

Intelligent Information Management Tools in a Service-Oriented Software Environment

Jens Pohl, Ph.D.

Executive Director, Collaborative Agent Design Research Center (CADRC)
California Polytechnic State University (Cal Poly)
San Luis Obispo, California, USA

Abstract

This paper draws attention to the increasing need for agile and adaptive software environments that are capable of supporting rapid re-planning during the execution of time-critical operations involving commercial end-to-end supply chain transaction sequences, as well as disaster response and military missions. It is argued that such environments are currently best served by information-centric software tools executing within a service-oriented paradigm. Service-oriented architecture (SOA) design concepts and principles are described, with a focus on the functions of the services management framework (SMF) and enterprise service bus (ESB) components. Differentiating between data-centric and information-centric services, it is suggested that only intelligent software services, particularly those that incorporate an internal representation of context in the form of an ontology and agents with reasoning capabilities, are able to effectively address the need for agile and adaptive planning, re-planning and decision-support tools.

The paper concludes with a description of the design components of a business process management (BPM) system operating within a SOA-based infrastructure, followed by a brief discussion of *Cloud* computing promises and potential user concerns.

Keywords: adaptive, agile, APEX, cloud computing, BPEL, business process execution language, BPM, business process management, choreographer, data-centric, enterprise service bus, ESB, information-centric, mediator, registry, services management framework, SMF, service-oriented architecture, SOA

Need for Adaptive Planning Tools

There is an increasing need in industry and government for planners and decision-makers to be able to rapidly re-plan during execution. Experience has shown that the best-laid plans will likely have to be changed during implementation. Operational environments are often impacted by events or combinations of factors that were either not foreseen during the planning stage or were thought to be unlikely to occur. In commerce, where *just in time inventories* have become an acknowledged cost-saving measure, suppliers and shippers are particularly vulnerable to the disruption of end-to-end supply chain sequences, such as inclement weather conditions, traffic congestion, accidents, equipment malfunction, and human error.

Military commanders, who often deal with extremely time-critical and human life endangering operations have learned from bitter experience that agile planning tools are essential for their ability to rapidly adapt to changing mission conditions. It can be argued that an information

management environment, with an agile planning capability of the type implied by the stated objectives of the *Adaptive Planning and Execution (APEX)*¹ process recently adopted by the U.S. military forces, requires both the ability to automatically interpret data in context and the flexibility to provide access to decision-support tools regardless of whether these are part of the same software application or another application.

This argument is based on the definition of *agility* as the ability to rapidly adapt to changing conditions, and has two implications. First, in a real world environment the operational data that enter a particular application may not adhere exactly to the specifications on which the design of the software was originally based. An agile software application will therefore need to have the ability to automatically interpret the incoming data within the appropriate context and make the necessary processing adjustments. Second, under such dynamic conditions it is likely that the user will have a need for tools that were not foreseen during the design of the application and are therefore not available. An agile software environment will therefore have to provide access to a wide range of tools, at least some of which may not be an integral component of the particular application that the operator is currently using. This suggests a system environment in which software tools can be seamlessly accessed across normal application domain boundaries. This is the objective of an information management environment that is based on the service-oriented concepts and principles described in this paper.

Information-Centric vs. Data-Centric

There are several reasons why computer software must increasingly incorporate more and more *intelligent* capabilities (Pohl 2005). Perhaps the most compelling of these reasons relates to the current data-processing bottleneck. Advancements in computer technology over the past several decades have made it possible to store vast amounts of data in electronic form. Based on past manual information handling practices and implicit acceptance of the principle that the interpretation of data into information and knowledge is the responsibility of the human operators of the computer-based data storage devices, emphasis was placed on storage efficiency rather than processing effectiveness. Typically, data file and database management methodologies focused on the storage, retrieval and manipulation of data transactions², rather than the *context* within which the collected data would later become useful in planning, monitoring, assessment, and decision-making tasks.

The term *information-centric* refers to the representation of information, as it is available to software modules, not to the way it is actually stored in a digital machine. This distinction

¹ Adaptive Planning and Execution Roadmap II, AO Review (Draft), Joint Chiefs of Staff, 8 February 2007.

