RC23109 (W0402-077) February 17, 2004
Computer Science

IBM Research Report

Towards an On Demand Services Oriented Architecture: Integrating Business Processes, Application and IT Infrastructure Enablement in Enterprise Computing Environments

Catherine H. Crawford
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Paul Bate, Luba Cherbakov
IBM Global Services
Bethesda, MD 20817

Kerrie Holley
IBM Global Services
San Francisco, CA 94105

Charles Tsocanos
IBM Global Services
Paramus, NJ 07653

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Towards an On Demand Services Oriented Architecture:

Integrating Business Processes, Application and IT Infrastructure Enablement in Enterprise Computing Environments

Catherine H. Crawford[1], IBM Research Division and
Paul Bate, Luba Cherbakov, Kerrie Holley &
Charles Tsocanos, IBM Global Services

Abstract:

The success of an on demand e-business is critically linked to the realization that **business process, application, and IT infrastructure integration** must merge into a comprehensive and cohesive architecture, where business process transformation drives services-based application enablement and on demand enterprise computing. This type of architecture is often described as a Services Oriented Architecture (SOA) and is seen as the prerequisite accelerator for the on demand operating environment.

The traditional focus of the SOA has been on dynamic re-configuration of services from a defined business process, and on writing components as standards, based on Web Services and, more recently, grid services. Current descriptions of SOA are less focused on overall IT infrastructure enablement both from a business policy perspective and within the context of service-based application development. For example, how are security, availability, or performance policies derived from business process policies? How do such policies relate to provisioning, scheduling or other resource virtualization technologies?

Minimal information has been published on how we can improve or enhance business processes in light of emerging technologies in Web Services, autonomic computing, grid, or utility based computing. Deficiencies in current SOA descriptions and the reciprocal problem of a lack of clear articulation regarding how the enabling technologies can deliver SOA implementations, are an impediment to the development of an integrated SOA solution and offering strategy. Even if we clarify the mapping between SOA descriptions, autonomic functions, and business policies/processes, we still face the non-trivial task of providing application and middleware tooling. Such tooling should be able to support the creation and lifetime management of these now flexible IT environments.

In this paper, we will extend the current ideas of SOA to include a more comprehensive integration of business process transformation and the enabling technologies of services-based application development and policy-based IT management. We call this extension the *On Demand* SOA. We will develop these concepts using an existing scenario: a Financial Services Sector "Life Changes" business process scenario, which involves distributed and disjoint transactions as well as stateless High Performance Computing (HPC) applications. This paper will conclude with a strategy for developing *On Demand* SOA service offerings.

[1] *Author to whom correspondence should be addressed: IBM TJ Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532*

Table of Contents

1 Introduction

Over the last four decades, IT architectures and development approaches have dealt with increasing levels of IT complexity resulting from mergers and acquisitions and myriad integration challenges. The level of complexity continues to increase, and traditional architectures seem to be reaching the limits of their abilities to deal with the challenge. At the same time, traditional needs of IT organizations persist: quick response to new business requirements, skills simplification, cycle times and cost reduction. Enterprises need to move from manual transactions with suppliers towards automated transactions, provide flexible interactions with partners with minimal process or IT impact, absorb and integrate new business partners and new customer sets. As an industry, we have gone through multiple computing architectures designed to allow fully distributed processing, several programming languages designed to run on any platform and greatly reduce implementation schedules, and a myriad of connectivity products designed to allow better and faster integration of applications. However, the complete solution continues to elude us.

For some time now, the existence of Web Services technologies has stimulated the discussion of Services Oriented Architectures (SOAs). The discussion isn't a new one; the concept has been developing for more than a decade, since Common Object Request Broker Architecture (CORBA) extended the promise of integrating applications on disparate heterogeneous platforms. Problems integrating those applications have always arisen, often because so many different (and non-CORBA-compliant) object models became popular. As a result, many architects and engineers became so bogged down in solving technology problems that the promise of developing a more robust architecture for simple, fast, and secure integration of systems and applications was lost. The problems, however, persist, and become more complex every year. We find that "point solutions" won't address the challenge. The problem, in many cases, is the lack of a consistent architectural framework within which applications can be rapidly developed, integrated, and reused. This framework allows the assembly of components and services for the rapid, and even dynamic, delivery of solutions. Many papers have been written about why particular technologies such as Web Services, autonomic computing, utility computing or grid technologies and standards are good, but what is needed is an architectural view, unconstrained by technology.

As the name suggests SOA is just that, an architecture. Within a business environment, a pure architectural definition of an SOA might be something like, "an application architecture within which all functions are well-defined as independent services with invokable interfaces that can be called in prescribed sequences to form business processes". If we examine the terms, *services, independent, and invokable*, we find the following:

- All functions are defined as *services*. This includes business functions, business transactions composed of lower-level functions, and system service functions.

- All services are *independent*. They operate as "black boxes." External components neither know nor care how they perform their functions, merely that they return the expected results.

- In the most general sense, the interfaces are *invokable*; that is, at an architectural level, it is irrelevant whether they are local (within the system) or remote (external to the immediate system), what interconnect scheme or protocol is used to effect the invocation, or what infrastructure components are required to make the connection. The service may be within the same application or in a different address space within an asymmetric multiprocessor, on a completely different system within the corporate Intranet, or within an application in a partner's system used in a business-to-business configuration.

Furthermore, SOA provides a distributed computing programming model where software resources (applications, methods, or functions) are viewed as services. In order for a resource to be accessible as a service within a distributed computing environment, the service must expose an addressable interface within a central registry. This type of component-based architecture allows for application writers to construct service flows corresponding to business processes that are dynamically reconfigurable for a high degree of code re-use. Simply put, SOA defines an application enablement paradigm in which software can be more directly aligned with and fundamentally driven by business processes.

An on demand SOA is a distributed computing model infused and enabled with the basic building blocks of Web Services, autonomic computing, grid services and utility computing. Taken separately, Web Services, SOA, autonomic computing, grid services and utility computing provide significant benefits, but the integration of these technologies as the building blocks for an SOA provide the foundation for on demand computing.

We illustrate our definition of an on demand SOA with a simple example from business-to-business electronic commerce – a purchase order request, or POR. Our buyer is a large manufacturing company – Acme, Inc. Our supplier is a large stationary company – Pens R Us. We assume that a contract already exists between the two parties and that Acme, Inc. wants to use an electronic POR to buy 500 reams of paper from Pens R Us. The first step in understanding how an SOA could be used to implement this or any process is to see how such a process could be broken down into services at an appropriate level of abstraction. The level at which an architect chooses to abstract different pieces of the process will determine the services make up of the resulting software application. For instance, at one extreme we may consider the POR as a single service, while at the other extreme, we may choose a level of granularity so fine that the POR might be a meta service constructed from hundreds of services. The choice of granularity will be a balance between meeting specific Quality of Service (QoS) characteristics, ubiquitous service re-use, and reducing complexity for meta service composition. For the POR, the supplier may choose to view the process as the following components, or steps:

1.) Supplier authenticates the purchaser.
2.) Supplier looks up the buyer contract based on purchaser ID.
3.) Purchaser browses the product catalog with negotiated prices from the contract.
4.) Purchaser adds items to his/her shopping cart.
5.) Purchaser checks out.
6.) Purchaser provides payment description and delivery information.
7.) Order information is sent to the fulfillment department.
8.) Confirmation of order with expected delivery date is sent to the purchaser.

From such a process description, we can further list the software or application services that are required:

1.) Login
2.) Contract lookup
3.) Catalog browsing with shopping cart and checkout
4.) POR data creation (from login ID, contract ID, shopping cart data, and other information supplied by checkout)
5.) Information delivery to fulfillment process
6.) Message to purchaser to confirm order

Other supply side business processes will likely use the above list of services (e.g., login, contract lookup, catalog browsing, messaging). The services are abstracted at a sufficiently high enough level so that the entire process or meta service could be constructed from only six service components.

The enablement of SOA with open standards (e.g., Simple Object Access Protocol (SOAP), eXtensible Markup Language (XML), Web Services Description Language (WSDL), Open Grid Services Architecture (OGSA), etc.), offers the ability to meet the promises and fulfill the value propositions for SOA implementations. These Open standards allow for the service to be decoupled from the underlying IT infrastructure. As long as vendor support for the standard exists across resources, the application writer need not be concerned with where the service will run only how to assemble flows between services. In fact, by using a services registry, application writers do not need to know where a software resource physically resides when the application is written. Additionally, dynamic service lookup means that service consumers need not be concerned with where underlying software resources exist on distributed, heterogeneous systems.

In isolation, Web Services, autonomic computing, grid services, and SOA are only parts of the answer. Most of today's production systems that use Web Services, autonomic computing or grid services are not SOAs. Many of the existing Web Services implementations are simple remote procedure calls or point-to-point messaging via SOAP. To achieve the promised benefits of SOA, one needs all of these technologies working together in a meaningful way to create what we define as the On Demand SOA. John Hagel has asserted that, "Over time, distributed service architectures enabled by Web Services technologies have the potential to become the dominant technology

architecture for all business activities" [HAGEL2002]. When we include autonomic computing, grid services, and Web Services, this vision becomes a reality, and we have the On Demand SOA.

1.1 Benefits of SOA

In order to clarify the benefits of SOA we need to understand more about the fundamental driver behind the SOA design requirements – business process transformation, which consists of horizontal integration of information assets with business activity workflows. In some cases, business processes can be completely sourced, based on their commoditization. For example, partner firms exist to manage internally focused business operations, such as HR. Automatic Data Processing, Inc. (ADP) is one example of a firm that is trusted to administer payroll for many companies. Similarly, IBM can begin to position certain on demand services that can source common business operations. For business processes to be responsive, focused, variable, and resilient, a high degree of automation and adaptability must be enabled within the underlying IT application infrastructure.

Business process transformation requires a highly dynamic operating environment. This operating environment consists of the system environment and the application environment. The system environment refers to all enabling technologies (formerly known as the system and network infrastructure) to provision resources based on workflow, information, and other application requirements. The application environment consists of the customized workflows, transaction flows, and use cases that can deliver modular services. SOA addresses the requirements of the application environment while enabling applications to take advantage of a virtualized IT environment.

The historical business problems facing IT organizations remain, namely corporate management pushing for better IT utilization, greater ROI, continued integration of historically separate systems, and faster implementation of new systems. Today's IT environment has only increased in complexity, making these problems even more difficult to solve. Legacy systems must be reused rather than replaced, because constrained budgets make replacement cost-prohibitive. We find that cheap, ubiquitous, and easy access to the Internet has created the possibility of entire new business models, which must be evaluated for their potential. Growth by merger and acquisition has become standard fare, so entire IT organizations, applications, and infrastructures must be integrated and absorbed. In an environment of this complexity, point solutions merely exacerbate the problem. Systems must be developed where heterogeneity is fundamental to the environment, because they must accommodate an endless variety of hardware, operating systems, middleware, languages, and data stores. The cumulative effect of decades of growth and evolution has produced the complexity that now tortures us.

The On Demand SOA proposes to alleviate these pain points with a design philosophy centered on abstracting software resources (applications and methods therein) from the system resources. More specifically, the addition of open standards and the corresponding emerging technologies within SOA has resulted in an application enablement design that addresses the shortcomings above and allows for:

- **Infrastructure virtualization** that can result in significant reductions in total cost of ownership (TCO) from reduction in inflexibility and brittleness of an infrastructure by allowing a distinct separation of business logic from systems (underlying resources)

- **Development framework** that more quickly and easily adjusts to changing business processes and requirements by standardizing virtualization of software resources and makes these software resources or services readily available on a network (also reducing brittleness)

- **Operating environment** consisting of supporting management service flows that are capable of controlling the infrastructure resource management, including workload management, scheduling, and provisioning, to reduce system complexity based upon business process driven policies (e.g., Web Service Level Agreement, or WSLA [WSLA])

The On Demand SOA can evolve based on existing system investments rather than requiring a full-scale system rewrite. Organizations that focus their efforts around the creation of services, leverage open standards, use Web Services, grid technologies and standards, and autonomic computing will realize several of the following benefits:

- **Leverage existing assets.** A business service can be constructed as an aggregation of existing components, using a suitable SOA framework and made available to the enterprise. Using this new service only requires knowing its interface and name. The service's internals are hidden from the outside world, as well as the complexities of the data flow through the components that make up the service. This component anonymity lets organizations leverage current investments, building services from a conglomeration of components built on different machines, running different operating systems, developed in different programming languages. Legacy systems can be encapsulated and accessed via Web Service interfaces.

- **Infrastructure, a commodity.** Infrastructure development and deployment will become more consistent across all the different enterprise applications. Existing components, newly developed components, and components purchased from vendors can be consolidated within a well-defined SOA framework. Such an aggregation of components will be deployed as services on the existing infrastructure, resulting in the underlying infrastructure considered more as a commodity element.

- **Faster time-to-market.** Organizational Web Services libraries will become the core asset for enterprises adapting the SOA framework. Building and deploying services with these Web Services libraries will reduce the time-to-market dramatically as new initiatives reuse existing services and components, thus reducing design, development, testing and deployment time.

- **Reduced cost.** As business demands evolve and new requirements are introduced, the cost of enhancing and creating new services by adapting the SOA framework and the services library for both existing and new applications is greatly reduced. The IT infrastructure is further optimized using grid services and/or utility computing.

