [17], and WS-SecureConversation [18] define formats and protocols on top of the base SOAP and HTTP protocols. The standards provide additional quality of service for interactions. The standards and protocols often have "switch settings" or parameter choices (e.g. what message digest algorithm may be used in a security protocol). Each Web service interoperability specification typically has a companion WS-Policy [19] extension letting a service document the details of protocols that it supports/requires.

The base policy framework supports Boolean composition of policy statements, for example AND/OR/NOT. This allows a service to specify sets of policies that it supports. During binding, the infrastructure performs a *policy intersection* computation to determine a choice of policy settings that both the caller and called support.

3.1.4 Bindings

Most BPM systems support protocols in addition to SOAP/HTTP. Metadata associated with a component documents support for other protocols. The binding may support the policies natively, for example reliable messaging on WebSphere MQ.

3.1.5 Enterprise Service Bus

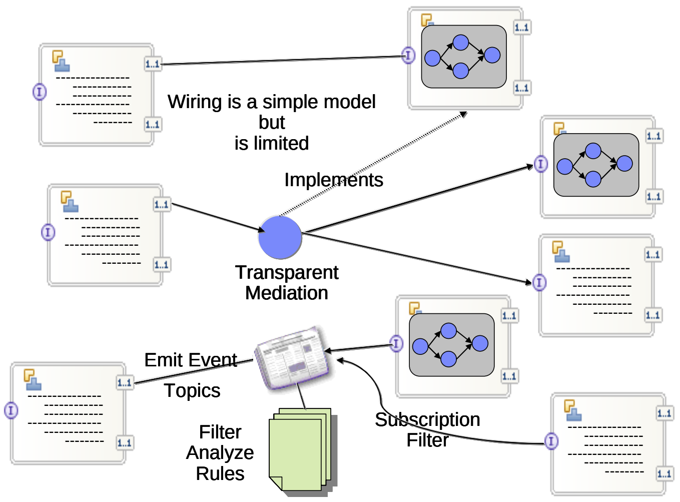

Figure 6. Enterprise Service Bus

Wiring is a powerful concept because of its simplicity and intuitive model. The approach appears limiting and in conflict with the flexible, dynamic and loose coupling of SOA. The service component model provides a layered approach to increasing flexibility and function. Logically, all wires flow through an *enterprise service bus.* A continuum of functions is available that may be provided by a wire.

- The wire may represent a direct, design-time connection between two components.
- The wire may represent a logical connection. The caller (wire source) may define a specification used to resolve the wire/connection at runtime. The specification can contain any of the metadata elements: interface, policy, binding, state machines, etc. The ESB's logic selects the "best" active component to be the wire's endpoint.
- The wire may itself be a component that implements the *intermediary* or *mediation* pattern. The simplest mediation is the *selector pattern* [11]. A selector mediation examines message and context,

and chooses where to route the message. In a PO system, the mediation may examine the customer number and route the approval request to a specific system.

- The mediation may also implement an *event broker* or *publish-subscribe* pattern. Messages sent by a source, by calling a required interface, become events on topics. The wiring process may connect a component's input to the event space using a filter to determine which service requests should be routed to a component.

Modifying and editing the behavior of wires through mediations is transparent to the service components which are the endpoints of wires. There is often a special developer role, *integration specialist,* that uses tools to examine wires, add behavior, etc. Consider a simple example:

- The purchase order process **P** in our example in Section 1.6 has a wire connection to a service **C** that computes discounts on POs.
- The business determines that a new rule based system is necessary to handle POs over $1 million. The enterprise develops the rule sets and deploys the rules as a new service **C2** onto the ESB. The interface is slightly different from **C'**s.
- The integration specialist associates a mediation with the wire from **P** to **C.** The mediation
    - Examines the PO
    - If the value is less than $1 million, the message flows to **C.**
    - If the value is greater than $1 million, the mediation
        - Transforms the message to **C2**'s format
        - Routes the message.

Mediations typically implement one of four functions:

- Routing or selection
- Transformation
- Side effects, for example logging.
- Augmentation, i.e. accessing a database or service to add information to messages.

ESB tools support composing basic mediations into composite mediations. The example above is a composite of routing and transformation.

The key benefit of the ESB mediation model is flexibility. New services, or new versions of services, can join the bus. An integration specialist or other developer (not necessarily a programmer) can change service request routing transparently to existing, deployed services and modules.

It is important to note that an ESB is not necessarily an external system or infrastructure. In many cases, sub-buses of the ESB execute within the endpoints and in the application processes, providing a level of performance comparable to a vertically-integrated or monolithic implementation.