² Most large organizations, including the Military, are currently forced to dedicate a significant portion of their operating budget, staff, project budgets, and time, on the piecemeal resolution of ad hoc problems and obstacles that are symptoms of an overloaded data-centric environment. Examples include: data bottlenecks and transmission delays resulting in aged data; temporary breakdown of data exchange interfaces; inability to quickly find critical data within a large distributed network of data-processing nodes; inability to interpret and analyze data within time constraints; and, determining the accuracy of the data that are readily available. This places the organization in a *reactive* mode, and forces it to expend many of its resources on solving the symptoms rather than the core problem. In contrast, an information-centric environment is capable of supporting: (1) the automatic filtering of data by placing data into an information context; (2) the automated reasoning of software agents as they monitor events and assist human planners and problem solvers in an intelligent collaborative decision-making environment; and, (3) autonomic computing capabilities.

between *representation* and *storage* is important, and relevant far beyond the realm of computers. When we write a note with a pencil on a sheet of paper, the content (i.e., meaning) of the note is unrelated to the storage device. A sheet of paper is designed to be a very efficient storage medium that can be easily stacked in sets of hundreds, filed in folders, folded, bound into volumes, and so on. As such, representation can exist at varying levels of abstraction. The lowest level of representation considered is *wrapped* data. Wrapped data consists of low-level data, for example a textual e-mail message that is placed inside some sort of an e-mail message object. While it could be argued that the e-mail message is thereby objectified it is clear that the only objectification resides in the shell that contains the data and not the e-mail content. The message is still in a data-centric form offering a limited opportunity for interpretation by software components.

A higher level of representation endeavors to describe aspects of a domain as collections of inter-related, constrained objects. This level of representation is commonly referred to as an information-centric ontology. At this level of representation context can begin to be captured and represented in a manner supportive of software-based reasoning. This level of representation (i.e., context) is an empowering design principle that allows software to undertake the interpretation of operational data changes within the context provided by the internal information model (i.e., ontology).

Even before the advent of the Internet and the widespread promulgation of SOA concepts it was considered good software design and engineering practice to build distributed software systems of loosely coupled modules that are able to collaborate by subscription to a shared information model. The principles and corresponding capabilities that enable these software modules to function as decoupled services include (Pohl 2007):

- An internal *information* model that provides a usable representation of the application domain in which the service is being offered. In other words, the *context* provided by the internal information model must be adequate for the software application (i.e., service) to perform as a useful adaptive set of tools in its area of expertise.
- The ability to *reason* about events within the context provided by the internal information model. These reasoning capabilities may extend beyond the ability to render application domain related services to the performance of self-monitoring maintenance and related operational efficiency tasks.
- Facilities that allow the service to *subscribe* to other internal services and understand the nature and capabilities of these resources based on its internal information model³.
- The ability of a service to understand the notion of *intent* (i.e., goals and objectives) and undertake self-activated tasks to satisfy its intent. Within the current state-of-the-art this capability is largely limited by the degree of context that is provided by the internal information model.

³ This must be considered a minimum system capability. The full implementation of a web services environment should include facilities that allow a service to *discover* other external services and understand the nature and capabilities of these external services.

Additional capabilities that are not yet able to be realized in production systems due to technical limitations, but have been demonstrated in the laboratory environment, include: the ability of a service to *learn* through the acquisition and merging of information fragments obtained from external sources with its own internal information model (i.e., dynamically *extensible* information models); extension of the internal information model to include the internal operational domain of the software application itself and the role of the service within the external environment; and, the ability of a service to increase its capabilities by either *generating* new tools (e.g., creating new agents or cloning existing agents) or automatically *searching* for external assistance.

Service-Oriented Architecture (SOA)

The notion of *service-oriented* is ubiquitous. Everywhere we see countless examples of tasks being performed by a combination of services, which are able to interoperate in a manner that results in the achievement of a desired objective. Typically, each of these services is not only *reusable* but also sufficiently *decoupled* from the final objective to be useful for the performance of several somewhat similar tasks that may lead to quite different results. For example, a common knife can be used in the kitchen for preparing vegetables, or for peeling an orange, or for physical combat, or as a makeshift screwdriver. In each case the service provided by the knife is only one of the services that are required to complete the task. Clearly, the ability to design and implement a complex process through the application of many specialized services in a particular sequence has been responsible for most of mankind's achievements in the physical world. The key to the success of this approach is the *interface*, which allows each service to be utilized in a manner that ensures that the end-product of one service becomes the starting point of another service.

