CLOUD COMPUTING

A COLLECTION OF WORKING PAPERS

THOMAS B WINANS AND JOHN SEELY BROWN

MAY 2009

Demystifying Clouds • Moving Information Technology Platforms To The Clouds • Motivation To Leverage Cloud And Service Grid Technologies

Cloud Computing frequently is taken to be a term that simply renames common technologies and techniques that we have come to know in IT. It may be interpreted to mean *data center hosting* and then subsequently dismissed without catching the improvements to hosting called *utility computing* that permit near realtime, policy-based control of computing resources. Or it may be interpreted to mean *only data center hosting* rather than understood to be the significant shift in Internet application architecture that it is.

Perhaps it is the name. Certainly it is more *nebulous* than *mnemonic*, if you'll pardon the poor pun. We happen to think so too. We'd rather use the term *service grid*, frankly, but that name also has its problems. The fact is that *cloud* and *service grid computing* are paradigmatically different from their common interpretations, and their use can shed light on how internet architectures are constructed and managed.

Cloud computing represents a different way to architect and remotely manage computing resources. One has only to establish an account with Microsoft or Amazon or Google to begin building and deploying application systems into a cloud. These systems can be, but certainly are not restricted to being, simplistic. They can be web applications that require only http services. They might require a relational database. They might require web service infrastructure and message queues. There might be need to interoperate with CRM or e-commerce application services, necessitating construction of a custom technology stack to deploy into the cloud if these services are not already provided there. They might require the use of new types of persistent storage that might never have to be replicated because the new storage technologies build in required reliability. They might require the remote hosting and use of custom or 3rd party software systems. And they might require the capability to programmatically increase or decrease computing resources as a function of business intelligence about resource demand using virtualization. While not all of these capabilities exist in today's clouds, nor are all that do exist fully automated, a good portion of them *can* be provisioned.

Are the services that are provided by the cloud robust in the enterprise sense?

Absolutely … especially if you mean *the enterprise as we know it today*. While there are important security, privacy and regulatory issues that enterprises need to sort through before full migration to the cloud, and cloud vendors need to strengthen cloud capabilities in these areas before enterprise applications can be effectively hosted in the cloud, there are benefits that cloud technologies *do* offer today that can be leveraged in the deployment of many enterprise applications. Further, cloud vendors are likely to offer *virtual private cloud* functionality that quite possibly will (at least temporarily) compensate for current deficiencies as cloud vendors work to address them in a more strategic and long-term way.

Migration to the cloud will not be immediate because of issues noted above, and because enterprises need to approach migration in a staged fashion, just as they would undertake any significant technology transition. The first stage of migration is to determine what enterprise infrastructure and applications can be reliably rearchitected using cloud computing technologies today to gain experience with a cloud-oriented way of organizing and accessing digital technology. This stage may include development of a migration path that progressively transitions more of the enterprise infrastructure and applications to cloud providers as they evolve more robust services that can support a broader range of enterprise IT activities. The key goal of this stage is to define the roadmap to replicate what is available on premise today using cloud technologies (public or private) where this makes sense, and to define fundamentals that will guide future architecture efforts. The second stage begins a period in which explicitly policy based architectures that help to support agility and innovation are designed. The third stage is the period in which implementation of these fundamentally new architectures – that are designed to support scalable networks of relationships across large numbers of very diverse and independent entities (i.e., possibly leveraging a more fully developed service grid) – takes place.

It is critical to give explicit attention to architecture when preparing to migrate to the cloud, since this represents an opportunity for corporations to rearchitect themselves as *next generation enterprises*. These globalized and distributed enterprises must scale process and practice networks (ultimately

comprising an entire ecosystem) to include thousands to tens of thousands of members, with the number of users increasing to the millions. This type of scale requires an architecture and interoperability strategy modulated by *harmonized technology and business policies* to scale business elastically. Widespread adoption of clouds as a computing platform will require interoperability between clouds and management of resources in one cloud by another. Since current-day architectures are not structured to externalize policy, the typical architecture fundamentals of applications that enterprises deploy must be modified to effectively use and exploit new cloud capabilities.