- **Risk mitigation.** Reusing existing components reduces the risk of introducing new failures into the process of enhancing or creating new business services. As mentioned earlier, there is a reduced risk in the maintenance and management of the infrastructure supporting the services as well.

- **Continuous Business Process Improvement.** An SOA allows a clear representation of process flows identified by the order of the components used in a particular business service. This provides the business users with an ideal environment for monitoring business operations. Process modeling is reflected in the business service. Process manipulation is achieved by reorganizing the pieces in a pattern (components that constitute a business service). This would further allow for changing the process flows while monitoring the effects, and facilitating continuous improvement.

- **Process-centric Architecture.** The existing architecture models and practices tend to be program-centric. Often, process knowledge is spread among components. The application components are much like a black box with no granularity available outside it. Reuse requires copying code, incorporating shared libraries, or inheriting objects. In a process-centric architecture, the process is decomposed into a series of steps, each representing a business service. In effect, each service or component functions as a sub-application. These sub-applications are chained together to create a process flow that satisfies the business need. This granularity lets processes leverage and reuse each sub-application throughout the organization.

1.2 Additional On Demand SOA Requirements

As mentioned in previous sections, maximum benefit from an SOA implementation in an on demand environment occurs when the underlying enterprise computing infrastructure is virtualized and enabled for policy-based system management, where the IT policies have been derived from the corresponding business processes. Current SOA focus is on process definition and application enablement via Web Services or grid technologies and standards. The link between business process policy and IT policy via application enablement has yet to be defined. For instance, in the POR example, the login service could be re-used across many business processes. However, different processes will have different policies concerning user class of service, i.e., type of authentication required. The virtualized IT infrastructure must be capable of ensuring different types of security based upon the business policy with respect to the type of technology used to enable the application, i.e., Web Services. Any solution development framework for On Demand SOA will focus on an integrated approach among the following: the on demand business

process transformation driving application enablement, corresponding IT policy and governance, and Service Level Agreement (SLA) based system management of virtualized resources.

The starting point for On Demand SOA is business process transformation. At the core of business process transformation is the policy and governance about how different parts of the process are integrated. This business process policy will be used to derive IT policies, QoS, SLA 's and the like. All of these terms are related to the business process, but each has a different meaning.

- **Policy** – a high level statement of how things will be managed or organized, including management goals, objectives, beliefs and responsibilities. Policies are normally defined at an overall strategy level and can be related to a specific area, for example, security and management policies. In many instances, policies reflect the law and givens to which the policies must adhere. This is especially true in the case of security and privacy policies.

- **SLA** – an agreement between an IT service provider and the business that includes:
  a. Performance and capacity (such as end user response times, business volumes throughput rates, system sizing and utilization levels)
  b. Availability (e.g., mean time between failure for all or part of the system, disaster recovery mechanisms, etc.)
  c. Security (e.g., response to systematic attempts to break in)

- **QoS** – addresses the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied objectives, (From ISO 8402).

Furthermore, the On Demand SOA solution design process must identify enabling technologies for various IT virtualization functions. Fundamentally, the On Demand SOA solution framework must clearly articulate to customers that:

- Transforming business processes to be more dynamic and responsive is only part of the solution.

- Re-writing or enabling applications with Web or grid services interfaces is only part of the solution.

- On Demand IT infrastructure enabled with grid (schedulers, resource brokers, and federated file systems), autonomic (workload management), and utility (provisioning) resource management is irrelevant without corresponding process-based policy and application awareness.

1.3 On Demand SOA Related Technology

Many key technologies and/or standards must be considered as part of an IT strategy if the goal is to become on demand through SOA. The key focus of enablement of SOA includes virtualization of the infrastructure and application automation of management. However, it is worthwhile to note that developing an On Demand SOA strategy must not be a "throw away everything and start all over again" philosophy. Enabling business processes, applications, and IT infrastructures cannot be a monolithic effort with rigid goals such as turning every application in an enterprise into a Web or grid service, or completing by a specified date, or orchestrating all server, network, and storage resources with a single policy. The very nature of an on demand application or operating environment is quite fluid. Some applications may be very suitable for immediate transition to an SOA, particularly those that are written in Java 2 Enterprise Edition (J2EE). However, most legacy applications and most system, data, and resource management IT middleware tools are not easily transformed into SOA and are prohibitively expensive to redesign in a fashion that is most suitable for SOA. In order to migrate these applications and IT middleware and tools to the SOA environment, we need to provide interfaces in OGSA or WSDL to expose these legacy services. We talk in more detail about how this is done in later sections, but our point here is that key technologies must allow for this type of flexibility when enabling applications or IT infrastructure for an On Demand SOA strategy.

1.3.1 Web Services

Web Services comprise a technology component and a business component. As a technology component, Web Services provide a self-describing connection technology, open standards, and a new programming model, which includes the following:

- Connection technology that enjoys strong industry leadership and early tooling, is rapidly evolving and gaining momentum through a combination of hype and growing experience.
- Standard interfaces that provide publish, discover, and subscribe capabilities for resources with no concern for actual implementation.
- Programming model that enables the SOA, with emphasis on request-response type interactions for now, and composition based, flow oriented interactions in the future.

As a business component, Web Services can be implemented to support an underlying business process or a packaged IT capability (e.g., "Open 529 Account" or "Add Beneficiary"). These Web Services can interact with other applications or business processes and operate in a loosely coupled fashion, and can be discovered by other systems.

Web Services propose easier integration within and across the enterprise. Simple point-to-point process integration internally and with trusted partners promises the most immediate value. In the future, challenges will focus on the ability to plug and play Web Services into virtual, dynamic, on demand applications to accomplish complex tasks. The maturity of standards in areas of security, transaction management, and QoS will be a key catalyst toward realizing the promises of Web Services.

Web Services are more likely to be adopted as the de facto standard to deliver effective, reliable, scalable, and extensible machine-to-machine interaction than any of its predecessors as a result of the timely convergence of several necessary technological and cultural prerequisites. These include the following:

- A ubiquitous, open-standard, low cost network infrastructure, and the technologies that make a distributed environment much more conducive to the adoption of Web Services than both CORBA and Distributed Computing Environment (DCE) faced.
- A degree of acceptance and technological maturity to operate within a network-centric universe that requires interoperability in order to achieve critical business objectives, such as distributed collaboration.
- Consensus that low-cost interoperability is best achieved through open Internet-based standards and related technologies.
- The maturity of network-based technologies (e.g., TCP/IP), tool sets (e.g., Integrated Development Environments (IDE's), Unified Modeling Language (UML), etc.), platforms (e.g., J2EE), and related methods (e.g., Object Oriented (OO) Analysis and Design, etc.) that provide the infrastructure needed to facilitate loosely-coupled and interoperable machine-to-machine interactions; a state far more advanced than what CORBA users experienced.

1.3.2 Grid Computing

Grid computing promises the adoption of a computing paradigm that changes how we view processors and data repositories across an organization. The Internet was essentially the network incarnation of a grid, whereby nodes were all made equal with inherent redundancy provided through the dynamic allocation of lowest cost route (e.g., Open Shortest Path First (OSPF)) and other similar protocols. It was created with no real mechanism for guarantee of service or service level commitment. The Internet was the first step in moving from hierarchical computing models to peer-based computing models. Virtual organizations are a fundamental concept to grid, as they represent a logical pooling of resources within a grid. Grids are about the problem of enabling resource sharing among

dynamic collections of individuals (users or applications), resource collections (infrastructure elements), and virtual organizations.

Grids are a coordination of resources using open, standard interfaces to deliver QoS to an application. Resources can consist of physical as well as logical elements, such as job schedulers or distributed file systems, but resources are defined by their interfaces, which are created in a grid using Web Services as the interface definition mechanism. This provides a foundation for modeling common grid services at the resource layer, connectivity layer, such as discovery, access, reservation, allocation, authentication, authorization, communication and notification. Early incarnations of open source grids were built using the Globus tookits v2 [GTKV2] or earlier. The move to Globus toolkit v3 [GTKV3] is a key change in that the core protocols are now replaced with Web Services interfaces defined by WSDL, SOAP communications, and XML payloads. The most basic taxonomy of grid includes a job scheduler that works with resource pools.

Grid technology provides an open standards approach for resource sharing and virtualization across an enterprise, allowing for the virtualization of the various applications on heterogeneous operating systems within an On Demand SOA. Grid technology provides the resource discovery, load balancing and scheduling, monitoring, accounting and billing, and license management features. These features allow the application to be decoupled from the specific hardware, achieving much higher utilization of each server and the potential of much faster processing and throughput of the applications.

Grids are typically built-out of islands of compute resources, which are owned by disparate lines of businesses with varying management domains. Grid middleware facilitates the sharing of these resources across these management domains, while enforcing policies for security, load balancing and scheduling and monitoring usage levels for later chargeback and reporting. An On Demand SOA requires these same features to extend the architecture within the enterprise by pooling its existing resources, while providing an open standards approach (via OGSA) to tap into external resources for 'peak shaving' during high demand periods.

The example grid solution architecture described below provides a means to:

- Create an application layer server virtualization mechanism
- Enable additional capacity on a variable utility basis
- Remove the requirement for over-provisioned, under-utilized infrastructure
- Enable applications to be decoupled from dedicated servers and be able to run virtually across their networks on the best suited resource at run-time
- Enable applications to migrate processes dynamically to the most appropriate resources based on business policies
- Leverage and optimize the resources available in the organization's infrastructure, including sharing application workloads between the centralized host and the distributed computing resources
- Implement a utility computing model that incorporates dynamic resource sharing and integrated accounting and reporting for chargeback and cross business silo sharing
- Increase application resilience, performance and/or scalability.

The Open Grid Services Infrastructure (OGSI) provides a framework and programming model for application developers to build grid services using existing open standards such as WSDL and SOAP. However, unlike existing Web Services containers, the OGSI container provides additional state management services that allow global referencing and additional security services supporting message signing, encryption and other authorization services. Therefore, grid technology provides infrastructure virtualization enablement and application enablement within the context of an On Demand SOA.

1.3.3 Autonomic Computing

Autonomic computing fundamentally drives the automation of traditionally manual operations management functions in a distributed systems environment. Functionality that has been provided within the mainframe is now being applied to the distributed environment. Over the past two decades, distributed operating systems and servers have become much more mature. UNIX systems have continued to evolve to develop robust capabilities in terms of cluster integration and advanced cluster management solutions. In addition, more advanced UNIX platforms provide dynamic capabilities such as on-line kernel re-compiling. Similar functionality is emerging and becoming more commonplace across what used to be termed personal systems. Low end servers are now co-opting advanced functionality that exists in mainframes and other large systems. Server hardware is capable of being actively reconfigured (hot swappable disks, memory, etc.) with minimal down time. In addition, storage has evolved from siloed disks within servers to common storage area networks (SAN) accessible through a shared high-speed data fabric. This offers the added benefit of massive economies of scale and flexibility in configuration by decoupling the storage from the processors. Storage pools today allow for dynamic partitioning of high performance disk arrays, which are managed inherently by the SAN devices and advanced SAN management solutions. This dynamically reconfigurable processor and storage environment sets the stage for required advances in IT management.

Autonomic computing defines four key areas: self-Configuring, self-Healing, self-Optimizing and self-Protecting (self-CHOP). Some aspect of each of these should be in an On Demand SOA.

Self Configuring, an integral part of an autonomic system, is the ability for an autonomic element to be introduced into a system and to register itself and its capabilities to the system. The self configuring functions support deployment and installation of autonomic elements and managers into a system and also the boot strapping of the function as it becomes active in the system.

Self Healing features anticipate errors or problems in the systems, detect faults and isolate errors from harming the systems and take actions to recover from the error and maintain the system function. The self healing also considers error avoidance by monitoring events and performance metrics to take forward-looking action, such as provisioning of additional resources, or by throttling back on transactions to avoid overrun conditions in the end to end solution.

Self Optimizing features monitor current utilization and elements of the operations, adjust the autonomic computing elements and tune them for optimal performance (i.e. DB2 SMART). These features could also optimize the resources utilization.

Self Protecting features focus on making systems more resilient to attacks and intrusions by hackers or viruses.

Within IBM, four functional blocks that are fundamental to autonomic system and application management have been identified: monitor, analyze, plan, and execute (the MAPE loop). Monitoring consists of gathering data from system and application components required to make control decisions regarding application or system health. Analyzing consists of gathering data and processing it to determine whether the system is healthy, based upon thresholds and perhaps online modeling. If the data analysis determines that the system is unhealthy or will be in an unhealthy state at a future time of interest, action must be taken to fix the underlying problems. Certain steps must be orchestrated for tuning, problem resolution, etc. this is the planning phase. Once a plan to correct the problem has been established or simply been retrieved from a database, the plan must be executed.