3.1.6 Configuration and Customization

Service components are highly configurable. Configuration may occur during assembly or at runtime. Some component kinds are inherently configurable: for example, rule sets, BPEL4WS processes and business state machines are effectively documents that an engine interprets. For such component kinds, configuration is as simple as changing the underlying document and setting an "as of time" for the change to take affect.

SCA defines a property model for binary components. The component may define an accompanying "document" that declares parameters that configure the component's behavior. An example might be the maximum value of a PO. Developers set the properties during assembly. It may also be possible to change the properties after component deployment.

Finally, there is a style or design pattern for building more complex customization and configuration. Consider the following pseudo-code:

```
POREf addLineItem (POREf P, LineItem li) {
      if (p.noLineItem() > poProperties.getMaxLineItems()) {
            // Do something horrible
      }
      … …
      … …
```

The *poProperties* may itself be a service component to which the PO has a wire and required interface. This factors volatile logic out of the core business behavior, and supports customization more complex than simple parameters. This is an example of the *Strategy Pattern [11]*. What SCA and the application architecture add is the ability to insert mediations. The actual instance of poProperties may change over time, may depend on the value of the PO or the customer, etc.

3.2 Kinds and Styles of Components

Different is hard. Programmers approach problems with specific skills. What one programmer finds easy another may find hard. Some programmers prefer the BPEL4WS flow model while others prefer state machines. SCA has an extensible set of *component kinds.* Various vendor products surface role- and component-kind-specific tools for building SCA components. In general, the programmer providing a component focuses on the task at hand, and is unaware of SCA. This approach limits the scope of what any individual needs to know, resulting in a simpler development experience for the individual and team.

Certain component kinds are more natural for certain tasks. Defining a long running process in BPEL4WS is more natural than using C.

We have discussed many SCA component kinds in this paper: Java, C, application adaptors, business rules, state machines, BPEL4WS processes, and mediations. The set is extensible and many other kinds will emerge. There are many documents on the Web describing component kinds and supporting products. There is also an open source supporting SCA and various implementation approaches [20].

3.3 Implications for Model – Assemble

Sections 3.1 to 3.2 explained recent standards and product support for Model and Assemble. There is a highly flexible, dynamic model for defining business processes, including structure and behavior. It is also possible to dynamically modify solution behavior using various techniques.

Component kinds also provide support for Model ↔ Assemble. Many of the component kinds naturally represent refinements of concepts in many modeling tools/methodologies. Business state machines are a natural mapping of documents in a business model. The coherence between model and assemble simplifies model transformation, and makes transformation and refinement less error prone.

3.4 Runtime Architecture and Deployment

Figure 7 provides an overview of the two key concepts in the runtime architecture. The first concept is the *container* abstraction. Logically all components, including business processes, execute/reside in a container. The container "wraps" the component, pre-processing and post-processing all service invocations. The pre- or post-processing implements the policies associated with the component. For example, if the component specifies that messages from a sender must be in order and digitally signed, the container implements these functions before passing the message to the business logic.

The container abstraction and policy have many advantages. Implementing quality of service logic should be separate from business logic. The necessary policies may change over time, or from composite application to composite application. Using the declarative model enables this flexibility. Most runtimes support a service provider interface (SPI) for adding new quality–of-service functions and policy types.

The runtimes typically support several *container kinds.* These include workflow engines, the ESB, containers for code (J2EE™ application servers), XML enabled databases, etc. The containers may be on separate runtimes and machines, or often within a single runtime. Finally, there is a set of *distinguished* *services* available on the bus. Examples are transaction/coordination manager, logging, authorization and authentication.