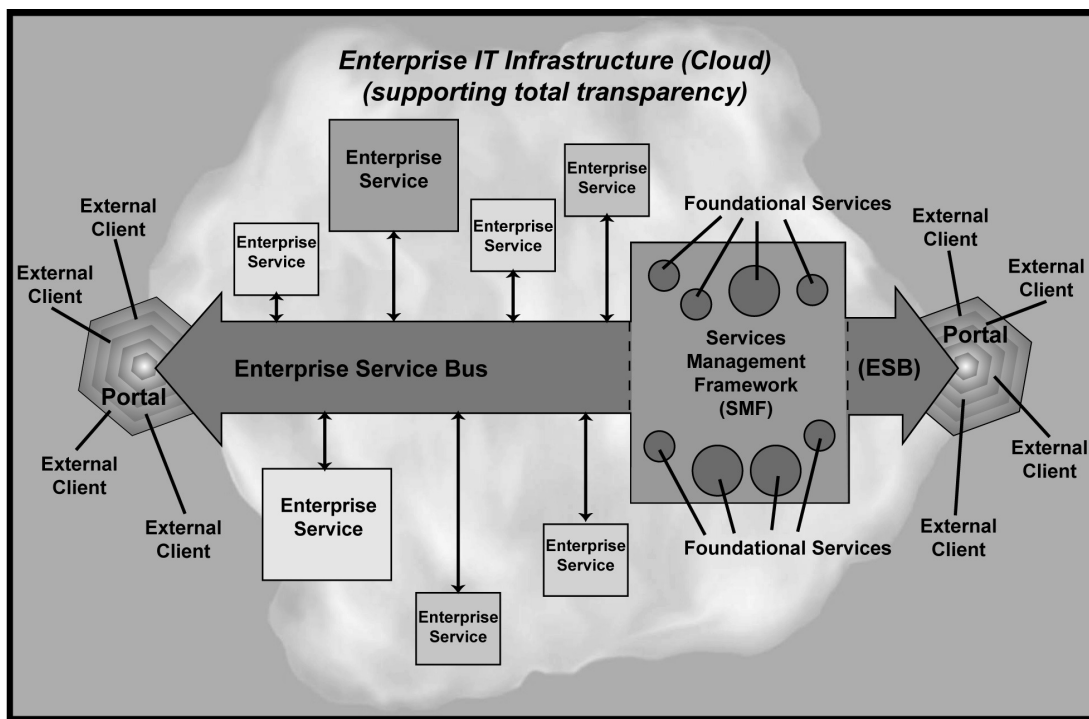


Figure 1: Principal components of a conceptual SOA implementation

In the software domain these same concepts have gradually led to the adoption of Service-Oriented Architecture (SOA) principles. While SOA is by no means a new concept in the software industry it was not until Web services became available that these concepts could be readily implemented (Erl 2005). In the broadest sense SOA is a software framework for computational resources to provide services to customers, such as other services or users. The Organization for the Advancement of Structured Information (OASIS)⁴ defines SOA as a “... *paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains*” and “...*provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects with measurable preconditions and expectations*”. This definition underscores the fundamental intent that is embodied in the SOA paradigm, namely *flexibility*. To be as flexible as possible a SOA environment is highly modular, platform independent, compliant with standards, and incorporates mechanisms for identifying, categorizing, provisioning, delivering, and monitoring services.

The principal components of a conceptual SOA implementation scheme (Figure 1) include a Services Management Framework (SMF), various kinds of foundational services that allow the SMF to perform its management functions, one or more portals to external clients, and the enterprise services that facilitate the ability of the user community to perform its operational tasks.

Services Management Framework (SMF): A Services Management Framework (SMF) is essentially a SOA-based software infrastructure that utilizes tools to manage the exchange of messages among enterprise services. The messages may contain requests for services, data, the results of services performed, or any combination of these. The tools are often referred to as foundational services because they are vital to the ability of the SMF to perform its management functions, even though they are largely hidden from the user community. The SMF must be capable of:

- Undertaking any transformation, orchestration, coordination, and security actions necessary for the effective exchange of the message.
- Maintaining a loosely coupled environment in which neither the service requesters nor the service providers need to communicate directly with each other; - or even have knowledge of each other.

A SMF may accomplish some of its functions through an Enterprise Service Bus (ESB), or it may be implemented entirely as an ESB.