1.3.4 Utility Computing

Utility computing is concerned with two different types of customer scenarios: server consolidation to create intra-enterprise utilities and third party hosting services acting as public utilities. Server consolidation focuses on TCO reduction through resources centralization. This consolidation effort reduces system administration, network maintenance costs, and increases utilization by allowing users across business units to access unused cycles on a wide array of resources. Customers usually contract third party or public utilities to address the prohibitively expensive cost of adding additional resources for temporary or transient computing needs. A better business case can often be made for renting resources for increased workload or demand. In either case, however, utility customers are concerned with pricing (or charge back), security of shared resources, high availability, and QoS. Utility providers are also concerned with the following issues:

- Implementing infrastructures which can adapt to changing “pay as you go or use” pricing models
- Providing secure resource access for provisioning and de-provisioning
- Automating provisioning of new resources for ease of administration to reduce the cost of the utility
- Monitoring or SLA’s for policy based resource management
- Metering usage and establishing certified billing processes
- Providing subscriber management functions to manage contracts
- Providing yield management to maintain accurate offerings derived from resource capacity and predicted utilization

Standards will be key for on demand services deployment in a utility. This is analogous to the use electrical outlets and the supply of electricity throughout the world. As a person moves from country to country, a "plug adapter kit" must be purchased because there are no worldwide standards for electrical outlets. It would be inconceivable to have proprietary interfaces for a given utility, which would "lock" an on demand service into only using that utility. Therefore, key to a utility providers infrastructure is the use of open standards.

2 Customer Scenario

In order to illustrate the concepts of On Demand SOA solution development, we present a scenario derived from an actual case study with a large financial services company. We call our business process scenario, “Life Change”.

2.1 Business Process Description and Customer Goals

Consider a client service consisting of several processing steps that a customer must follow to modify employee benefits when a new child is added (by birth or adoption) to a customer’s family. This includes health insurance additions, life insurance increases, and investment portfolio analysis. More specifically, the business process consists of the following fundamental sub-processes:

1.) Change W-4 exemptions
2.) Add dependent medical coverage
3.) Add beneficiary to 401k
4.) Open 529 account
5.) Run advice engine to suggest an investment strategy to achieve 529 goals
6.) Create payroll deduction to fund 529 plan and adjust 401k investment options
7.) Increase term life
8.) Run advice engine to suggest funding mechanism for term life, with minimum tax implications
9.) Suggest selling shares to fund term life increase and execute sell
10.) Run portfolio analysis including pension plan estimate with various fund accounts to meet new long term financial objectives
11.) Suggest rebalancing of portfolio and execute rebalance

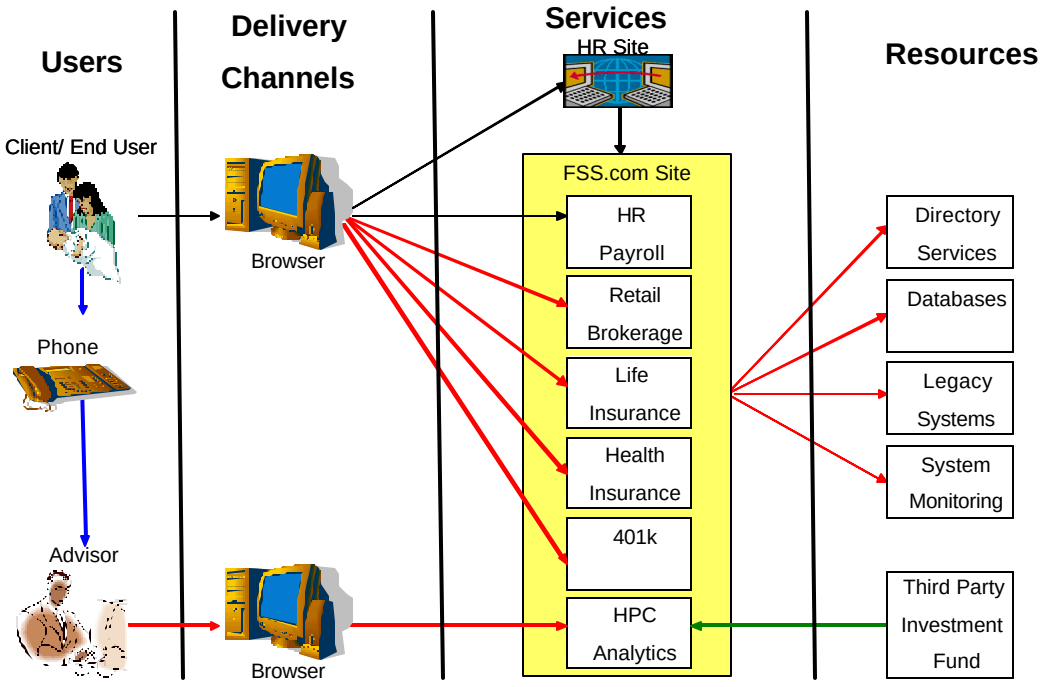

Figure 1: **Current implementation of a financial services company's "Life Change" process**.

In Figure 1, black arrows indicate the functions outlined above that are initiated from the customer's HR site, red arrows indicate the functions that are initiated from the company's web site, blue arrows indicate functions that are now initiated by a phone call to an advisor and are asynchronous in nature, and the green arrow indicates a third party interaction.

The current implementation of this process consists of applications and resources distributed not only across different departments (i.e. health benefits and investment benefits), but also resources outside the company (external investment funds). From an application point of view, these steps include stateful and stateless transactions, stateless analysis engines, and stateful data interactions. Furthermore, the process is currently supported by on-line sub-processes and off-line batched analysis/advice engines driven by asynchronous requests, including phone center requests. These flows are illustrated in Figure 1.

The current process implementation requires customers to follow a list of action items, connect to different Web sites or electronic forms, make phone calls, wait for data to be generated by various applications, and share data among the different applications. The client company has indicated that the current process results in a level of customer dissatisfaction as well as a number of customer requests that cannot be served by current resources. This threatens competitiveness in the marketplace and, therefore, the client is in danger of losing business in a highly competitive market.

The financial services company's primary goal is to find a low cost solution to improve customer satisfaction and increase request throughput. The resulting higher efficiency from the improved process would in turn generate higher volumes of completed requests and drive additional business as more customers are served. More specifically, we can define the objectives of business process transformation for the "Life Change" scenario to be:

1.) A single identity for customers for both internal and external interactions
2.) Collaboration with partners through real-time interfaces replacing off-line batch interactions
3.) Consolidation of many customer facing systems into consistent customer interactions
4.) Dynamic use of system resources as types and volumes of customer requests change
5.) A customer experience that allows a combined view across all relationships, both internal and external

The next section outlines QoS and availability policies, for the overall business process and the sub-processes listed above that are required to understand the On Demand SOA solution.

2.2 Process Driven Meta Service Composition and Policy Definition

From the list of steps given in section 2.1 and the functional flows shown in Figure 1, we can immediately see that within the coarse grained "Life Changes" business process there are sub-processes that will become services. Hence, we will associate the "Life Change" business process with a corresponding composition of services or a *meta service*. In this section, we will demonstrate that each of the services have an associated policy, whether that be in terms of QoS or availability, in order to meet the overall business process goal. Therefore, one way to evaluate our customer scenario is through the combined process decomposition into multiple services and the policy drivers for each service. Establishing this view of the customer scenario is vital, as it is a key ingredient for our design work in the sections that follow.

As part of policy based management, control hierarchy and services must be described. The policy hierarchy is used to decompose business process requirements into policy statements regarding the operational and functional characteristics of the services and underlying infrastructure. This is important, because formerly functional and operational requirements were captured as part of a static architecture that was manually and iteratively tuned. With the adoption of On Demand SOA, it becomes necessary to capture requirements as policy statements in human and machine-readable formats. Thus emerging technologies, such as WSLA, have been created.

Regardless of the implementation, it is important to understand the steps required to decompose a process into a series of meta services, services and operational characteristics. Process workflows and application use cases are used as a basis for defining the composition of services. During the development of process workflows, functional and operational characteristics are captured such as: time and resource constraints, environmentals, performance, capacity, security, organizational issues, etc. These characteristics, once documented, need to be implemented in an appropriate resource management schema. Process requirements are used in order to provision enough resources to meet QoS requirements defined within SLAs. Global resource managers in coordination with schedulers and workload managers determine what hardware or operating environments are most suitable and available for a specific job or task. In addition, environmental requirements may dictate that a specific type of engine or configuration be provisioned or instantiated in order to reallocate the environment based on a predictive analysis of workload trends and/or a manually triggered allocation or configuration change. These policies are defined as business, functional and operational (non-functional) requirements within a typical On Demand SOA solution architecture engagement.

A second set of policies define guiding principles which are focused on the management of the environment in terms of how the organization, processes, and tools are defined, developed, and built. This is required in order to sustain a well operated environment. Abstracting the processing/data environment away from the services development environment introduces a series of issues that must be addressed. The separation of business, application, and IT policies requires that various issues be correlated, rationalized and arbitrated. This is covered in more detail in section 3.4.2.

3 Designing an On Demand SOA

There are multiple techniques that can be used to perform business transformation. Transformation implies that re-engineering process workflows do not end solely with a successful implementation of a system or education program. Rather, business transformation has many broader elements including those handled by organization and change management competencies. On demand requires the key elements of an organization transformation to be in place. The culture of the organization can continue to be a major obstacle in adopting new processes, technologies or management techniques. In addition, an on demand transformation is much broader than any tactical initiative and needs full management leadership and commitment from the top down. Dynamic process and technology sourcing decisions will not make people comfortable without appropriate communication and education. As a result, business transformation projects are put at risk with employees potentially resisting any change and in some cases acting as saboteurs to the adoption of new processes. In addition, technological constraints, whether purposefully implemented or due to limitations of current technologies, can impose barriers to meeting certain performance

objectives. Performance objectives are typically metrics developed into business processes in order to measure key attributes such as speed, cost, and accuracy of results.

Business Process Integration is the business enabler that requires a very dynamic SOA to be in place in order to provide the necessary workflow plumbing. In addition, application development tools have evolved to a point in which process lifecycles are enabled end to end and a process modeling tool can automatically generate use cases and interaction models. These models can then be dynamically instantiated where components are provided as part of the application framework. Applications are assembled from a collection of components that provide specific functionality. This functionality becomes part of an extensive library that can be published as Web Services that can be reused by others, by simply discovering the service via the appropriate service directory. The benefit of this model is that it is based on applying application frameworks, which are being improved on a daily basis to add increased breadth of packaged functionality. This utopian view of services is still a ways off in the transformation roadmap. However, by painting that picture we can clearly see the large gap from today to the future. Initial opportunities for process transformation stem from development practices and are driving towards consistent reuse of code and common lifecycle management processes and tools. IBM has strong application effectiveness capabilities to help clients start to improve the development, testing, and deployment of applications to large shared production environments. Additionally, legacy applications are being maintained today with the expectation that over time they will be transitioned to a new paradigm through custom efforts, off-the-shelf solutions, or through service providers.

Highly dynamic application environments that are built upon readily instantiated services, require a highly dynamic infrastructure that can:

- Support the dynamic provisioning of resources needed to instantiate the service
- Measure the Service Level Characteristics of the service
- Sustain Service Level Agreements within tolerance by adjusting the infrastructure from available resources

Furthermore, integration of the underlying resources with these services and the enablement of corresponding IT management systems must be accomplished through the use of open standards. Applying open standards will allow companies to maximize the benefit of various vendor component solutions and that these components will interoperate.

Using the scenario described in Section 2, we now proceed to the steps required to design an On Demand SOA solution. These steps will focus on the concepts of business process and policy, application, and IT infrastructure enablement. More specifically, we will need to understand the business process in the following contexts:

1.) Current state workflow definition
2.) Fits/gaps against industry standards
3.) Componentization of workflows
4.) Parameterization of workflows
5.) Policy definitions within the workflow

Once we understand the business process within these contexts, we can proceed with application and IT infrastructure enablement since one of the goals for our solution framework is for the business process and policy to drive the application and IT infrastructure implementation.

3.1 Process and Policy Definition

The first step in building a solution for the financial services company's "Life Change" scenario is to clearly articulate the goals of the business process transformation. In section two, we summarized the goal in two parts:

1.) Increase customer satisfaction with a more uniform and seamless approach to the "Life Change" business process.
2.) Increase the throughput of available request processing by leveraging existing resources and increasing the utilization therein.

Furthermore, the combination of these two goals results in the financial services company remaining competitive in the marketplace.

We also described the "Life Change" process in section two as a series of steps executed to build a complete business process. At that point, we alluded to the fact that this process could be thought of as a meta service composed of a set of service components in a workflow pattern. That workflow is shown in Figure 2.