In this context, we urge executives to develop more explicit cloud computing strategies based on a more explicit and longer-term view of the likely trajectories of cloud computing evolution. There are undoubtedly compelling benefits to participating in clouds today. But the real power of cloud computing platforms stems from the potential over time to re-think and re-design IT architectures at a more fundamental level. Companies that gain early experience with this new infrastructure will be in the best position to harness these new architectural approaches to re-shape the broader business landscape.

To seed this effort, we cull from various candidate definitions for both *cloud* and *service grid computing* a set of concepts to be used as a thoughtful framework of discussion for what happens next in Internet-based computing. We present, here, a set of three papers that discuss:

- Characteristics of what we believe to be next generation architectures that will support substantive changes in global enterprise constructs and operations;
- Transformation from existing to next generation architectures to simplify the architectures and better align them with the businesses they enable, and provide the means to externalize and manage policy across all architecture layers; and
- Pain points that might be eliminated altogether by migration to next generation architectures.

We understand that cloud and service grid computing, in their present state, do not meet all distributed computing or enterprise needs. However, they *do* meet many of them in a way that will provide a smooth transition to whatever next generation distributed computing becomes, and they already are significantly helpful in modulating the technology changes that enterprises face today. The rapid pace at which cloud vendors are systematizing their platforms and attracting stalwart industry supporters and users confirms that ecosystems are forming that are based upon the capabilities that cloud computing enables. The speed with which they are forming strongly suggests that cloud computing will not only meet many of the needs of enterprise computing as we have come to know it, but also could form the digital platform for a shaping strategy[1] guiding next generation enterprises in their migration to and participation in such ecosystems. Hence our optimism of things to come, and our contribution to a discussion that we hope readers will find helpful.

[1] Shaping Strategy in a World of Constant Disruption, Harvard Business Review, by John Hagel III, John Seely Brown, and Lang Davison, Oct 2008

TABLE OF CONTENTS

DEMYSTIFYING CLOUDS: Exploring Cloud and Service Grid Architectures

MOVING INFORMATION TECHNOLOGY PLATFORMS TO THE CLOUDS: Insights Into IT Platform Architecture Transformation

MOTIVATION TO LEVERAGE CLOUD AND SERVICE GRID TECHNOLOGIES: Pain Points That Cloud And Service Grids Address

DEMYSTIFYING CLOUDS

EXPLORING CLOUD AND SERVICE GRID ARCHITECTURES

THOMAS B WINANS AND JOHN SEELY BROWN

26 MAY 2009

INTRODUCTION

Cloud Computing is in vogue. But what is it? Is it *just* the same thing as *outsourcing the hosting of Web applications*? Why might it be useful and to whom? How does it change the future of enterprise architectures? How might clouds form the backbone of twenty-first-century ecosystems, virtual organizations and, for a particular example, healthcare systems that are truly open, scalable, heterogeneous and capable of supporting the players/providers *both big and small*? In the past, IT architectures took aim at the enterprise as their endpoint. Perhaps now we must radically raise the bar by implementing architectures capable of supporting entire ecosystems and, in so doing, enable these architectures to scale both downward to an enterprise architecture as well as upward and outward.

We see cloud computing offerings today that are suitable to host enterprise architectures. But while these offerings provide clear benefit to corporations by providing capabilities complementary to what they have, the fact that they can help to elastically scale enterprise architectures should not be understood to also mean that simply scaling in this way will meet twenty-first-century computing requirements. The architecture requirements of large platforms like social networks are *radically* different from the requirements of a healthcare platform in which geographically and corporately distributed care providers, medical devices, patients, insurance providers, clinics, coders, and billing staff contribute information to patient charts according to care programs, quality of service and HIPAA constraints. And the requirements for both of these are very different than those that provision *straight-through processing* services common in the financial services industry. Clouds will have to accommodate differences in architecture requirements like those implied here, as well as those relating to characteristics we subsequently discuss.