Enterprise Service Bus (ESB): The concept of an Enterprise Service Bus (ESB) greatly facilitates a SOA implementation by providing specifications for the coherent management of services. The ESB provides the communication bridge that manages the exchange of messages among services, although the services do not necessarily know anything about each other. According to Erl (2005) ESB specifications typically define the following kinds of message management capabilities:

⁴ OASIS is an international organization that produces standards. It was formed in 1993 under the name of SGML Open and changed its name to OASIS in 1998 in response to the changing focus from SGML (Standard Generalized Markup Language) to XML (Extensible Markup Language) related standards.

- *Routing*: The ability to channel a service request to a particular service provider based on some routing criteria (e.g., static or deterministic, content-based, policy-based, rule-based).
- *Protocol Transformation*: The ability to seamlessly transform the sender's message protocol to the receiver's message protocol.
- *Message Transformation*: The ability to convert the structure and format of a message to match the requirements of the receiver.
- *Message Enhancement*: The ability to modify or add to a sender's message to match the content expectations of the receiver.
- *Service Mapping*: The ability to translate a logical business service request into the corresponding physical implementation by providing the location and binding information of the service provider.
- *Message Processing*: The ability to accept a service request and ensure delivery of either the message of a service provider or an error message back to the sender. Requires a queuing capability to prevent the loss of messages.
- *Process Choreography and Orchestration*: The ability to manage multiple services to coordinate a single business service request (i.e., choreograph), including the implementation (i.e., orchestrate). An ESB may utilize a Business Process Execution Language (BPEL) to facilitate the choreographing.
- *Transaction Management*: The ability to manage a service request that involves multiple service providers, so that each service provider can process its portion of the request without regard to the other parts of the request.
- *Access Control and Security*: The ability to provide some level of access control to protect enterprise services from unauthorized messages.

There are quite a number of commercial off-the-shelf (COTS) ESB implementations that satisfy these specifications to varying degrees. A full ESB implementation would include four distinct components (Figure 2): Mediator; Service Registry; Choreographer; and, Rules Engine. The Mediator serves as the entry point for all messages and has by far the largest number of message management responsibilities. It is responsible for routing, communication, message transformation, message enhancement, protocol transformation, message processing, error handling, service orchestration, transaction management, and access control (security).

The Service Registry provides the service mapping information (i.e., the location and binding of each service) to the Mediator. The Choreographer is responsible for the coordination of complex business processes that require the participation of multiple service providers. In some ESB implementations the Choreographer may also serve as an entry point to the ESB. In that case it assumes the additional responsibilities of message processing, transaction management, and access control (security). The Rules Engine provides the logic that is required for the routing, transformation and enhancement of messages. Clearly, the presence of such an engine in combination with an inferencing capability provides a great deal of scope for adding higher levels of intelligence to an ESB implementation.