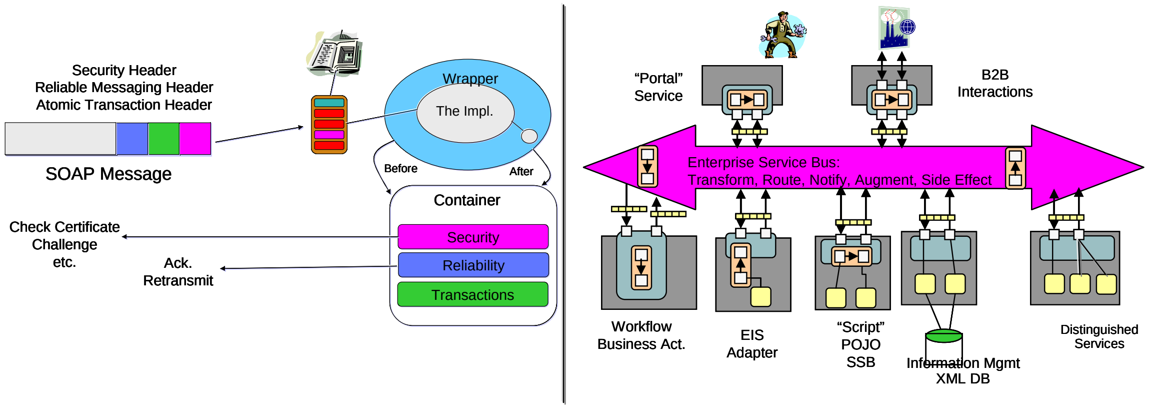

Figure 7. Runtime Architecture

A complete application has associated metadata that defines the components, modules and their versions. The metadata also defines the container types for each component. Modern deployment systems like Tivoli® Provisioning Manager [21] implement deployment and change-management business processes supporting the deployment and update of solutions.

3.5 Monitoring and Management

Business process monitoring and management occurs through a *business event* model. The modeling and assembly phases define explicit business events. For example, there may be an explicit "emit business event" activity in a state machine or BPEL4WS process. The component may emit this event in the same way that code logs a condition or an exception. Containers also support dynamic, automatic event generation. For example, the container may emit business events when messages arrive for a component. The event includes the input message and response. Most BPEL4WS containers support configurable monitoring of BPEL4WS processes, emitting business events for selected activity transactions.

There may be a mismatch between the emitted event format and the observer's expected format. Business events flow through the ESB, which enables routing, transformation, etc. Many ESB systems also support mediations that use rules or logic to examine event streams looking for patterns. When the mediation detects a pattern, it emits an event indicating that the pattern occurred. The mediation may also invoke an operation on a service when recognizing the pattern. This enables automated execution of business processes, rules, and so forth when specific conditions occur.

Business process management systems also enable the logging of events to a data warehouse for business intelligence purposes, and the connection of end-user dashboards to the event system.

4 The Role of Standards

4.1 The Three Aspects of Standards

Standards play a key role in business process management. Our example of Section 1.6 is only possible through the use of standards. Thus, first and foremost, standards are the mechanism for integrating applications and infrastructure between companies, lines of business, geographies in a multinational company, etc. Many people are familiar with and understand interoperability standards like SOAP/HTTP and WS-Interop.

Second, standards enable interoperability between different organizations development tools (and management products), and between products provided by different software vendors. For example, SOAP/HTTP provides runtime interoperability between service **A** and service **B.** The developers use the services' WSDL, XSD, WS-Policy, BPEL4WS abstract processes, etc. to develop the services.

Finally, standards enable *portable* services, components and modules. This area is evolving. J2EE™ is one example. BPEL4WS/BPELJ, XSL and XQuery are other (mostly) portable models. Finally, WS-Policy and policy grammars for infrastructure also enable portability.

4.2 Mapping Standards to the Architecture

Despite their central importance, or perhaps because of it, the standards landscape is in flux and confused. There does seem to be an emerging standards architecture, however. This section attempts to briefly describe the architecture, at the risk of alienating everyone.

XML Schema Definition, WSDL and WS-Policy are becoming *de facto* and formal standards for service metadata. These standards underpin most other standards. A consensus also seems to be emerging around integrated profiles of infrastructure services. Profiles with canonical test cases define how standards combine to solve a specific infrastructure need. An example is the Reliable, Asynchronous Message Profile (RAMP) [22]. Others are emerging, which will clarify the role and integration of extensions to SOAP/HTTP.

There is *a lot* more confusion around Unified Modeling Language (UML), BPEL4WS and Business Process Modeling Notation (BPMN) [23]. Some aspects are clear:

- UML
  - UML supports general modeling through the profile mechanism. For example, it is possible to model complex hardware configurations (servers, switches, operating systems, etc) with UML. Early Common Information Models (CIM) [24] for systems management are expressed in UML.
  - UML is useful for implementation modeling. Many of the UML constructs are suitable for modeling and explaining the implementation of components and their interactions. Use case diagrams are an example. Sequence diagrams are another. Unlike other process modeling techniques, UML sequence and use case diagrams typically document one (or a small number) of representative interactions that may occur.
  - UML is useful for modeling structure, e.g. business artifacts and their associations.
- BPEL4WS, and its extensions like BPELJ [25] and BPEL for People [26], are executable languages. They are not modeling languages. The detail is too great for most modeling tasks and methodologies. Modeling tools should be able to emit/read BPEL dialects, however.
- BPMN is more unclear. BPMS *seems* to be *an approach* to modeling. Many teams follow other methodologies, and the space is fragmented. There seems to be a growing consensus around BPMN and related specifications for business modeling.