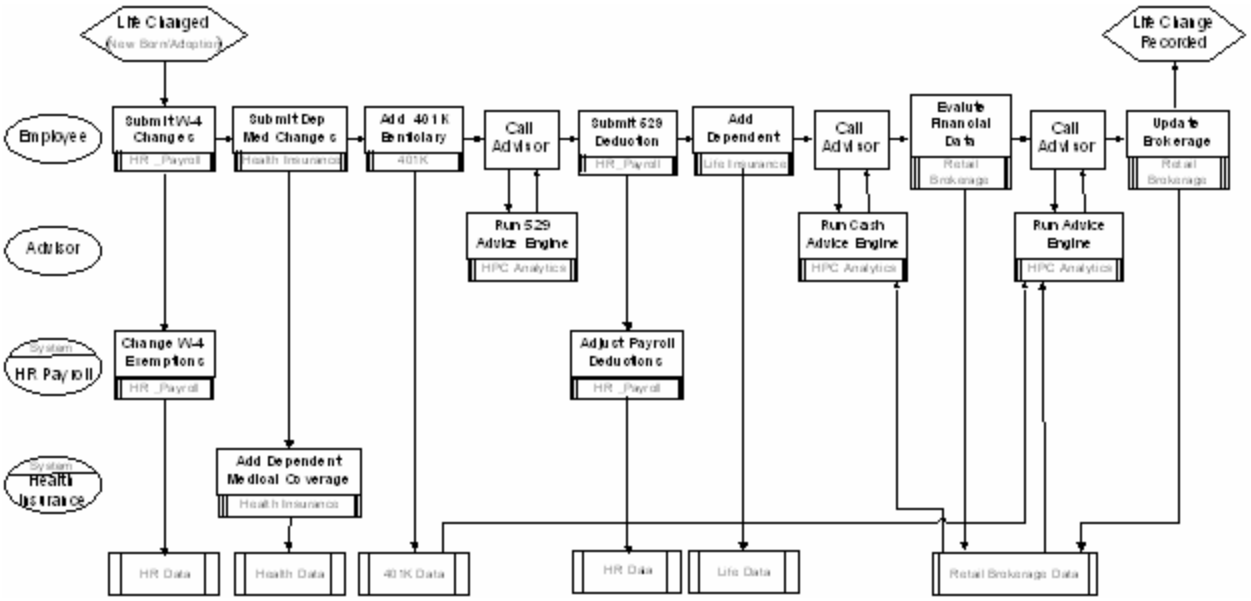

Figure 2: A flow diagram indicating sub-processes and databases for the "Life Change" process

We now must define some overall process policies, which will in turn generate policies for each individual service. Let's assume that market research indicates that in order for the financial services company's implementation of the "Life Change" process to be competitive, the time to completion for an end-to-end request must be less than five minutes. In addition, customers expect a 95% availability rate for the "Life Change" service (this is across all componentry), and are willing to pay more for increased availability or throughput. Finally, and as outlined in the objectives in section 2.1, the financial services company needs to present a single client facing portal for all services within the "Life Change" process to simplify the process. We can therefore set the following process policies:

1.) An end-to-end request must complete in 5 minutes minimally.
2.) The supporting application and IT infrastructure should have a 95% availability target.
3.) A gold customer will have precedence over a silver customer, and a silver customer will have precedence over a bronze customer. Customer classes will be based upon a price/throughput tuple.
4.) All data feeds (data supplied by the customer/data sent to the customer) must go through a single, secure application portal.

Once these high level policies have been set, we must derive policies for each of the component services listed in section 2.1. From an availability aspect, we can make the simplifying assumption that if one service is not available, the entire process is unavailable, and hence, we can state that each service must also have an availability target of 95%. The response time QoS will require a greater understanding of the complexity of the applications and the potential infrastructure used to run each of the services. For instance, how will we determine a "theoretical" minimum response time for each of the services? We may decide that each transactional service must complete in 20 seconds and the analytics computation must complete in 2 minutes in order to reach our overall business process QoS. Establishing customer classes on a per service basis to mediate requests when these objectives cannot be met will be a further policy requirement.

There will be a single point of entry and exit for data. Consequently, there will be a single governing security policy for the process and the individual services will adhere to this policy simply by presenting data (i.e., analytics data, account listings, etc.) through the secure channel provided by the initial login/authentication service. So, a process policy has actually driven a new service to be defined – a login/authentication service. We redefine the services workflow for the process and show this in Figure 2. It should now be clear why process and process policy must drive the services and service policy definition.

3.2 Resource Mapping

Now that the business process and corresponding services have been identified, we need to qualify the resources -- software, hardware and change management -- that will be candidates for re-engineering. In the "Life Change" scenario, we need to identify the resources associated with each of the services outlined in section 2.1 (application and corresponding system infrastructure, i.e., servers and DB), and map dependencies for each service. This is especially important in understanding how legacy transformation fits into application enablement. For instance, the applications corresponding to the HR Payroll service may exist entirely on mainframe technology using underlying messaging middleware. We need to understand the effort required to move those applications and corresponding middleware to another hardware environment like a Linux cluster. This contributes to definition of our approach (outlined in the following section) that describes applications transformation into services. This approach calls for the maximum possible decoupling of applications from the underlying resources on which they currently operate. Figure 3 shows a possible representation of this resource mapping, with an application definition and a hardware dependency shown within each service in the workflow. Note that in Figure 3 we have omitted a physical topological drawing of the actual IT resources in the interest of brevity. However, physical topology is also a necessary step in the resource mapping to determine what actual physical resources will be affected, especially if the resources cross departments or lines of business.

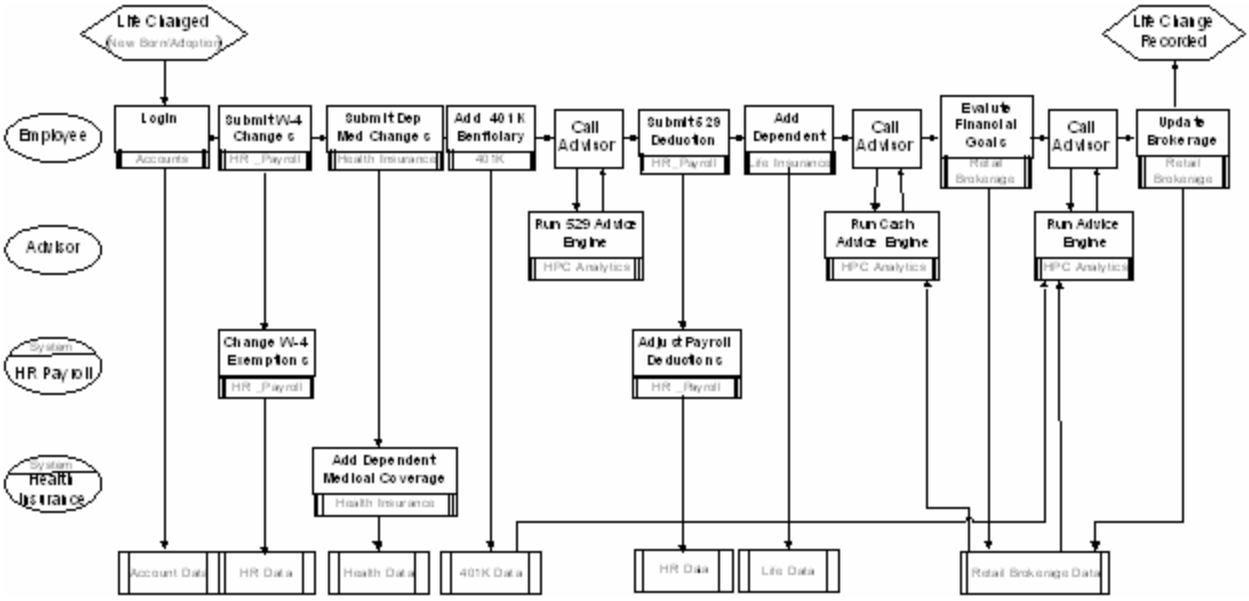

Figure 3: A flow diagram indicating sub-processes and databases for the "Life Change" process.

Note that we have added a Login sub-process and a corresponding accounts database to comply with the single point of entry policy.

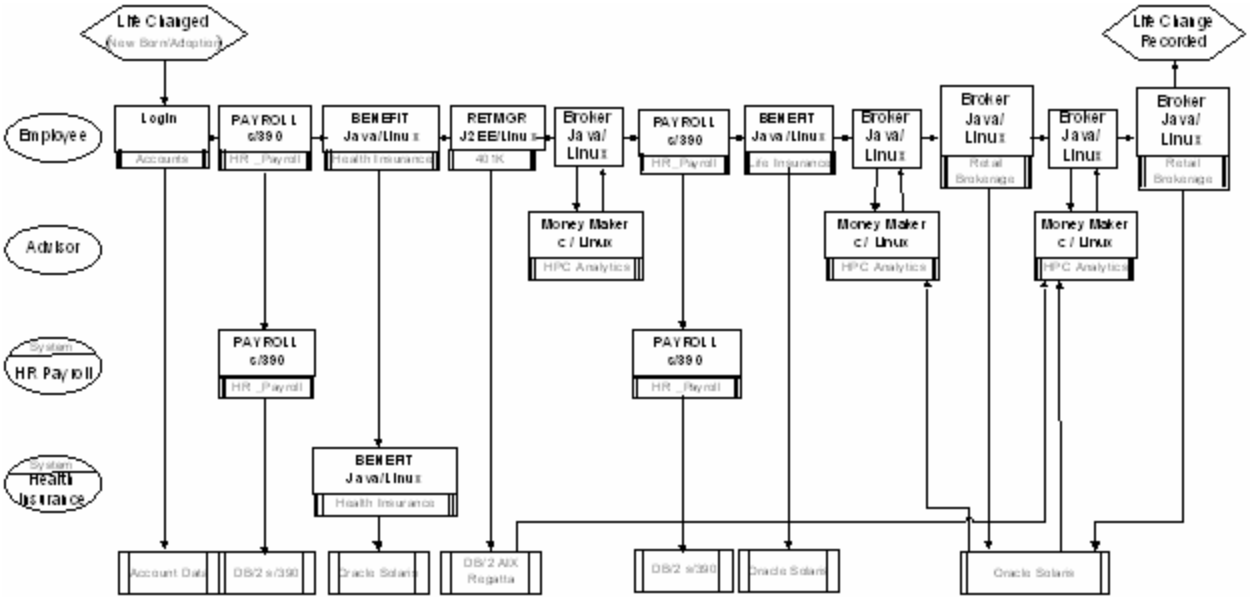

Figure 4: A resource mapping (application and o/s) for the "Life Changes" process corresponding to Figure 2.

Note that we have not drawn the true IT topology map for the software and hardware dependencies.

3.3 Application Enablement

At this point, the services and corresponding workflow have been identified. We have also mapped application functions to services they will deliver. Now we must decide how to best encapsulate those applications in the SOA. At one extreme, we could completely re-engineer and re-write the application to make it hardware agnostic from the outset (i.e., J2EE) and easily exposed as a Web or grid service. However, this could be prohibitively expensive. At the other extreme, we may choose NOT to implement an application as a service, but instead continue to invoke the application "as is" with a service layer wrapper around the software. Clearly, this does not enable the application to run on every platform, but may be sufficient to achieve the overall business process goals. In the following sections, we discuss principles for the application enablement within an SOA. We use the "Life Change" scenario to illustrate some specific design decisions.

3.3.1 The Application Enablement Environment for SOA

One of the most fundamental concepts of SOA is that a service is made available by publishing the interface specification in the application environment. The application environment can then be used to create process driven workflows that are easily reconfigurable through the dynamic integration provided by advance workflow engines coupled with standardized application interfaces. In addition, the application environment must provide a framework within which developers can easily determine when and how to abstract the data. Various repositories are linked together through these abstractions and corresponding data sharing mechanisms. Applications can begin to use common interfaces in a common namespace. The discovery of the data location and its replication and caching is handled by the underlying infrastructure.

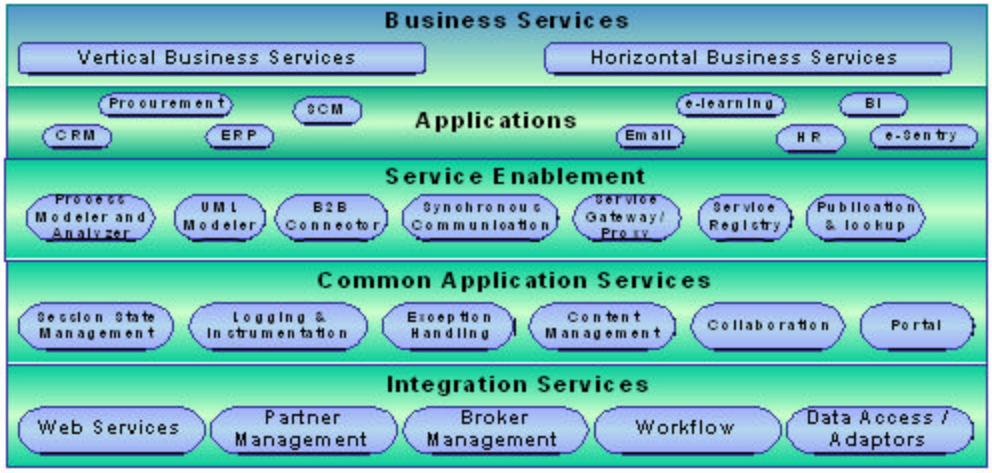

Figure 5: The application environment stack showing different levels of virtualization for application enablement within an On Demand SOA

Figure 5 shows the "stack" within the application enablement environment. At the top of the stack are the services that make up the business process we are composing. The next layer corresponds to the applications containing the functions that the services must support. In a true SOA, the applications themselves may be written on top of underlying application services and integration services.