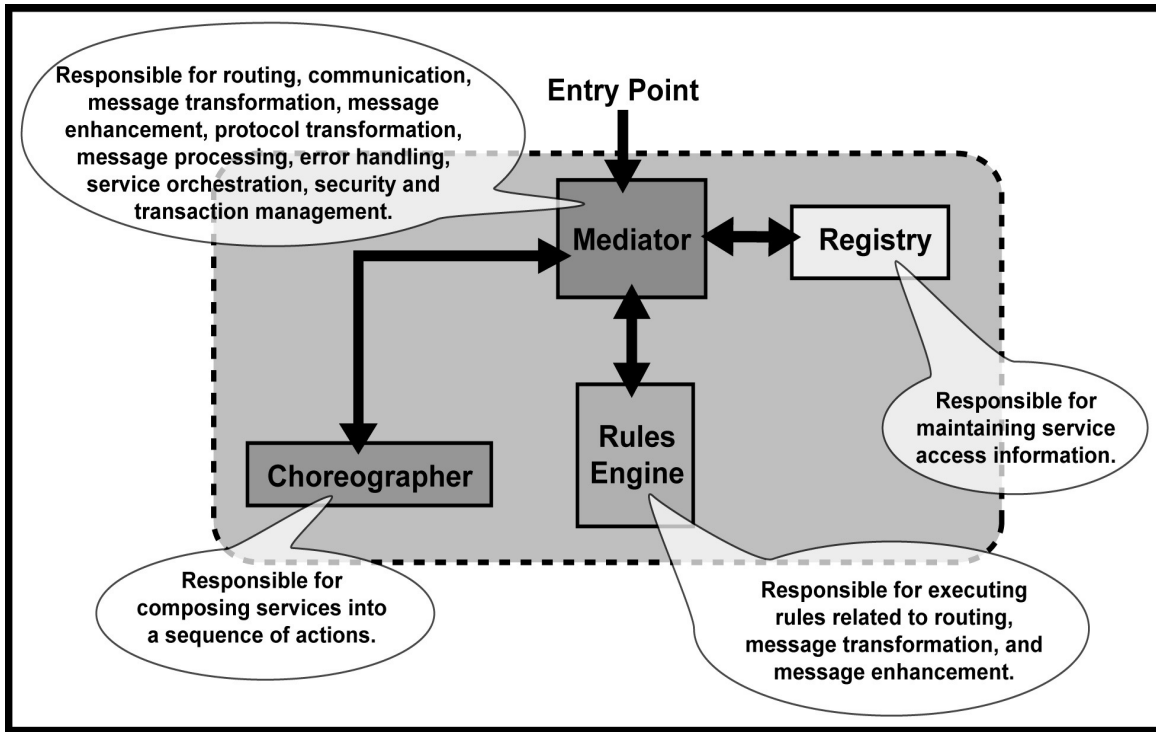


Figure 2: Primary ESB components

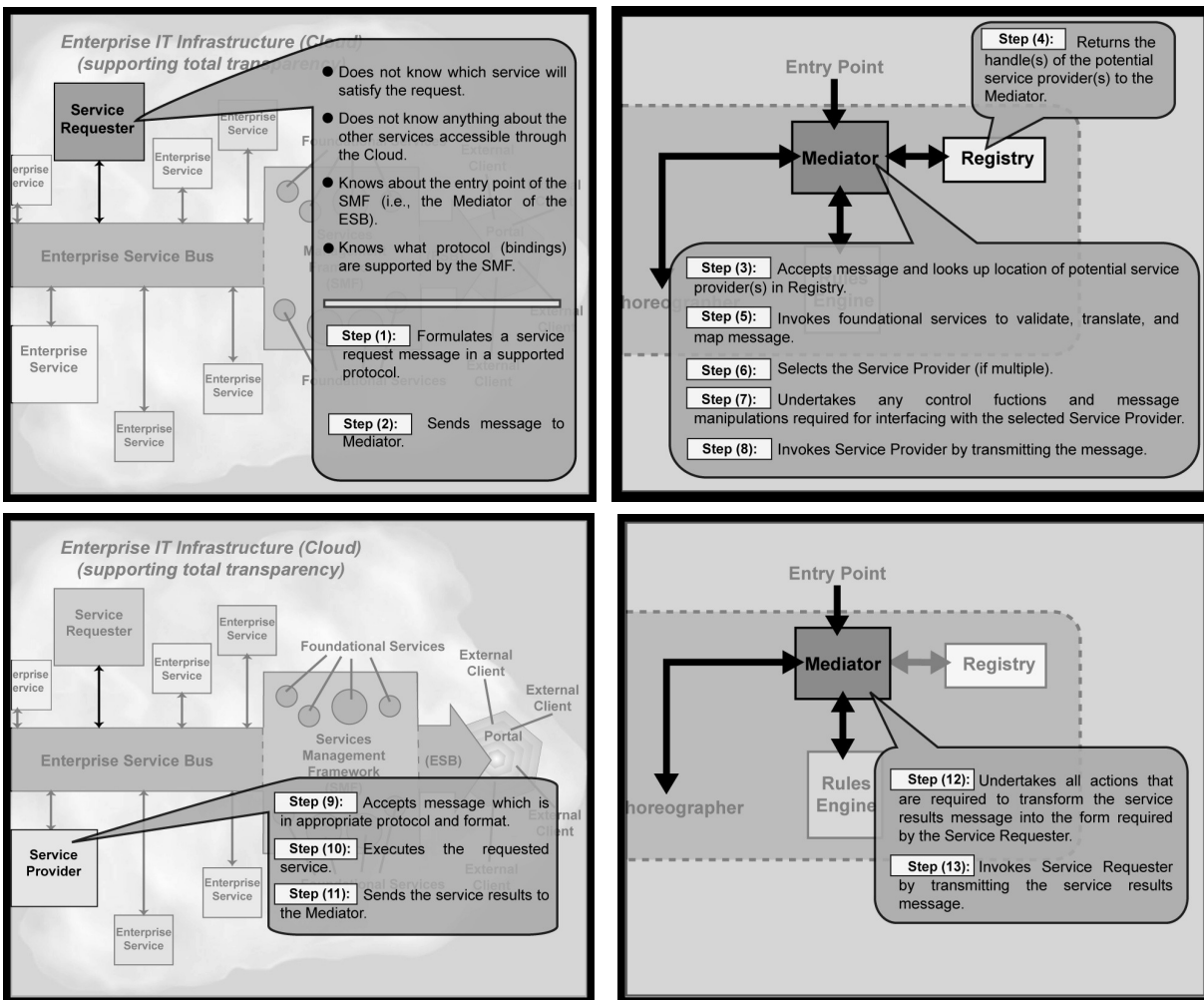
Typical Service Requester and Service Provider Scenario