In this paper, we want to revisit *autonomic computing*, which defines a set of architectural characteristics to manage systems where complexity is increasing but must be managed without increasing costs or the size of the management team, where a system must be quickly adaptable to new technologies integrated to it, and where a system must be extensible from within a corporation out to the broader ecosystem and vice versa. The primary goal of autonomic computing is that "systems manage themselves according to an administrator's goals. New components integrate … effortlessly ..."[i]. *Autonomic computing* per se may have been viewed negatively in the past years – possibly due to its biological metaphor or the *AI* or *magic-happens-here* feel of most autonomic initiatives. But innovations in cloud computing in the areas of virtualization and finer-grained, container-based management interfaces, as well as those in hardware and software, are demonstrating that the goals of autonomic computing can be realized to a practical degree, and that they could be useful in developing cloud architectures capable of sustaining and supporting ecosystem-scaled use.

Taking an autonomic approach permits us to identify core components of an autonomic computing architecture that Cloud Computing instantiations have thus far placed little emphasis on. We identify technical characteristics below that must not be overlooked in future architectures, and we elaborate them more fully later in this paper:

- An architecture style (or styles) that should be used when implementing cloud-based services

- External user and access control management that enables roles and related responsibilities that serve as interface definitions that control access to and orchestrate across business functionality
- An Interaction Container that encapsulates the infrastructure services and policy management necessary to provision interactions
- An externalized policy management engine that ensures that interactions conform to regulatory, business partner, and infrastructure policy constraints
- Utility Computing capabilities necessary to manage and scale cloud-oriented platforms

AN AUTONOMIC FRAME OF MIND

Since a widely accepted industry definition of *Cloud Computing* - beyond a relationship to the Internet and Internet technologies – does not exist at present, we see the term used to mean *hosting of hardware in an external data center* (sometimes called *infrastructure as a service*), utility computing (which packages computing resources so they can be used as a utility in an always-on, metered, and elastically scalable way), platform services (sometimes called middleware as a service), and application hosting (sometimes called software or applications as a service). All of these ways seem – in some way – right, but they are not helpful to understand the topology of a cloud, the impact that Cloud Computing will have on deployment of business platforms, whether or not the business system architecture being deployed in commercial or private data centers today will be effective in a cloud, or what architectures should be implemented for cloud-based computing. Neither do they even begin to get at the challenge of managing very large and dynamic organizations, called *virtual organizations* (to be defined later in this paper), that reorient thinking about the need for an architecture to scale massively, and the need to make parts of an architecture public that, to this point, have been kept private.