Once we have determined which applications provide function for services, we then have to decide at what layer in this stack we will be virtualizing the corresponding function or collection of functions as a service. As stated previously, the level at which this virtualization is done would ideally be determined by a compromise between compositional complexity (i.e., building a process workflow out of tens of services versus thousands) and maximizing code re-use. However, the reality of legacy application transformation dictates that this does not always hold. For instance, it may be impossible in some legacy applications, or at least prohibitively expensive, to decompose an existing application into a collection of services. However, when re-engineering is feasible, we need to take the following design steps:

1.) Determine compositional or fundamental building block model (what is ubiquitous for applications and at what level should abstraction take place) with an emphasis on dynamic and reconfigurable building blocks.
2.) Determine the programming paradigm for each fundamental block (i.e. data parallel vs. control parallel, distributed memory vs. shared everything, etc.).
3.) Select a programming model for each building block.
4.) Determine the statefulness of each building block.
5.) Select programming model for each compositional block and overall service (application), taking advantage of open standards for resource virtualization (i.e. Message Passing Interface (MPI), J2EE, SOAP, WSDL, OGSI/OGSA)

Before we continue, it is important to elaborate on point number 5. Fundamentally, in order for an application to take advantage of a distributed environment, the application must be written using distributed programming paradigms. There are two distributed computing paradigms: control parallel / multiple instruction multiple data (MIMD) or data parallel / single instruction multiple data (SIMD). We can think of control parallel as the production line approach, with each station in the production process modifying the materials to eventually get to a finished product. This is the classical view of a multi-tier architecture and is most often used with transactional applications. Data parallel codes are the mainstay of high performance, numerically intensive computing. The degree of parallelism in the SIMD code is determined by how much data sharing / message passing occurs within the computation; the higher the degree of data dependencies, the more message passing must occur between nodes in the network, and the computational speedup is reduced. Once the programming paradigm for the application has been identified, the programming (or implementation) model is selected as described above. For application writers

starting from scratch, the programming paradigm and programming model decisions are crucial to effective leveraging of On Demand SOA distributed programming environment.

In our "Life Change" scenario we use the information provided by the resource mapping shown in Figure 3 to draw the following conclusions about application enablement:

1.) The HR Payroll service involves a single legacy HR application, which we are unable to decompose into lower level services at this time so a simple WSDL wrapper connecting to a Java-RMI interface will be used.
2.) The analytics calculation, in the interest of performance, should be virtualized at the parallel job level, not at the level of independent tasks within the job (i.e., considered as a single service executing on a virtualized server which consists of several compute nodes).
3.) The login/authentication service is entirely new and stateless and will therefore be written using J2EE and WSDL interfaces, but will be based upon some of the technology from the retail brokerage HTTP client.
4.) Since the Life and Health insurance applications have Java APIs and are stateful and exchange transactional data, we will write an OGSI wrapper around these applications to service enable.
5.) The existing retail brokerage and 401k J2EE code will be enhanced with OGSI interfaces, since that data is stateful and transactional data is shared across both services.

3.4 Enterprise Infrastructure Enablement

A virtualized infrastructure is a fundamental step in implementing an On Demand SOA. The on demand environment requires an infrastructure that can dynamically adjust based on new workloads and changes in the business process. However, before we enable the IT infrastructure for on demand, it is important to determine if the existing IT resources are sufficient to meet our business process goals and policies. For instance, are there enough nodes in our analytics analysis engine cluster to handle a response time of two minutes for an investment strategy calculation? Once we have determined that sufficient capacity exists in our infrastructure, making assumptions that we will be operating in a virtualized environment with improved utilization, we can then begin to virtualize the physical resources.

3.4.1 A Virtualized Systems Environment

The on demand systems environment provides the infrastructure of connectivity to resources on an as needed basis. The infrastructure will also provide more commonly reusable and dynamically instantiated containers that will allow for ease of portability, and common interprocess communication. The OGSA specifications enable this to become a reality by evolving LDAP standards to Web Service oriented discovery services. The core systems environment will provide OS and hardware level integration that will enable the dynamic reconfiguration and partitioning of resources. Partitioning and reconfiguration will be performed by a number of controllers, which can be built conforming to the OGSA standard. Virtual machines interlocked with OGSA enabled provisioning systems will completely abstract processors and storage from OS images. As this occurs, resource allocations will be effected by service level objectives subsequently correlated to metering data provided by complex and autonomic monitoring sub-systems. These layers of virtualization and corresponding system management support are shown in figures 6, 7, and 8.

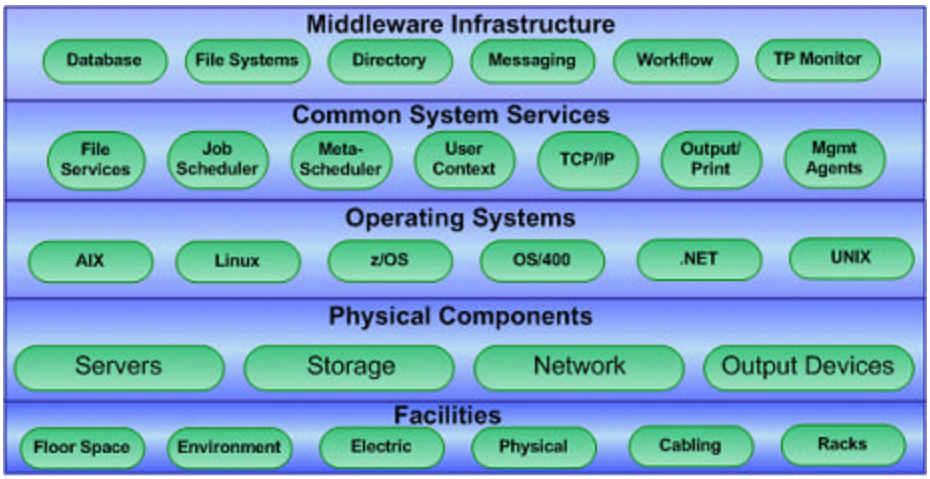

Figure 6: A resource stack view of Enterprise IT infrastructure

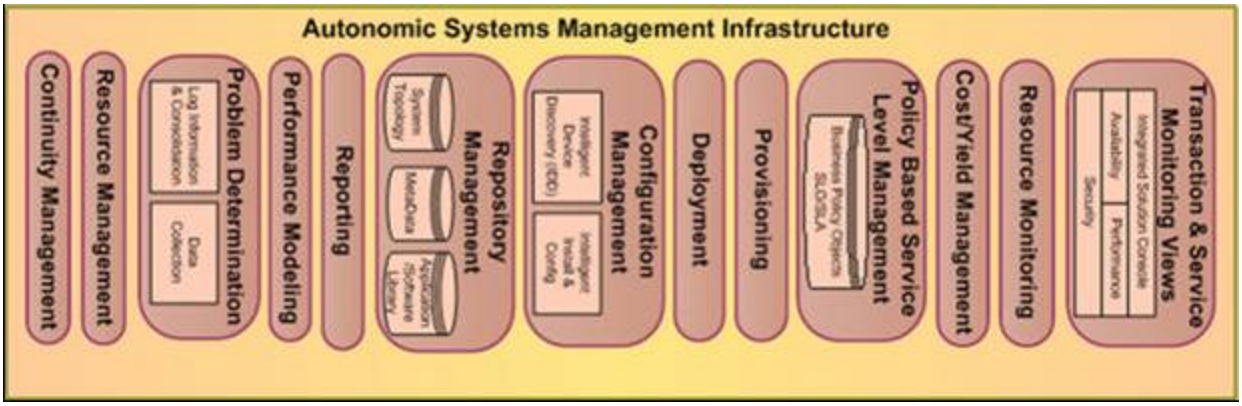

Figure 7: Required functions in system management infrastructure to support infrastructure virtualization

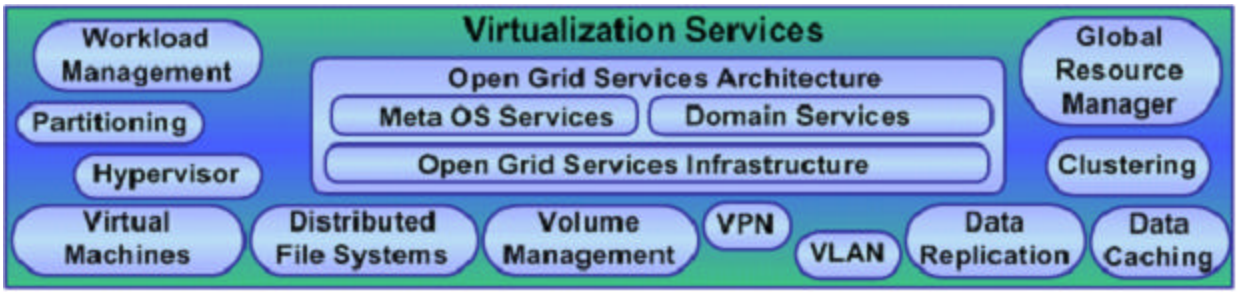

Figure 8: A description of the virtualized services provided by the system management infrastructure shown above

OGSA provides the bridging between the Systems and application environment, allowing for the virtualization of both systems resources and application meta OS services.

Finally, security and support services need to be interlocked in order to maintain service integrity and provide access based on service subscription agreements. Lines of Business (LOBs) will pay for what they use, and alternatively, the infrastructure will be compensated to maintain quality.

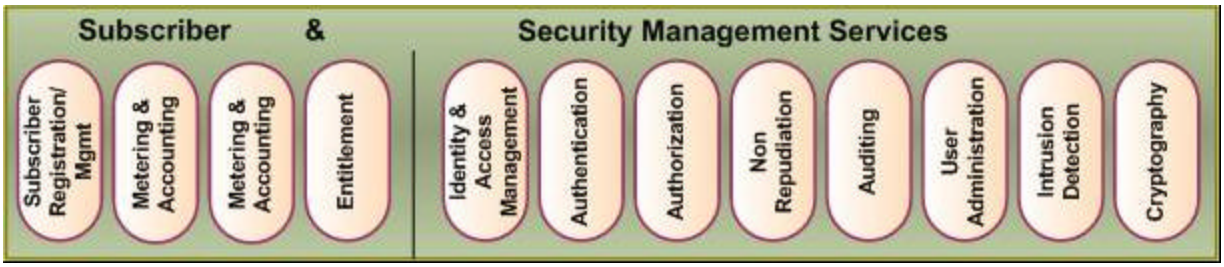

Figure 9: Virtualized security management services for the On Demand SOA IT infrastructure

3.4.2 IT Management Process and Policy Definition

Managing an On Demand SOA begins with translating business process policies into enforceable IT policies to manage the various services in a virtualized environment. A strategy must be put in place defining how to enable the various applications of the process as services. Once services are defined, subsequent policies must be identified and defined. Policies determine how an on demand business process should be orchestrated. These policies concerning privacy, security, authentication, resource sharing priorities, accounting and chargeback can be formalized through policy workshops and applied using IT management models. We describe those concepts here.

3.4.2.1 Policy Workshops

Policy definition is often conducted using a well documented technique known as a policy definition workshop. This workshop has its roots in what are also referred to as guiding principles workshops. Essentially, the mechanism by which policies are defined is through the facilitation of a meeting using a pre-defined straw man starter set.

Policy workshops can be used to define the guiding principles within an organization that determine how the organization, processes, and tools are defined, developed, and built in order to sustain an On Demand SOA. Key issues that are typically used as a starter include the following:

- **Sharing**: What are the organization rules of engagement when resources are contended for?
- **Ownership**: Who maintains the financial burden for the asset, and how is the cost allocated, distributed, and recouped?
- **Charges**: What are the usage charges associated with the utilization of services? To what level of granularity is usage metered, and what is the associated overhead associated with fine grained usage metering and billing?
- **Allocation**: How are these resources allocated and how are priorities defined?
- **Commitments**: How are SLAs measured, and how is monitoring data aggregated and mathematically modeled in order to measure QoS or SLA attainment?
- **Schedules**: What are the rules associated with defining batch, maintenance and other operational windows, where functionality of a particular resource pool may shift in allocation and or functional configuration?
- **Standards**: What level of standardization and interoperability will be required for architectural elements?
- **Designs**: What is the general approach or priority for migrating legacy applications, analyzing enterprise application portfolio, better leveraging HPC resources, and better leveraging transactional resources?
- **Services**: What core services must the environment provide in order to enable key management and accounting functions? How will various resource managers intercommunicate and publish their interfaces?

3.4.2.2 Management Model

The management model defines the point of execution and control across a management infrastructure. Naturally the points of application execution and control must also be understood; however in terms of management functions, a management model is instrumental in helping to institutionalize an On Demand SOA. There are various forms of a management model:

- **Centralized**: Central points of execution and central points of control are found in traditional legacy environments.

- **Enterprise Hierarchical**: This model is typical of a large distributed environment where control must be centralized, but systems will continue to locally execute functionality and communicate results upward vertically to a single enterprise manager. This is similar to an IntraGrid across an enterprise.
- **Distributed Hierarchical**: A suitable model for a federated approach to management (distributed and collaborative control), allowing local domains a certain level of autonomy (distributed execution), while ensuring that there is cross-domain collaboration as required. This is similar to ExtraGrid or multiple IntraGrids where resources are managed as distinct domains, but may at some level participate in a collaboration.
- **Workgroup**: A model where islands of automation are a viable and efficient solution due to simplicity.

3.4.2.3 Template based process and service re-engineering

IT Process re-engineering is often based on the IBM IT Process Model (ITPM) or as an alternative IT Infrastructure Library (ITIL). These comprehensive models provide a framework of activities grouped by relevant scope of processes. These various process outlines include where processes begin and end, activities, inputs and output, measurements, dependencies on other processes and data, and what is included and excluded in the scope of each process. These templates coupled with IBM intellectual capital and the IT Management engagement models, provide for a template based approach to design processes based on prebuilt workflows. The idea is that many of these workflows, such as problem management, are nearly 80% complete and applicable in most customers' scenarios.

Service definitions are changing based on new technology adoption for an On Demand SOA. The autonomic architecture defines a series of service flows, which essentially couple various process activities together in order to provide some end result to a customer of IT. As an example, change deployment can be coupled with other process activities to create a service, as an example, when it becomes part of a provisioning service. Provisioning, when defined as a service, can and will include elements of configuration management, and change management, among other process activities. The adoption of services definitions will become more critical as On Demand SOAs drive the need for more orchestrated cost and resource management. Users will truly become subscribers to an On Demand SOA and will necessitate the need for well defined and measured services that deliver a suite of capabilities. The intent is to decouple users and applications from an abstracted SOA framework and provide resources dynamically to address the QoS/SLA requirements of the business.