The following sequence of conceptual steps that must be taken by the SMF to support a SOA system environment is not inclusive of every variance that might occur. It is intended to provide a brief description of the principal interactions involved (Figure 3).

While the Service Requester knows that the Mediator is the entry point of the ESB component of the SMF and what bindings (i.e., protocols) are supported by the Mediator, it does not know which Service Provider will satisfy the request because it knows nothing about any of the other enterprise services that are accessible through the Mediator. Therefore, the conceptual SOA-based infrastructure shown in Figure 1 is often referred to as a *Cloud*.

The Mediator is clearly in control and calls upon the other primary components of the ESB if and when it requires their services. It requests the *handle* (i.e., location and mappings) of the potential Service Providers from the Service Registry. If there are multiple Service Provider candidates then it will have to select one of these in Step (6) to provide the requested service. The Mediator will invoke any of the foundational services in the SMF to validate (i.e., access control), translate, transform, enhance, and route the message to the selected Service Provider. The latter is able to accept the message because it is now in a data exchange format that the Service Provider supports.

Similar transformation and mapping actions are taken by the Mediator after it receives the reply message from the Service Provider, so that it complies with the data exchange format supported by the Service Requester. On receiving the response message the Service Requester does not know which service responded to the request, nor did it have to deal with any of the data exchange requirements of the Service Provider.

Figure 3: Conceptual *Cloud* operations

Business Process Management (BPM)

From a general point of view, Business Process management (BPM) is the orchestration of activities between people and systems. More specifically, BPM is a method for actively defining, executing, monitoring, analyzing, and subsequently refining manual or automated business processes. In other words, a business process is essentially a sequence of related, structured activities (i.e., a workflow) that is intended to achieve an objective. Such workflows can include interactions between human users, software applications or services, or a combination of both.

In a SOA-based information management environment this orchestration is most commonly performed by the Choreographer component of the ESB (Figure 2). Based on SOA principles, a sound BPM design will decompose a complex business process into smaller, more manageable elements that comply with common standards and reuse existing solutions.

The BPM design solution should be based on an analysis of the problem within both its local and global contexts (Figure 4). It must describe and support the local business process requirements

as its primary objective and yet seamlessly integrate this micro perspective into a global view. Successful integration of these two perspectives will require an understanding of external interactions and the compliance parameters that apply to interprocess protocols. The principal components of a BPM design solution include a Business Process Execution Language (BPEL) engine, a graphical modeling tool, business user and system administration interfaces, internal and external system interactions, and persistence (Figure 5).