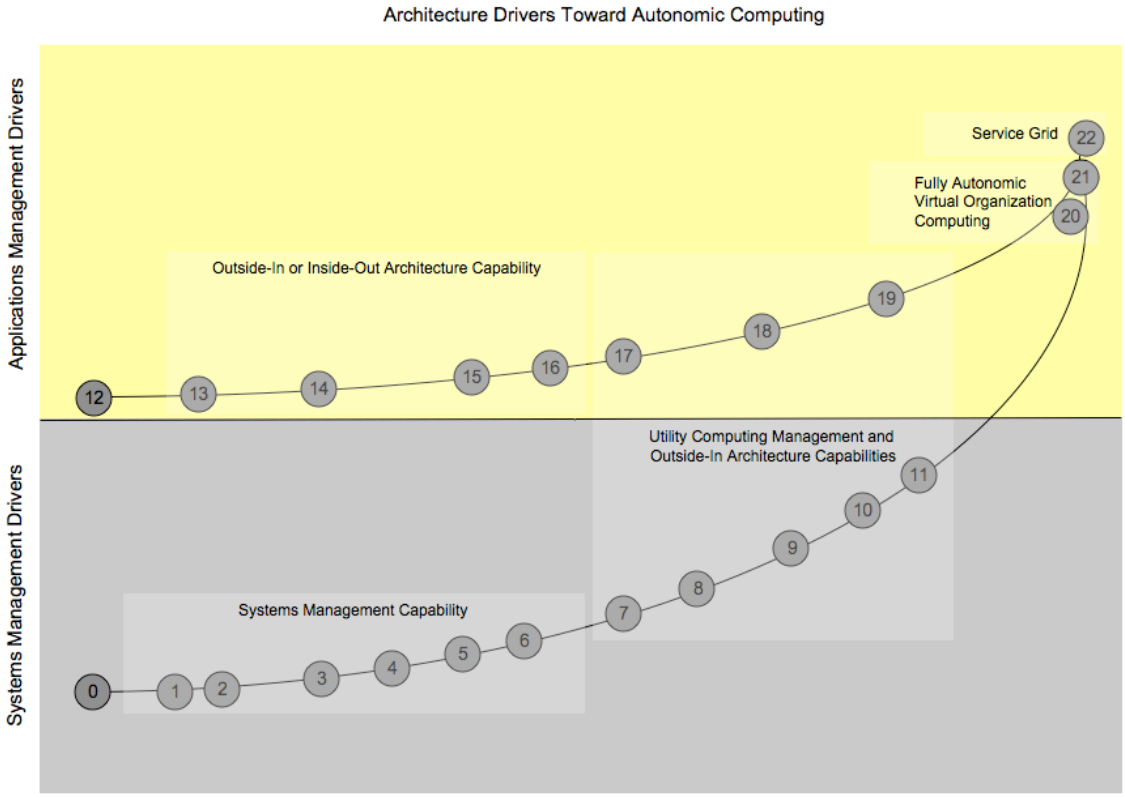
To satisfy the requirements of next century computing, cloud computing will need to mean more than just externalized data centers and hosting models. Although architectures that we deploy in data centers today should be able to run in a cloud*,* simply moving them into a cloud stops well short of what one might hope that Cloud Computing will come to mean. In fact, tackling global-scaled collaboration and trading partner network problems in government, military, scientific, and business contexts will require more than what current architectures can readily support. For example:

- It will be necessary to rapidly set up a temporary collaboration network enabling network members to securely interact online, where interaction could imply interoperability with back office systems as well as human-oriented exchanges – all in a matter of hours. Examples that come to mind include emergency medical scenarios, global supply chains and other business process networks. Policies defining infrastructure and business constraints will be varied, so policy must be external to, and must interact with, deployed functionality. These examples also imply the need for interoperability between public and private clouds.
- Business interactions have the potential to become more complex than personal transactions. Because they are likely to be formed as composite services, and because services on which they depend may be provisioned in multiple clouds, the ability to provision and uniformly manage composite cloud services will be required, as will be the ability to ensure that these services satisfy specified business policy constraints.
- The way that users and access control are managed in typical applications today is no longer flexible enough to express roles and responsibilities that people will play in next-generation business interactions. Roles will be played by people outside of or across corporate boundaries in an online context just as frequently as they are inside. Access control and the management of roles and responsibilities must be externalized from business functionality so that it becomes more feasible to composite functional behavior into distributed service-oriented applications that can be governed by externalized policy.

These considerations suggest that clouds will have to have *at least* the following characteristics[ii]:

- Clouds should be uniquely identifiable so that they can be individually managed even when combined with other clouds. This will be necessary to distinguish and harmonize cloud business and infrastructure policies in force.
- A cloud should be dynamically configurable: configuration should be automatable in varying and unpredictable, possibly even event-driven, conditions.
- Systems management technologies for clouds must integrate constraints on business with constraints on infrastructure to make them manageable in aggregate.
    - A cloud should be able to dynamically provision itself and optimize its own construction and resource consumption over time.
    - A cloud must be able to recover from routine and extraordinary events that might cause some or all of its parts to malfunction.
    - A cloud must be aware of the contexts in which it is used so that cloud contents can behave accordingly. For example, if clouds are composited, policy will have to be harmonized across cloud boundaries; when in multitenant mode, service level agreements may be used to determine priority access to physical resources. Application platforms today are unaware of their usage context, but business functionality in next-generation platforms will have to be managed with context in mind.
- A cloud must be secure, and it must be able to secure itself.