3.4.3 Virtualized Infrastructure Design

The guiding principles for a virtualized infrastructure are based on policy based resource sharing. Of course, the level at which resources are shared from an LPAR to a cluster, is partially dependent upon the applications which will be utilizing the shared resources as well as the organizational governance from which the policies are derived. For instance, in our customer scenario, both mission critical e-business applications and HPC analytics applications can run on Linux clusters. However, depending upon how one department can charge for use of those resources, i.e., on a cluster or on a per-node basis, this will help determine at what level the resource is virtualized, i.e., cluster or node level. In fact, the underlying system and workload management of that cluster may only be capable of a given level of resource sharing granularity. Before embarking on a virtualized infrastructure design, these types of constraints must be documented. Even more fundamentally, before undertaking the effort to consolidate resources for a virtualized infrastructure, some capacity planning and analysis is required. This planning will determine if enough resources exist to meet projected demand based on policies. This will be accomplished through continual on-line modeling and analysis of different systems.

3.4.3.1 Meta Services Scheduling

As we have previously stated, in an SOA design a business process can be described as a composition of several services; we will refer to this composition of services as a meta-service. A request from an end-user to execute a meta-service must have some initial policy based scheduling, most likely based upon user class of service and security based policies. For instance, customers in our scenario could be paying higher prices to ensure better "Life Change" process throughput rates; these "Gold" class customers would be preferentially scheduled over "Bronze" class customers. The meta service scheduler needs to decompose the meta service description of several participating services and identify a workflow or a calendar plan of how these services should be executed. This would include such things as synchronous versus asynchronous service scheduling constraints. Once the plan is in place, the scheduler routes the service to the appropriate workload domain for scheduling/instantiation on the local

resources. A workload domain consists of a request scheduler or router and a collection of resources. In certain cases multiple workload types can be scheduled within the same workload domain. For instance, the WebSphere services would be routed to a WebSphere Web Services scheduler, and the HPC application would be routed to the appropriate HPC scheduler in our "Life Change" scenario. Data and state that must be shared by the various services within the workflow can be maintained within the meta services scheduler. Some SOA standards such as OGSI allow for state notification and data sharing so that this function would not be utilized within a meta services scheduler. We also note that the meta services scheduler could act as a single point of entry for all resource (services based or otherwise) requests on the system, and then this scheduler would be responsible for routing the request to the appropriate workload domain for local resource scheduling.

3.4.3.2 Policy Based Resource Sharing

Policies for resource sharing can be as simple as time of day based, or they can be as complex as guaranteeing a certain level of performance or ability for a given service. In order to meet increases in demand, resources from other services must be re-allocated. The corresponding optimization problem for autonomic resource management across multiple workload domains is quite daunting. As a first step, however, we can rely on the human decision making process to decide when to move resources based upon QoS guarantees and QoS violation events. Each of the actual de-allocation and allocation processes can be automated by supplying tools and workflows for reconfiguring the available resources. We may refer to this as an enterprise, or global, or multi-workload domain view of resource sharing. However, in order to develop a comprehensive workload management system we need to further address additional levels of granularity within domains and resources.

In addition to the enterprise scenario described in the previous paragraph, we must also consider policy based resource sharing within a single workload domain, i.e., a WebSphere cluster, a LoadLeveler cluster, a Cisco cluster. We are concerned with how requests from different users for a given service should be handled based upon incoming order and/or customer type and/or current QoS statistics and if services of different types are managed within a single workload domain, i.e. within a single WebSphere cluster, how resources should be allocated for services of different types. For workloads of different types, the actual scheduling or load balancing algorithms could differ greatly. For instance for HPC parallel jobs, the scheduling requirements include resource matching, advanced reservation, and backfill scheduling for batch jobs. Interactive e-business or http request load balancers may not be concerned with advanced reservation or backfill. In a resource sharing system, both schedulers will need some user-based policy such as QoS, security, etc. These policies must be built into each workload management system they impact within the domain. We may refer to this type of resource sharing as policy based scheduling or load balancing within a workload domain.

Finally, resources within a single box can be allocated and managed according to a given set of policies. Operating system level controls, LPAR configuration, job swapping, paging, etc., can be used to manage workload priorities according to a set of policies within a server. Memory and CPU resources can be moved across LPARs using hypervisor controls, and jobs within a single LPAR can be halted while jobs of higher priority are swapped in to get the required system resources. Of course, the priorities of jobs, applications, and the like are set by the service policies. We note here that in all these cases, the mapping of services policies to the actual tuning knobs on a system, whether that system be a single server or a collection of clusters on a network, may not always be straightforward and may take some additional tooling to implement.

We need to relate this infrastructure to our customer scenario. Recalling the flows shown in Figures 1-3, we see that the 401k, Health and Life Insurance, Retail Brokerage, and HPC Analytics services can all run on Linux clusters. We can realistically divide this into two workload domains: an interactive, HPC parallel service workload and a Web Services e-business multi-tier transactional workload. Our meta services scheduler for the cluster is quite simply a scheduler, which knows how to route the different requests to different domain schedulers. The meta services scheduler is also a multi-domain resource manager which will build an initial set of nodes and dynamically manage the resources allocated between the two workloads to satisfy the policy constraints, most probably QoS or performance based, for the different services. The dynamic, autonomic management of these resources between two workloads is a very difficult task and a topic of intense research today. We should also mention that the meta scheduler would also be responsible for routing requests to the mainframe for the HR W-4 based service.

3.4.3.3 SLA Based Monitoring and Metering

In the above section, we reviewed requirements for policy based resource sharing. This type of resource allocation is impossible without the appropriate monitoring and event infrastructure in place. SLA based monitoring must be based upon service level guarantees and QoS violations. Furthermore, as we will see in the next section, SLA based monitoring is a requirement for accurate accounting, billing and chargeback, especially when we view pricing as a type of policy on the system. In addition to monitoring metrics specified in a given SLA, various components of virtualized resources must be monitored so that predictive models or analysis can provide information which can be used to better manage future allocations of resources. This type of analysis and allocation will enable the automation to anticipate SLA based violations before they occur, and take prescribed preventative measures.

Another important aspect of this type of monitoring relates to transactional monitoring and dependency analysis. Consider that one of our services is transactional in nature and the underpinnings of the service is a traditional two-tier architecture with an application server and a database server. This is the case with the benefits and retail brokerage services in our scenario. In order to determine how to best allocate resources for this service once the QoS based policies have been violated, we will need to know if more application servers or more database servers are required to meet our QoS goals. Bottleneck identification is a key for appropriate resource management in the On Demand SOA for transactional applications. Often times, the topology of a service request is not known and dependency analysis is required to determine how many tiers are involved in processing a request. Therefore, dynamic instrumentation of services to identify bottlenecks needs to be available

3.4.3.4 Accounting and Billing

In any virtualized On Demand SOA infrastructure, accurate accounting and billing will be required. For the scenario in this paper, Linux cluster resources will be shared between the analytics department, the retail brokerage group, and the benefits businesses. Beyond the meta service request for which we must provide accurate metering information for proper billing of end-user customers, these shared resources may be used independent of the meta-service request. For example, the analytics department may want to use the combined Linux resources when available to run larger and more complex value at risk simulations for portfolio analysis. They would be using the other department's and organization's resources to do this. We need to account for the fact that these resources were not being used to support the "Life Change" business process (or any other business process running on the On Demand SOA environment) and which department used them and for how long. Billing or chargeback for shared resources is feasible using this type of information

3.4.3.5 Federated Data

Thus far, we have really concerned ourselves with sharing of compute or server resources. However, most On Demand SOA scenarios, including our scenario will have data sharing requirements as well. To understand this, we only need to think about how portfolio management is done. In our calculations, we must consider investments in multiple funds, including 401k, 529, and actual stock portfolios. Most often, these data sets would exist on different departmental databases. When an advice engine is required, a human being needs to access disparate servers, collect the data by running a series of complex queries, and use the aggregate data to generate some investment advice. If the data was easily accessible programmatically by each of the services, the need for human intervention in this process would be removed. The architecture by which data is easily accessible in this fashion is called data federation, and in this case we would be federating databases. The case of file system federation most often occurs for numerically intensive computing jobs where data I/O has extensive performance requirements. In this case, data is persisted in files, and the files must be accessible to all nodes in a distributed system, including access across clusters. For our scenario, data in files may include states or checkpoints to save inter job state for each of the portfolio calculations.

3.4.3.6 Network Provisioning

Network allocation is also a key within distributed system and shared resource management. Some end-user customers might be of higher priority than others so we would need to implement policy-based bandwidth allocation on the network, both into the main site (portal) as well as between the different workload domains and within them as requests move from one service to another in the workflow. When nodes are allocated and de-allocated, network provisioning is also very important to ensure secure access as well as appropriate network topologies for many multi-tier transactional services. Network provisioning is also important as internal utilities begin providing hosting

services for external utilities. VLANs are one example of network resources, which are configured through switches to provide data and server security.

3.4.3.7 Capacity Planning

Key to any distributed system design is capacity planning for several reasons. Capacity planning in the On Demand SOA architecture is very similar to classic IT performance analysis work. There are, however, some new business process and IT policy implications for capacity planning for an On Demand SOA. First, business processes, applications, and underlying IT infrastructure of our solution, is on demand by design. This means that, ideally, we would like to dynamically allocate resources not just according to demand, but in advance of the peaks and valleys of service requests. For instance, we would like to allocate servers at different times of the day to handle the "Life Change" scenario transactional services when the request rate is highest, but perhaps re-allocate some Linux servers for different types of HPC analytics which could run over night to help provide more information for accurate portfolio balancing the following day. How far and with what accuracy these peaks and valleys need to be predicted is determined by IT QoS policy. Fundamentally, these changes in the prediction horizon interval correspond to a real time analysis requirement for on demand computing. This means that we may need to run some predictions within a closed feedback control loop every five minutes; these predictions may not need to be extremely accurate, but accurate enough to prevent over provisioning or thrashing. We may still choose to run highly-accurate, off-line, traditional capacity planning algorithms at the end of every day, once a week, or once a month to aid in on-line prediction accuracy or simply to assess the efficiency of our current system architecture. We would like for our capacity planning element in the On Demand SOA to be capable of both on and off-line analysis and prediction.

Capacity planning in the SOA environment means that we must also be able to provide performance analysis and prediction for services, not just single applications or transactions. This philosophical shift will require that a services dependency or topology diagram be made available, either through dynamic generation via traces and tokenization, or through static user description. Of course, these are really two ends of the "dynamic" spectrum, and we envision that there will be ways to generate dependency diagrams that are a combination of user defined topology and dynamic discovery of resources within a service. The service dependency diagram will correspond to an IT topology diagram, mapping services to resources. For instance, the Health Insurance subscription service corresponds to a two-tier application and database server architecture. Capacity planning for that service involves planning for both types of server capacity. The dependency analysis and corresponding IT topology map are required for that type of capacity planning for the service. Once the IT topology for a service is known and the desired values for analysis are input, the requisite measurements, including type and frequency, must be determined.

Finally, when On Demand SOA solutions include a grid implementation, we have the problem often referred to as "moving resources". By this we mean that resources in a grid (compute, network, storage) can join and leave a grid in a non-deterministic fashion. Therefore, we will not always be able to know when capacity will be growing or shrinking in our distributed system. We note that although this may seem like an intractable problem, most intra grids will be built with fairly strict IT policy about when and how nodes can join or leave a grid within the On Demand SOA. This is because business process, policy and governance drive the On Demand SOA, so that the risk of not knowing when resources will be available for a grid is mitigated because of underlying policies.

A summary of the On Demand SOA solution architecture is shown in Figure 10; a more complete account of the flows between On Demand SOA components is given in the figure caption. In the following section, we provide a proposed list of product technologies that could be applied to build a real solution. We also indicate where current products are lacking in function and provide a possible roadmap that could be used to get to a complete, On Demand SOA solution.

4 Implementing an On Demand SOA Solution

The final step in our hypothetical On Demand SOA customer engagement would be to implement our solution with existing products as they exist today and provide a reasonable technical roadmap for achieving a more complete solution. After all, one of our key premises for this architecture was that we could not afford to have a "throw it all away" solution, or a solution in which all of the applications and all of the IT infrastructure had to be transformed at once to see real benefit from an On Demand SOA. Furthermore, since we are developing solutions for an evolving technology, we are somewhat limited to what is already existing and mature within distributed computing and Web Services product bases. Not only that, since the On Demand SOA philosophy is based upon open standards,

implementations will rely on the open standards roadmap. Finally, we note that in this section we are proposing a set of technologies that are from autonomic, grid, and utility computing technologies. There may be other viable technologies solutions as well. However, our purpose here is to demonstrate by example how one could implement an On Demand SOA for a given customer scenario.