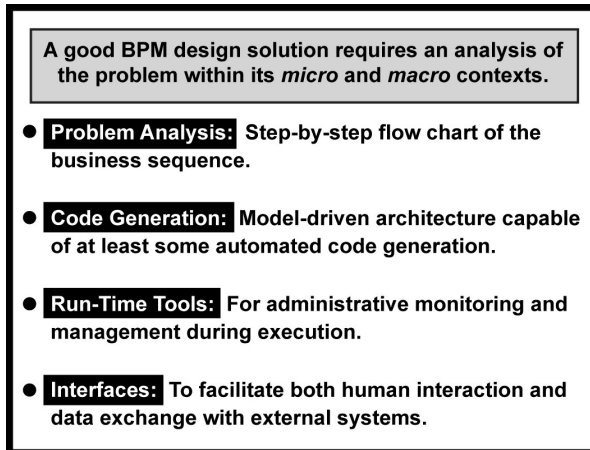


Figure 4: BPM design requirements

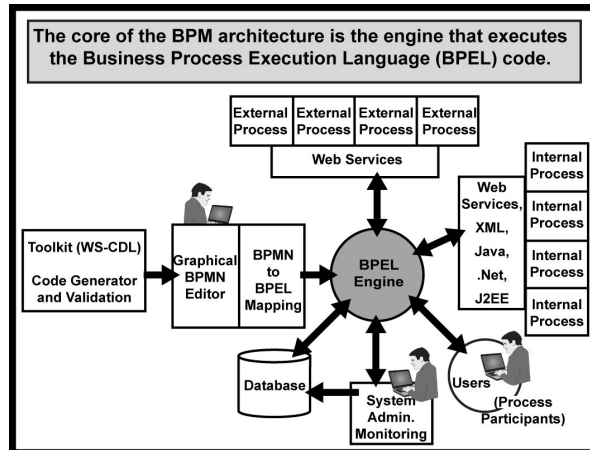


Figure 5: BPM design components

BPEL Engine: BPEL, which is the preferred process language, is normally XML-based⁵ and event driven. The BPEL Engine is responsible for detecting events, executing the appropriate next step in the business process sequence, and managing outbound message calls.

Graphical Editor: Effective communication during design is greatly facilitated by a standard system of notation that is known to all parties involved in the design process, and a graphical tool that allows design solutions to be represented in the form of diagrams. Both the Business Process Modeling Notation (BPMN)⁶ and the Unified Modeling Language (UML)⁷ Activity Diagram provide the necessary capabilities. However, BPMN is normally preferred because it incorporates BPEL mapping capabilities and is considered to be the more expressive notation. Whichever graphical modeling tool is chosen it should be capable of representing the different views of the process that are desired by the business user and the technical user. The business user is interested in the overall flow of the process, while the technical user is interested in the more detailed behavioral characteristics of each step.

⁵ The Extensible Markup Language (XML) is a general purpose specification that allows the content of a document to be defined separately from the formatting of the document.

⁶ BPMN provides a graphical representation for describing a business process in the form of a workflow diagram. It was developed by the Business Process Management Initiative (BPMI) and is now maintained by the Object Management Group following the merging of these two organizations in 2005.

⁷ The Unified Modeling Language (UML) provides a standard notation for modeling systems and context based on object-oriented concepts and principles (Booch G., J. Rumbaugh and I. Jacobson (1999); 'The Unified Modeling Language User Guide'; Addison-Wesley, New York, New York.)

User-interfaces: Typically, separate user-interfaces are required for the business user who has a functional role in the business process and may from time to time be required to interact with the BPEL Engine, and the system administrator who may be monitoring the task flow for reactive or proactive system maintenance reasons. The business users essentially require a *worklist*⁸ interface that allows them to contribute manual tasks to the automated BPM process. This should be a user-friendly, role-based interface with process status reports and error correction capabilities. The system administrators require a user-interface that allows them to perform a host of management tasks including: defining a process (i.e., find, activate, deactivate, remove, or add); controlling the execution of processes known to the BPEL Engine and worklist tasks or activities (i.e., find, suspend, resume, or terminate); managing user roles (i.e., add, modify, or remove users and roles from applications); and, configuring application connections.

Both the business and system administration user-interfaces must incorporate security measures to prevent unauthorized access and ensure that only authorized role-based actions can be executed.

System interactions: A business process is likely to involve both internal and external system interactions. In general terms these interactions may be characterized as four distinct modes: process receives a message from another system; process receives a message and sends a response; process sends a message to another system; and, process sends a message and waits for a response. External interactions are typically choreographed as web services, with a wide variety of system interfaces being supported through a generic adapter facility. This means that the BPEL Engine must include a web services listener capable of accepting an inbound message (e.g., in SOAP⁹ format), insert it into the runtime engine, obtain a response (if any), and send out the response as a SOAP message. Internal interactions are typically either client-server interfaces to other systems executing on the enterprise network or inline code snippets.

Persistence: To survive the inevitable need to restart the BPEL Engine the current process state must be stored in a database. Tables in the database typically include: process definition; process execution state; message content and identification code; process variables; activity execution state; and, worklist task execution state.

While BPM and SOA concepts are closely connected, they are certainly not synonymous. Described more precisely, a SOA-based system environment provides the enabling infrastructure for BPM by separating the functional execution of the business process from its technical implementation.