These coarse-grained characteristics, sometimes described as *autonomic computing,* can be represented in the form of finer-grained architecture drivers that are useful in characterizing steps *toward* an autonomic computing architecture. Cloud Computing offerings that are available today share many of the same drivers that we have organized into *Systems* and *Application Management Drivers* in the figure below.



Numbered circles in the graphic above denote drivers that are listed below:

0. Architecture state: no systems management
1. Systems and resources must be identifiable
2. System and resources must be manageable
3. Policy-driven secured access to the system and managed resources must be provided
4. System must reallocate managed resources on failures as a function of policy
5. System must reallocate managed resources on various system-level conditions by policy
6. System must be managed lights-out in a single data center context
7. Systems management capability must scale across clouds of the same type
8. Systems management capability must scale across clouds of different types; these clouds must be managed uniformly while maintaining separate cloud identities
9. System must reallocate managed resources on various system-level conditions as a function of policy to accommodate real-time and business-oriented usage patterns
10. Systems management policies are harmonized across cloud boundaries
11. It must be possible to integrate management policies of different clouds
12. Monolithic applications and traditional application integrations exist/are sufficient
13. Application platform must be service oriented
14. Applications are replaced with business services
15. Business services have secured access
16. An Interaction Container[1] must be used as *application container* in a single-tenant environment
17. Policies must be consolidated and managed using a single (possibly federated) policy engine
18. System must reallocate managed business services on various business-level conditions by policy to accommodate real-time/batch usage patterns
19. An Interaction Container must be used as *application container* in a multitenant environment
20. Business service and systems management policies are integrated
21. Architecture state: positioned as an autonomic architecture platform for virtual organization-oriented application systems
22. Architecture state: additional structural and business constraints positioning architecture platform as a service grid

[1] Defined in the next section

The graphic shows two paths toward autonomic computing that ultimately converge at an architecture point that could support business ecosystems and emergent and fluid virtual organizations:

- The first path, *Systems Management Drivers*, begins with no systems management, and ends with a systems management capability that is policy driven, and that enables automated systems management in a cloud and harmonization of business and infrastructure policies within and across cloud boundaries – in both single- and multi-tenant modes. The drivers for systems management are grouped to illustrate needs common to basic systems management (Systems Management Capabilities), and needs that go beyond basic capabilities (Utility Computing Management and Outside-In Architecture[iii] Capabilities).

- The second path, *Applications Management Drivers*, begins with common monolithic corporate applications. It ends with these applications having been replaced with service-oriented ones, where policy has been externalized so that business policies can be harmonized with utility management policies, where it is possible to implement end-to-end service level agreements and enforce conformance to business and regulatory constraints, and where the use of business functional and infrastructural components can be metered and elastically load balanced. At this endpoint, business services and infrastructure can be organized into a cloud and used in both single- and multitenant modes.

Systems and Applications Management Drivers paths converge at the point where it is necessary to manage both the business and the infrastructure using common management capabilities, and where related policies *must* be harmonized.

Presenting drivers on paths is sometimes risky, as such suggests a linear progression toward implementing an ultimate architecture, or gives preference to one suggested architecture vision over another. Neither is meant in this case. In fact, one can view how far one traverses each path as one of architecture need over a perceived architecture maturity. To underscore, we make the following observations relating to commercially available cloud computing products:

- Cloud computing does not realize the goals of autonomic computing as they are defined currently, though combining the characteristics of existing clouds gets closer to this goal. This fact does not diminish their value for optimizing deployments of applications in place today.
- Not every cloud needs to be autonomic – but there are benefits along each path regardless.
    - Implementing architecture features on the Applications Management Drivers path will lead to optimizing costs of operating and maintaining infrastructure and business functionality that currently run a business, and automating systems management, resulting in more efficient data center management.
    - Evolving an architecture toward Utility Computing Management *and* Outside-In Architecture Capabilities will help organizations expand their IT systems beyond corporate boundaries. This supports implementation of more flexible partner networks and value chains, but it also can scale to serve virtual organizations.