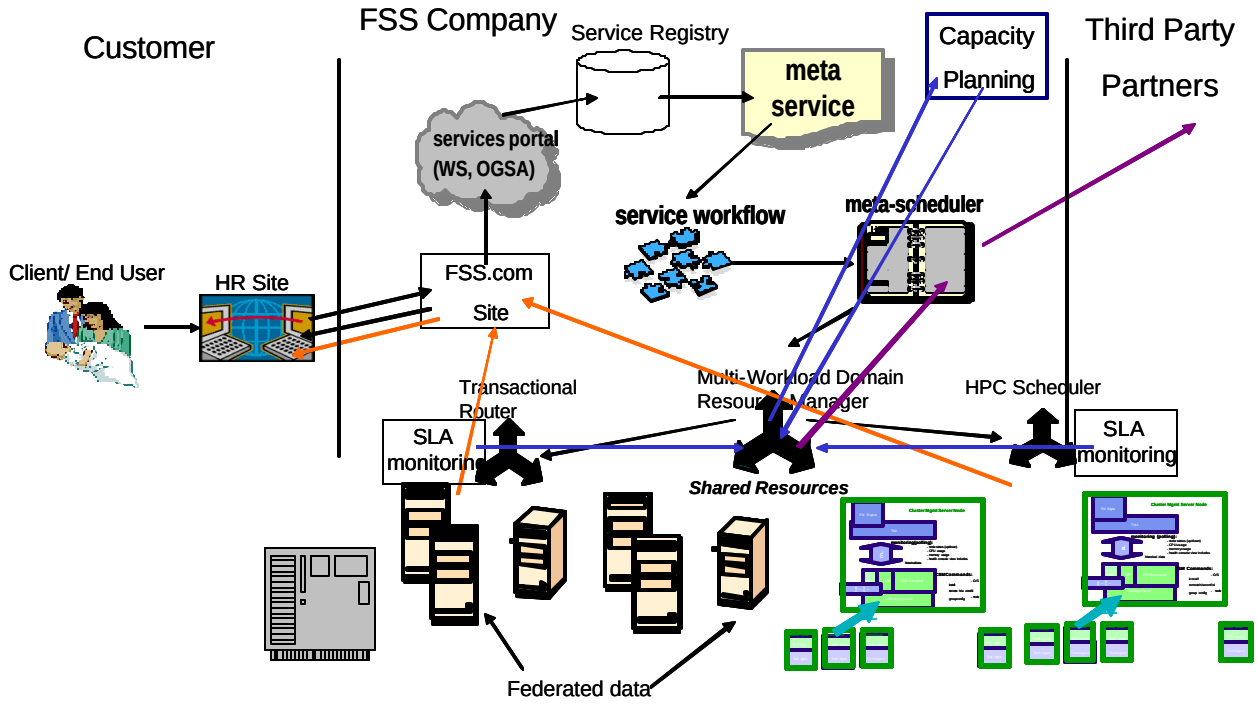

Figure 10: Architecture for an On Demand SOA solution based upon the FSS "Life Change" scenario

Black indicates service requests generated by the user. Orange indicates data flow (synchronous or asynchronous) from a service to the user, redirected through the FSS.com site. Blue indicates monitored SLA based data collected at the different schedulers or load balancers and sent back to the Resource Manager; the Resource Manager also provides this and perhaps other data to the capacity planning tool. The Capacity Planning component provides SLA based events or threshold analysis to the Resource Manager so that the Manager can either reconfigure existing allocations to keep up with demand before peaks or valleys occur or configure the meta scheduler to offload a request, if appropriate, to an available third party hosting service, which is shown in purple.

4.1 Technology Enablers within the On Demand SOA Solution

As has been the theme in this paper, we describe the technology in our implementation within business process, application, and infrastructure enablement. We do this by also calling out which technologies are autonomic, grid, and utility based in nature.

4.1.1 Business Process Enablement

Developing overall on demand corporate business strategy will remain the work of experienced business consultants who have an in depth knowledge of the enabling emerging technologies and open standards. Once an overall on demand business strategy has been developed, specific business processes, which have been identified for On Demand SOA enablement should be addressed within the context of a certain core group of technology. For instance, the different IT policy descriptions, i.e., Enterprise Workload Management (eWLM [AMAN97]) and WSLA, should be understood at a reasonable level so that the business process policy does not assume a level of IT policy where such capabilities do not exist. Business processes are enabled at the IT level by using policy based resource management. We will review the core set of those technologies in section 4.1.3. One specific technology

for business process enablement would be the language used to describe a business process or meta service. Today, Business Process Execution Language for Web Services (BPEL4WS [LEYMANN2002]) would be one obvious choice because it is standards based, and BPEL4WS supporting technology such as composition tools and service decomposition engines exist today. In our scenario, BPEL4WS could quite easily be used to describe our meta service as a collection of services, and our meta scheduler would then just need a BPEL4WS engine to decompose and parse the meta service definition. We will also describe more about the meta scheduler in section 4.1.3.

4.1.2 Application Enablement

As stated previously, the key decision for any services architect in an On Demand SOA is to decide at what layer different applications should be virtualized as services. This type of analysis should become part of existing Legacy Transformation & Application Portfolio Analysis assets and offerings. However, once this has been decided, the next real issue is Web Services versus grid technologies and standards, or WSDL versus OSGI. The decision on using OGSI or WSDL is at this point a very subtle one, and one that we would expect to become more difficult as the two standards begin to have more of the same features. For our scenario, we may choose to do something like this:

a. For the services which change portfolio or investment profiles (retail brokerage, 529 account creation, 401k management) we will write the services using OGSI, taking advantage of the notification mechanisms.
b. For the HR application and the benefits (life and health insurance) and the numerically intensive computing, we will use WSDL.

WebSphere provides a framework for developing and building WSDL based services called WebSphere Application Developer (WSAD). There is no such equivalent in Globus Toolkit 3 for building OGSI based services. Once the service has been built using OGSI or WSDL, a container is required on each node that the service will be instantiated. For Web Services, this is typically WebSphere [WASREF] or BEA WebLogic [BEAREF] or Tomcat [TCREF]. OGSI containers will be available on all IBM eServer platforms. In the meantime, and for non-IBM platforms, one can use any SOAP container, such as Tomcat. Again, the choice on which container to use will be a fairly subtle decision based upon available middleware budget, support, etc. For our scenario, we use WAS to develop WSDL applications and minimally build out Tomcat on every node in the system for containers, with WAS installed on the mainframe.

4.1.3 IT Infrastructure Enablement

We will now review the functional blocks from section 3.4.3 and Figure 10 and describe the corresponding list of candidate technologies from autonomic, grid and utility computing, which could be implemented to meet the design requirements of our scenario.

4.1.3.1 Scheduling

First, we assume that BPEL will be the language used to describe the meta service. Therefore, we will need a technology based on a BPEL4WS engine. Second, we will need a scheduler that has fairly rich APIs to plug in the workload domain specific load balancers and schedulers for a specific type of service. Third, the meta service scheduler should work across administrative domains as well. The Globus Resource Allocation Manager (GRAM) [GRAMREF] provides a rich set of standardized APIs for various schedulers so that these workload domain specific resources can be discovered and used by the appropriate applications. However, GRAM does not support a BPEL4WS engine. The DataSynapse broker manager [DSREF] provides service based scheduling across many resource domains, but it is also lacking a BPEL4WS engine and it has no policy based scheduling. Recently developed technology in research, called eUtopia [EUTOPIAREF] is a Web Services scheduler with SLA based scheduling, and they claim to also support meta service decomposition and requirements mapping. It is unclear at this time how eUtopia technology would support multiple WAS clusters. Not enough is known publicly about the Platform Symphony [SYMPHONYREF] product or the recently announced Community Scheduler Framework (CSF) [CSFREF], which will be an open source meta scheduler. Therefore, there could be three reasonable solutions for the meta scheduler in our scenario:

1.) Customers write their own with an underlying BPEL4WS engine and GRAM infrastructure.
2.) Customers write their own BPEL4WS and policy based scheduler and use DataSynapse to schedule services to the different workload domains.

3.) A prototype based upon eUtopia technology, including multi domain support.

Now, for each of the workload domains we also need a scheduler. For the transactional workloads, i.e., everything but the HPC Analytics service, we could use eUtopia technology as is. eUtopia is a good choice since it has some resource affinity and SLA based scheduling as stated previously. For the HPC scheduler, there are many choices including Platform LSF [LSFREF], Condor [CONDORREF], Open PBS [PBSREF], etc. Platform LSF is a sensible choice because of its existing Linux and multi-platform support, however, its licensing fees can be prohibitively expensive. Open PBS has a Linux and multi-cluster solution, but our customer will probably not want to implement an open source solution for a key middleware product like scheduling in a mission critical deployment. Another option would be IBM's LoadLeveler [LLREF], which at this writing is under limited availability on Linux; additional work would have to be done for multi-cluster support. DataSynapse might be another choice if the analytics application fits well within the DataSynapse programming paradigm. Platform Symphony actually has all of the function needed to schedule parallel services like the HPC Analytics, but much about the technology is still unknown and license fees may still be an issue.

We will now elaborate on another possible solution to the HPC scheduling and parallel service problem. IBM has an existing parallel application programming environment called Parallel Operating Environment (POE) [POEREF]. POE represents a distributed parallel job with one context in a Unix or Linux operating environment. Today, POE is a run time library, which contacts a resource manager, in this case LoadLeveler, to acquire a certain number of nodes with specified requirements to execute a parallel job on a system. POE provides wrappers around the main execution of the code on each node to start up and terminate jobs. In order for an HPC application to be treated as a service, POE would now need a Java wrapper to publish some type of WSDL for the service. Furthermore, if POE did not terminate the job once the main method of the code was finished, there would also be a mechanism to stream new data into the application code without restarting the job.

With that as background, one could now start up the parallel service using this modified POE and the LL resource manager. Once that has been done, we would only need a single services scheduler for the Linux clusters, since the POE parallel service would be seen to the service scheduler as just another service on a "virtualized server" (collection of Linux nodes to execute a parallel job). The task level parallelism within the job could be handled within the application, using standard master-slave parallel programming paradigm implementations.

4.1.3.2 Policy Based Resource Sharing

In our scenario, we are most concerned with how the resources on the Linux clusters will be shared for the 401k, Health and Life Insurance, Retail Brokerage, and HPC Analytics services. If we know that one service is more important than the others to make sure that the "Life Change" process executes with reasonable QoS, then we can give that service preferential access to resources. We now need to implement this type of preferential policy within a server and then within the distributed system.

Within a node, we can use eWLM to manage resources preferentially. Within a the Linux cluster we could use eUtopia's policy based scheduling to make sure that resources are scheduled optimally to meet different Web Services SLA metrics (i.e. throughput). Finally, we could use IBM Tivoli Intelligent ThinkDynamic Orchestrator connected to an SLA based monitoring and event system to dynamically re-provision the cluster(s) resources if the services could be split into different types of workload domains, like the HPC domain and the transactional domain, or even dedicated service nodes within the transactional domain. Furthermore, some feedback between these different components will be required for more coordinated resource sharing. At this point, the policy language between bWLM, eUtopia, and ThinkDynamics are all different.

4.1.3.3 SLA Based Monitoring and Metering

Before we decide what SLA based monitoring we will employ on our system, we have to decide what products are using SLA based monitors for resource sharing as well as accounting and billing. As we stated previously, the resource sharing infrastructure components actually use a myriad of policy languages and all provide relevant monitoring information for accounting and billing. One approach here would be to provide a central SLA based monitoring manager that could actually connect to all of the data being generated by the resource manager agents. This manager would then also be responsible for sending events to a variety of subscribers from the resource managers and schedulers and provisioners to the accounting and billing functions. Currently, there is a system under development at IBM Research, which provides just such an abstraction. The WSLA manager [SLMREF] is

configurable to connect to any data source for monitored data, as long as the interfaces to the data source are built, and they send data to any listener subscribing to SLA based events. Work with the customer would involve writing the correct interfaces for data and events for customer-specific systems.

4.1.3.4 Accounting and Billing

Accounting and billing are highly dependent on functions required to manage a utility such as monitoring. Fundamentally, data for accounting is based on usage metrics captured from monitoring subsystems. Monitoring systems tied to SLA management components will determine if a target level of service is being breached, however the same data can be tied into a usage accounting systems to correlate which application was running across which system for a given period of time.

Accounting functionality will continue to evolve over time as the probes become less intrusive and as the measurement overhead is reduced. This will enable greater granularity in the data points that can be captured relative to application cycle counts over a period. Currently applications can be instantiated on a provisioned server and a trigger will "check-out" the server and begin the billing period for a particular user of a particular resource. Similarly this can also be accounted for through historical analysis of data such as system usage logs. Software exists today that can be configured to address this such as SAS IT Resource Manager [SASITRMREF], Apogee NetCountant [APOGEEREF], and Tivoli Decision Support for OS/390 [TDSREF].

Billing and chargeback is based on the information in accounting systems that include the usage as well as the cost models and mechanisms by which the cost is allocated. Billing systems have existed for many years and will not require major overhauls. However, these systems will need to undergo a transformation in terms of how they are used in chargeback schemas within IT providers to ensure that costs are properly allocated to the correct subscribers based on the usage data from accounting systems.

A variety of products may be available for data collection, post-processing of data, storage of data, and reporting and analysis of data. Ideally, tool standardization will simplify the processing and establish a minimum of reconciliation steps. The key tool requirements are invoicing and reporting. So a tool with a rich set of tabular and graphical reporting options for a variety of presentation methods, including intranet/internet pages is desirable. The design of the reports should be simple and easy to understand, auditable (if archived for a predetermined period of time to meet audit requirements), repeatable, adaptable, and automation-oriented.