There is also some confusion around "events," "state" and "management," with a cluster of specifications from Microsoft and others. The major players have stated an intention to converge the specifications into a logical, coherent model [27].

This paper has not discussed the interaction between people and business processes. Since business processes are Web services, many models for interaction are possible, e.g. J2EE or Web Services Remote Portlets [28] . Business process management is an evolution of forms processing and routing. XForms [29] is emerging as the *de facto* standard for forms.

There are two major areas for increased focus on standards:

- Rules, rule formats and rule engine behavior.
- Standards focusing on business domains. It is hard to define a standard's architecture without agreed, well-defined use cases.

5 Summary and Future Directions

Business process management and monitoring is becoming **<u>the</u>** way that enterprises implement their business models. SOA and Web services have enabled BPM to grow and expand. As the standards continue to emerge, BPM will become progressively more powerful. This is especially true because the standards enable cross-enterprise collaboration, globalization, etc. Much of the emphasis on BPM derives from two observations:

1. SOA, components and BPM, and the architecture of businesses (componentization, partnership, outsourcing, …) are co-evolving to similar design points. Thus, BPM and SOA are the natural ways to implement flexible business models.
2. The industry is realizing that many problems previously solved with non-BPM concepts are in-fact ideally suited for BPM. Complex systems management and data center governance processes are an example. Application development and governance itself is a *business process management problem.*

BPM and SOA are not complete, however. The BPM platform exposes several open problems and directions for research:

- Modeling is a powerful, useful concept. Model constructs and environments have only rudimentary support for reasoning about and analyzing the model. Simulation and testing are the state of the art.
- Beauty is in the eye of the beholder. One of programmers' most common questions is, "What makes something a good service? Should I surface my database through one monolithic service, or have a service per table?" Experiential guidance and tool support for service identification, factoring, etc will be increasingly critical.
- Most business processes integrate people and commercial data processing applications. Web service and Grid service models have converged. This makes calling just-in-time optimization, simulation, etc in business processes possible. Our industry is only at the beginning of understanding and exploiting these capabilities.
- There has been recent discussion of "situational applications," "ad hoc applications," "mash-ups" or "just-in-time" applications. These applications are targeted for a very specific, usually short duration and narrowly scoped problem. In some sense, the applications are the natural evolution of wikis, spreadsheets and Web services. Simple BPM concepts will increase the power and flexibility of these applications.

References

[1] http://www-306.ibm.com/software/integration/wbimodeler

[2] http://www.intalio.com/

[3] http://www-306.ibm.com/software/integration/wid/

[4] http://www.sap.com

[5] http://www-306.ibm.com/software/tivoli/features/it-serv-mgmt/

[6] http://www.itil.co.uk/

[7] *Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 1.* OASIS Standard, 9 March 2005. www.oasis-open.org

[8] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm

[9] http://msdn.microsoft.com/ws/2004/10/ws-management

[10] "On demand business process life cycle – Build reusable assets to transform an order processing system," http://www-128.ibm.com/developerworks/ibm/ws-odbp/index.html

[11] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series, 1995. ISBN 0-20-163361-2.

[12] Linda S. Sanford, Dave Taylor. *Let it Grow – Escaping the Commodity Trap.* Pearson Education, 2006. ISBN 0-13-148208-4.

[13] *Service Oriented Architecture Reference Model*, Working Draft 05, 03 May 2005. www.oasis-open.org.

[14] Service Component Architecture, http://www-128.ibm.com/developerworks/library/specification/ws-sca/

[15] www.oasis-open.org/committees/wsrf/

[16] www.ibm.com/developerworks/library/ws-coor/

[17] www.ibm.com/developerworks/webservices/library/ws-rm/

[18] www.ibm.com/developerworks/library/ws-secon/

[19] www.ibm.com/developerworks/library/ws-polfram/

[20] http://incubator.apache.org/tuscany/releasesscajava.html

[21] www.ibm.com/software/tivoli/products/prov-mgr/

[22] www.ibm.com/developerworks/webservices/library/ws-ramppaper.html

[23] http://www.bpmi.org/

[24] www.dmtf.org/standards/cim/

[25] www.ibm.com/developerworks/webservices/library/ws-bpelj/

[26] www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/

[27] Kevin Client et al., "Toward Converging Web Service Standards for Resources, Events, and Management." A joint whitepaper from Intel, IBM, HP and Microsoft.

[28] www.oasis-open.org/committees/wsrp

[29] www.w3.org/MarkUp/Forms/