CHARACTERISTICS OF AN AUTONOMIC SERVICE ARCHITECTURE

As cloud computing solutions and products are implemented, we believe it critical – especially to those being driven by their business needs up the Systems and Applications Management Drivers curves – to carefully consider their need for support of the architecture characteristics that we sketched in the opening part of this paper and that we now elaborate.

ARCHITECTURE STYLE

Architecture styles define families of software systems in terms of patterns for characterizing how architecture components interact. They define what types of architecture components can exist in architectures of those styles, and constraints on how they may be combined. They define how components may be combined together for deployment. They define how units of work are managed, e.g., whether or not they are transactional (n-phase commit). And they define how functionality that components provision may be composited into higher order functionality, and how such can be exposed for use by human beings or other systems.

The *Outside-In* architectural style is inherently top-down and emphasizes decomposition to the functional level, but not lower; is service-oriented rather than application-oriented; factors out policy as a first-class architecture component that can be used to govern transparent performance of service-related tasks; and emphasizes the ability to adapt performance to user/business needs without having to consider the intricacies of architecture workings[2].

The counter style*,* what we call *inside-out,* is inherently bottom-up and takes much more of an infrastructural point of view as a starting point, building up to a business functional layer. Application platforms constructed using *client server*, *object-oriented*, and *2/3/n-tier* architecture styles are those to which we apply the generalization *inside-out* because they form the basis of enterprise application architectures today, and because architectures of these types have limitations that require transformation to scale in a massive way vis-à-vis outside-in platforms (see Web Services 2.0 for a more detailed discussion of both Outside-In and Inside-Out architecture styles).

Implementation of an outside-in architecture results in better architecture layering and factoring, and interfaces that become more *business* than *data* oriented. Policy becomes more explicit and is exposed in a way that makes it easier to change it as necessary. Service orientation guides the implementation, making it more feasible to integrate and interoperate using commodity infrastructure rather than using complex and inflexible application integration middleware.

As a rule, it is simpler to integrate businesses at functional levels than at lower technology layers where implementations might vary widely. Hence we emphasize decomposition to the functional level, which often is dictated by standards within a market, regulatory constraints on that market, or even accounting (AP/AR/GL) practices.

Architecture style will be critical to orchestrating services and enabling operability between thousands of collaborating businesses. The Li & Fung Group manages supply chains involving over 10,000 companies located in over 40 countries of the world. Point integration solutions are infeasible at this scale. Similarly, attempts to integrate hundreds of hospital patient management systems and devices into a healthcare cloud, replete with HL7 variants and new and legacy applications, would result in the same conclusion that interoperability must be realized through the implementation of an architecture that integrates at a business functional level rather than a data level.

EXTERNAL USER AND ACCESS CONTROL MANAGEMENT

User and access control management *usually* is implemented within a typical enterprise application. A user is assigned one to many application roles, and a role names a set of privileges that correlate to use of particular application functionality through a graphical user interface, or through some programming interface. User authentication and authorization can be integrated with corporate identity management solutions (e.g., single sign-on solutions) that are in place to ensure that only people within a corporation or corporate partner network are permitted to use corporate applications.

[2] An outside-in architecture is a kind of service-oriented architecture (SOA) which is fully elaborated in Thomas Erl's book called "Service-Oriented Architecture: Concepts, Technology, and Design," so we will not discuss SOA in detail in this paper.

But as businesses globalize and couple more fluidly and dynamically, the management of users and their assignments to roles and responsibilities/privileges must be implemented in a scalable fashion that supports composition of services into more complex service-oriented behavior. Further, it must be possible for role players to transparently change in response to business- and partner-related changes made over time, especially in business interactions that could be in progress over months to years.

A fundamental part of user management is *identity management*. There are numerous identity solutions available today from vendors like Microsoft, Sun Microsystems, and Oracle. The challenges facing these solution vendors include their ability to manage the varied ways a user can be represented in an online context, the means to verify identity and detect and manage identity theft, the need to accommodate audits of transactions and interactions conducted with a specific identity, and so forth. Identity Management is *much* larger than any single cloud or software vendor, and forming a solution for the twenty-first-century is even likely to require help from national governments[iv].