In Conclusion: Cloud Computing

The concept of *Cloud* computing as a massively scalable, user-transparent computing resource that can be readily accessed by multiple users across a global network is indeed a compellingly

⁸ A BPM *worklist* allows a manual task to be assigned to a user and track the progress of that task. In this way the human user can be the source of events that trigger the BPEL Engine.

⁹ The Simple Object Access Protocol (SOAP) is a protocol specification for the exchange of data among web services. It utilizes XML as its message format and depends on other protocols, such as Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP) for transmitting the message.

attractive proposition. Combined with the SOA design and implementation principles described above, the *Cloud* not only takes care of all of the intricate technical interoperability and data exchange incompatibility issues that have plagued computer users in the past, but also provides essentially ubiquitous access to powerful and seamlessly integrated computer-based capabilities as services. Naturally, multiple *Clouds* can be linked in a manner that is quite similar to the way services are registered within a particular *Cloud*. In such an environment neither the service requester nor the service provider needs to know, or even care, where the request originated and where it was processed, even if the request for services had to traverse several *Clouds* before the necessary service provider could be found.

It is of interest to note that this view of computing as a service is not new. During the 1960s and 1970s time-share computer systems, which linked multiple remote user terminals through modems to a central computing facility, provided a similar computing service. However, there were some major differences. First, access and data exchange was strictly confined to a single computer center and in most cases to the particular application that the user was authorized to use. Second, very little of the underlying computing environment was transparent to the user. Third, the users were almost as rigidly tied to their access terminal location as the service provider was tied to the location of its computer center. The time-share concept became obsolete as soon as the advent of microcomputers brought the computing power to the user.

We might ask: *Was it a desire by the computer users to have complete control over their computing resources or convenience that led to the preference of ownership over service?* While *Cloud* computing promises to overcome the inconvenience, immobility, and lack of interoperability constraints of the time-share service environment, it does pose other problems that will need to be overcome. Chief among these is the issue of data security. Will organizations be willing to entrust their proprietary data to a remote *Cloud* environment over which, in reality, they have little control? They must trust the *Cloud* service provider to not only maintain adequate internal security, but to resist even the most sophisticated and continuously changing external intrusion attempts. Also, as Robert Lucky (2009) recently wrote "... once all your petabytes of data are out there in the *Cloud*, can you ever get them back?

Finally, there is the question of user autonomy and control. Are current and will future privacy laws be sufficient to protect the user from a plethora of potential consumer abuses, for example, the automated collection of data about a user's activities in the *Cloud* without the need to actually trespass the data repositories themselves. Such data are already being collected by Internet service providers and utilized to determine collective and individual preferences for advertising and directed marketing purposes. Perhaps users will not be greatly concerned about the potential privacy infringements of such activities, and in the end the convenience and inexpensiveness of *Cloud* computing may become the deciding factors.

References

- Burlton R. (2001); 'Business Process Management: Profiling from Process'; SAMS, Indianapolis, Indiana.
- Chang J. (2005); 'Business Process Management Systems'; Auerbach Publications, Auerbach/Vogtland, Germany.

- Erl T. (2005); 'Service-Oriented Architecture (SOA): Concepts, Technology, and Design'; Prentice Hall Service-Oriented Computing Series, Prentice Hall, Englewood Cliffs, New Jersey.
- Havey M. (2005); 'Essential Business Process Modeling'; O'Reilly, Sebastopol, California.
- Jeston J. and J. Nelis (2006); 'Business Process Management: Practical Guidelines to Successful Implementations'; Butterworth Hein Elsevier, United Kingdom.
- Lucky R. (2009); 'Cloud Computing'; (under *Reflections*) IEEE Spectrum, Institute of Electrical and Electronics Engineers, 46(5), May (pp. 27).
- Pohl J. (2005); 'Intelligent Software Systems in Historical Context'; in Jain L. and G. Wren (eds.); 'Decision Support Systems in Agent-Based Intelligent Environments'; Knowledge-Based Intelligent Engineering Systems Series, Advanced Knowledge International (AKI), Sydney, Australia.
- Pohl J. (2007); 'Knowledge Management Enterprise Services (KMES): Concepts and Implementation Principles'; InterSymp-2007, Proceedings Focus Symposium on Representation of Context in Software, Baden-Baden July 31, Germany.
- Taylor D. and H. Assal (2008); 'Using BPM as an Interoperability Platform'; C2 Journal, Special Issue on Modeling and Simulation, CCRP, Washington, DC, Fall.