Measurement data is usually available from several existing sources. For instance in the OS/390 environment, tools already in place will likely be SMF (system management facility), DCOLLECT, online transaction monitors and database logging. For the UNIX and LAN/WAN environments, tools that may already be in place which provide measurement data, are CA-Unicenter [CAREF], HP OpenView [HPOVREF] or BMC Patrol [BMCREF]. The consideration will be requirements for these tools to interface with or feed data to the IT Accounting software.

The IT Accounting or Chargeback specific products that are commonly used are:

- SAS IT Service Vision and IT Charge Manager
- CA-MICS (NeuMICS) Accounting and Chargeback [CAREF]
- CIMS [CIMSREF]
- Tivoli Decision Support for OS/390 Accounting and Accounting WorkStation
- Apogee NetCountant
- Ejacent Micromeasure [EJACENTREF]
- ADC Singl.eView [ADCREF]

4.1.3.5 Federated Data

We are concerned with two types of data federation in our scenario. First, the HPC analytics application could require shared data via a file system. For this use case, we are concerned with cluster file systems. In the Linux cluster market IBM's own General Parallel File System (GPFS) [SCHMUCK2002] is state of the art, and cheapest. GPFS is also being extended to work over WAN so that multi cluster support is available. We note that several customers who are looking at using HPC applications within SOA are also looking to save the state between job execution cycles, i.e., intermediary computational results in a type of distributed cache. Currently work is underway to investigate GPFS as the basis of a distributed cache [DSO2003]. No solution exists in the marketplace today for

the distributed cache in a cluster problem that performs well enough and is general enough to be part of an HPC solution.

The second type of data federation deals with database type storage, so that data across tables and even physical volumes can be shared. In this arena, much of the technology is still underdevelopment, but IBM does have a significant existing product with DB2 Information Integrator [DB2IIREF].

4.1.3.6 Network Provisioning

As noted in section 4.1.3.2, policy-based resource sharing ties together a series of management components in order to reallocate cluster/server resources and/or application components or workload. Network provisioning couples policy based resource sharing with an underlying dynamic connectivity environment to perform both topology reconfiguration and policy reconfiguration.

Intelligent networks today allow for the dynamic construction and deconstruction of private VLANs to create logical segments. In addition, VPNs created across wide areas can now be used to secure bridges between private segments and allow collaboration across a public network. These services can be set up as required by network administrators to control traffic flow across a private and public network and in our scenario could be used to build virtual Linux clusters for different services which is more time and cost effective than creating clusters on truly distinct LANS. However, these services alone do not provide the dynamic provisioning required to ensure that the QoS requirements for application connectivity are met.

Congestion management is a more recent form of network provisioning as firms begin performing QoS management using tools from firms like Sitara Networks or Packeteer to manage congestion. These tools typically address the scenario where a firm knows it has more traffic than it can send over the current network, and therefore needs to control, which users or applications are given priority and which are denied service based on management policy. Networks tend to discard excess (above a certain threshold) traffic. QoS management systems enable the network to make decisions based on (QoS) reporting on how policies are being adhered to. Typically these systems provide a probe inline between the WAN and border routers to manage congestion. Products like those from Sitara [SITARAREF] or Packeteer [PACKETEERREF] provide devices or probes that measure QoS to rearrange traffic. They manage defined traffic flows and tell the administrator about unknown traffic flows in addition to ones that are configured. This is reliant on a well-specified input in terms of policy and traffic flow analysis.

Over time it is expected that QoS and advanced reservation protocols will become more commonplace and embedded within vendor hardware and operating systems in order to ensure that applications can be provided connectivity at a certain guaranteed level of service. In the meantime, point solutions exist that can be coupled together in order to begin orchestrating allocation of bandwidth to applications.

4.1.3.7 Capacity Planning

As stated previously, capacity planning for On Demand SOA should be services based. This means that we will need dependency analysis and corresponding IT based topology maps for each service. In our scenario, we will need to understand the request path or server topology for each of the services. For instance, we will need to know which servers and how these servers are connected (i.e. application server and database server) are required for an "Add dependent medical coverage" service. eWLM provides the required dependency map. However, in order for requests within a service to be traced with eWLM, the application and/or associated middleware should be instrumented with ARM calls. For some applications, this is not possible. New technology for dynamic dependency analysis that does not require such instrumentation is under development [EBIZDEPREF]. Even this technology has some limitations, however, since it requires the application to be written in Java. For non-Java applications, building dependency and topology maps can be much more difficult, and this is where experience in analyzing existing application logging is crucial. Holosofx [HOLOSFXREF] can be quite useful in building these dependency graphs by hand.

Once the topology and dependencies are known for one of the services, we would specify a certain QoS (i.e. throughput or response time) and generate an appropriate system configuration to support that QoS. Combining these capacity requirements in an optimization problem and doing what if scenarios will be crucial to the advanced reservation problem for any policy based resource manager as well as for dynamic provisioning or resources. In our scenario, we would use such tooling to configure the system one way for a time of day where we expect a heavy

customer load on the value at risk calculations, perhaps allocating more Linux servers to the advice engine application. Then we could re-configure the system to give more capacity to the other Linux applications during off-peak hours. Sophisticated capacity planning and optimization models are clearly required for these scenarios. An example of such technology for this advanced tooling combining request topology, use prediction, performance analysis, and resource optimization can be found in [CRAWFORD2002].

5 Conclusions

The value of the On Demand SOA is the ability to build once, use often. One place to make a change is to bring encapsulation from the object world, where the reuse impact is small, to the enterprise world, where the impact is significant. Interface by contract will enable the loosely coupled requestors and providers, so each can vary independently. Integration is explicitly defined and better understood at the application and enterprise level. We will have few, large-grained interactions, each commonly agreed upon, which again promotes looser coupling as small-grained state and process models are independent. Less variation and complexity to manage is the result. Easier to change large-grained processes are engaged if individual steps are well-defined and less complex Flexibility is the result with time to market, and speed is a natural outcome. In addition, on demand SOA will lower development, operations, and maintenance costs.

In this paper we used a real customer scenario to demonstrate how on demand SOA allows us to define business process explicitly, and separate it from definition of individual steps, allowing multiple processes to share the same implementation of common individual steps. This allows processes and implementations to vary independently. We also showed how to enable the processes and the implementations, both in the application and system areas, by using existing technology and standards.

The Internet revolutionized the way people talk to computers and systems. For customers, new business models were created or existing opportunities were extended. For employees, the benefits are new transparency and improved collaboration. For IT departments, a dramatic reduction in infrastructure costs and complexity in deploying IT solutions. The key was a universal server-to-client model based on simple open standards and industry support.

On demand SOA promises to do the same thing for the way systems talk to systems. The ability to integrate one business directly with another without waiting for people is key to flexibility and speed to market. The ability to do this within the enterprise promises to provide IT organizations a new programming model where composers and service assemblers are introduced as new roles in addition to the developer role.

Ultimately, this paper has been about framing a foundation for the way in which we engage with customers focusing on a consistent architecture and organizational framework.

6 Acknowledgments

We are grateful to Rhonda Childress, Daron Green, Mark Ernest, Cary Perkins, and Chris Reech for their participation in the initial brainstorming sessions which were the basis for this paper. We are especially appreciative to Paul Magnone for posing this problem to us in the first place and to Scott Penberthy for providing financial support for the work encapsulated in this paper. CHC is grateful for the constant management support of Nagui Halim, Dan Dias and Asit Dan and the funding support from Dave Turek.

7 References

[HAGEL2002] J. Hagel III, *Out of the Box, Strategies for Achieving Profits Today and Growth Tomorrow through Web Services*, Harvard Business School Press, 2002, p. 15.

[WSLA] H. Ludwig, A. Keller, A. Dan and R. King, "A Service Level Agreement for Dynamic Electronic Services", in WECWIS, June 2002.

[GTKV2] The Globus Alliance. Globus Tool Kit 2.0, 2002. http://www.globus.org/gt2.4/download.html

[GTKV3] The Globus Alliance. Globus Tool Kit 3.0, June 2003. http://www-unix.globus.org/toolkit/download.html

[AMAN97] J. Aman, C.K. Eilert , D. Emmes, P. Yocom and D. Dillenberger, "Adaptive algorithms for managing a distributed data processing workload", IBM Systems Journal, 36(2), 242-283, 1997.

[LEYMANN2002] F.Leymann,D.Roller, "Business processes in a Web services world – A quick overview of BPEL4WS", IBM, August 2002, also at http://www-106.ibm.com/developerworks/library/ws-bpelwp/
[WASREF] IBM. WebSphere v.5.1, November 2002. http://www.ibm.com/websphere.
[BEAREF] BEA. BEA WebLogic Server 8.1, March 2003. http://www.bea.com/products/weblogic/server/index.shtml
[TCREF] The Apache Software Foundation. Tomcat 4.1.29, October 2003. http://jakarta.apache.org/tomcat/
[GRAMREF] The Globus Alliance. Globus Toolkit - GRAM v3, July 2003. http://www-unix.globus.org/developer/resource-management.html
[DSREF] DataSynapse. Grid Server 3.2, June 2003. http://www.datasynapse.com/solutions/gridserver.html
[EUTOPIAREF] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, A. Youssef, "Performance Management for Cluster Based Web Services", Proceedings of 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003), Colorado Springs, Colorado, March 2003.
[SYMPHONYREF] Platform Computing. Platform Symphony 1.5, March 2003. http://www.platform.com/products/Symphony/symphony.asp
[CSFREF] Platform Computing. Platform Community Scheduler Framework (CSF), November 2003. http://www.platform.com/products/Globus/
[LSFREF] Platform Computing. Platform LSF 5.1, February 2003. http://www.platform.com/products/LSF/
[CONDORREF] University of Wisconsin-Madison (UW-Madison). Condor Version 6.6.0, 2003. http://www.cs.wisc.edu/condor/
[PBSREF] NASA. OpenPBS, early-mid 90's. http://www.openpbs.org/main.html
[LLREF] IBM. LoadLeveler Version 3, Release 2, October 2003. http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/loadleveler.html
[POEREF] IBM Parallel Operating Environment. IBM Parallel Environment for AIX (PE), http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/pe.html
[SLMREF]A.Dan, D.Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, A. Youssef, "Web services on demand: WSLA-driven automated management", IBM Systems Journal, 43(1), 2004.
[SASITRMREF] SAS. SAS IT Resource Management Version 2.6, 2003. http://support.sas.com/documentation/onlinedoc/itsv/
[APOGEEREF] Evident Software (former ApogeeNetworks). NetCountant. http://www.evidentsoftware.com/ITN02042002_1.html
[TDSREF] IBM. Tivoli Decision Support for OS390 Version 1.6. June 2003. http://gwareview.boulder.ibm.com/software/tivoli/products/tds-390/
[CAREF] Computer Associates International, Inc. Unicenter NeuMICS Resource Management Analyzer Option for MQSeries, 2002. http://www3.ca.com/Solutions/ProductOption.asp?ID=1404
[HPOVREF] Hewlett-Packard. HP OpenView products (multiple vertions and release dates) . http://www.openview.hp.com/
[BMCREF] BMCSoftware. PATROL® Enterprise Manager (PATROL EM) environment. http://www.bmc.com/products/proddocview/0,2832,19052_19429_23191_7266,00.html
[CIMSREF] CIMS CIMS Lab, Inc. develops, markets and supports the CIMS suite of Resource Accounting, Chargeback, and Capacity Planning software products, initially established in Walnut Creek, California in 1974. http://www.cimslab.com/default.htm
[EJACENTREF] Ejacent Micromeasure, A comprehensive data center usage data aggregation, reporting, and analysis application that provides visibility and alignment of IT to the business objectives by enabling usage based reporting, cost allocation and charge-back of IT assets. http://www.ejasent.com/products/micromeasure.html
[ADCREF] ADC. Singl.eView Billing and Commerce v.5.01, June 2003. http://www.adc.com/software/billingandcommerce/
[SCHMUCK2002] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters", in Proceedings of the Conference of File and Storage Technologies (FAST'02), 28-30 January 2002, Monterey, CA, pp. 231-244.
[DSO2003] C.H. Crawford and K. Gildea, "Building a Distributed Shared Object Cache on top of a Parallel File System", IBM Corporation, In Progress, 2003.
[DB2II] IBM Corporation. DB2 Information Integrator. http://www-306.ibm.com/software/data/integration/db2ii/
[SITARAREF] Sitara Networks. QoSWorks AOSv2.0, October 2002. http://www.sitaranetworks.com/products/QoSWorks/

[PACKETEERREF] Packeteer , Application Performance Solutions for the WAN and Internet, http://www.packeteer.com/

[EBIZDEPREF] M. Gupta, A. Neogi, M Agarwal, G. Kar Discovering Dynamic Dependencies in Enterprise Environments for Problem Determination. IEEE Distributed Systems: Operations and Management, October, 2003, Heidelberg, Ge rmany.

[HOLOSFXREF] Holosfx, WebSphere Business Integration Modeler and Monitor (formerly Holosfx) provide software tools to help you model, simulate, analyze, automate, and monitor complex business processes quickly and effectively.http://www-306.ibm.com/ software/integration/wbimonitor/

[CRAWFORD2002] C.H. Crawford and A. Dan, "eModel: Addressing the Need for a Flexible Modeling Framework in Autonomic Computing", in Proceedings of MASCOTS2002, October 2002, Dallas, TX.