INTERACTION CONTAINER

The J2EE/Java EE community introduced the notion of container to the enterprise architecture community as a means to streamline the structure of thin java clients. Normally, thin-client multitiered applications would require code to implement persistence, security, transaction management, resource pool management, directory, remote object networking, and object life cycle management services. The J2EE architecture introduced a *container* model that made this functionality available – transparently, in many cases - to business logic implemented as classes of behavior as long as it was implemented to conform to special (e.g., *bean*) interfaces, freeing developers to focus on implementing business functionality and not infrastructure – resulting in a standardized use of infrastructure services. Containers are hosted in application servers.

As we move toward service orientation, there is need for an analog to an application server that not only manages common infrastructure services but provides the infrastructure extension points for managing policy that is harmonized across technology and business functional stacks within a cloud. For the purpose of discussion here, we use the term *Interaction Server* to mean an architecture component that provides runtime services used by Interaction Containers (defined below) to manage the life cycle of multi-party business service interactions in both single- and multitenant contexts. Runtime services can include those similar to application services (e.g., like J2EE container services), but also services to manage policy (harmonization across architecture layers, policy enforcement, and policy exceptions), Interaction life cycle management, and even specialized collaboration services (e.g., event-based publish and subscribe capabilities, and services that bring together those people who are involved in business interactions).

We use the term *Interaction* to mean a service oriented multiparty business collaboration. An interaction can be viewed as an orchestration of business services where orchestration *flow* (not *workflow* in the typical enterprise application integration sense) is managed using externalized policy (please see Web Services 2.0 for a more detailed discussion on this topic). An Interaction is hosted within an Interaction Container (defined below), and orchestrates services provisioned in distributed contexts. Interaction life cycle events are used to trigger system behavior and enforce management policies and are published by the Interaction Server to subscribers.

Finally, we use the term *Interaction Container* as an analog to *J2EE/Java EE application container.* The Interaction Container is hosted in an Interaction Server, statically and dynamically configured to provide infrastructure and policy adjudication services that are specific to a business user's environment, integrated with systems management capabilities, and used to manage one-to-many Interactions and their life cycles. An Interaction Container essentially holds an execution context in which role players – people or systems participating in an Interaction and conforming to specific roles (interfaces) – interact to perform their parts in a business orchestration and manage exceptions and/or faults should they occur in the process.

An *Interaction Container* can be considered to be organizationally based (i.e., it can be used to manage many Interactions between a set of participants/role players over time), or outcome-based (in which only one Interaction would be performed). These two usage scenarios reflect the need to manage Interactions in a dynamic user community, where role players could change over time, and the need to manage an Interaction as a single possibly long-running business transaction.

EXTERNALIZED POLICY MANAGEMENT/POLICY ENGINE

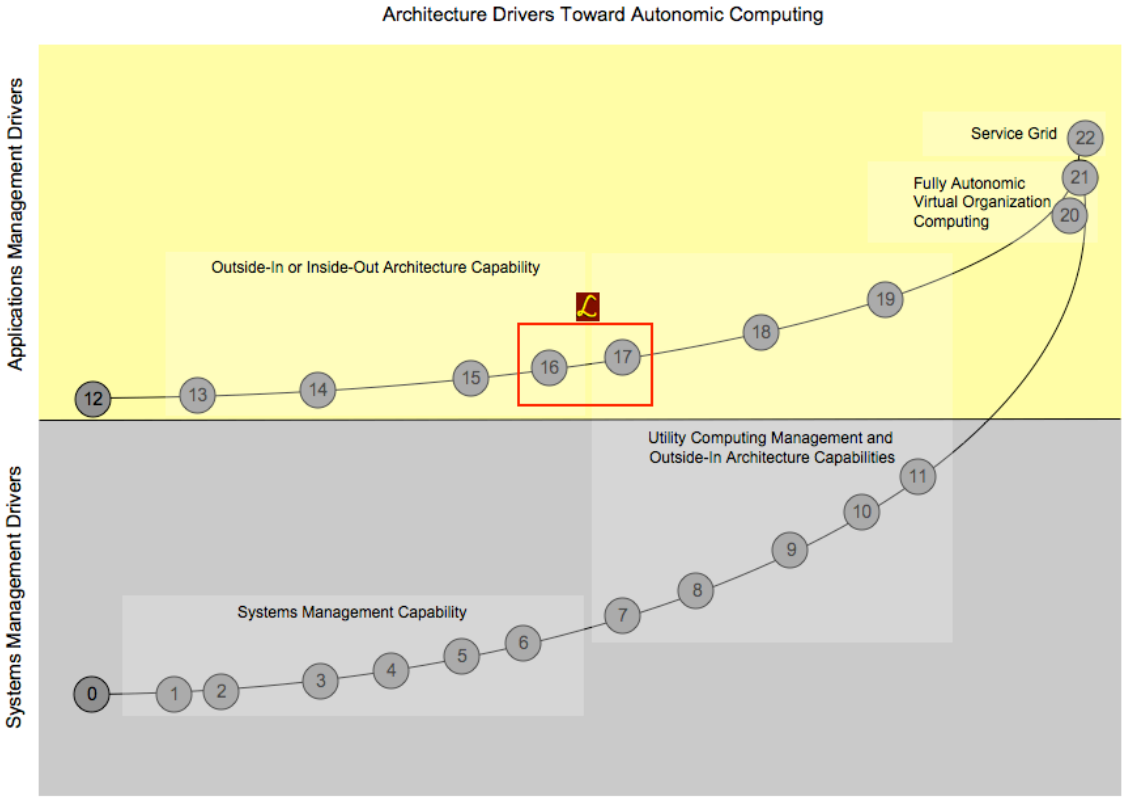
A Policy Engine harmonizes and adjudicates conflicting policies used across architecture layers. Components at all architecture layers can participate in policy harmonization and enforcement, which requires the following:

- Policy extension points must be exposed and formally declared in any part of the architecture that must be managed.
- Policy management must support *policy pushdown* to enable extensible and dynamic detection of policy violation and policy enforcement.
- It must be possible to version policy so that policy decisions made at a given time can be reproduced.
- Policy exceptions should be managed in as automated a fashion as possible, but support also must be given to cases where human judgment and decision making may be required. Note that *fault* or *exception* can connote both system-level occurrences *and* domain evolution in which policy constraints valid in the past become invalid. For example:
    - Inability to connect to a database is a system fault that should be automatically handled as a software system exception.
    - A regulatory constraint that permitted conduct of business in one way to a certain point in time, but that no longer does due to changes in law, is a business exception that may require human judgment to determine if completion of a business transaction according to old law should be permitted.

Policy embedded in application functionality is not easy to change, but future software systems will have to be implemented in a way that views change as the norm – where change results from the emergence of new markets, market evolution, changes in regulations and standards, fixing of policy bugs, the whims of interaction participants, and maybe even their customers' whims.

Externalizing policy highlights a significant distinction between Inside-Out and Outside-In architecture styles. Inside-Out architectures usually involve legacy applications in which policy *is* embedded and thus externalizing it is – at best – very difficult. Where application policies differ in typical corporate environments, it becomes the responsibility of integration middleware to implement policy adjudication logic that may work well to harmonize policies over small numbers of integrated systems, but this will not generalize to manage policy in larger numbers of applications as would be the case in larger value chains. The red rectangle in the figure below identifies where an Inside-Out architecture must transform (and simplify in the process) to become an Outside-In architecture, making it more feasible to externalize policy and progress toward the fully autonomic computing endpoint[3].

[3] In private conversations, we refer to this transformation as a kind of architectural Laplace Transform that helps in solving challenging problems with an architecture by transforming it to a simpler form by creating an alternative frame or point of view.



What Is Policy?

The word *policy* could have a number of meanings as it is used in conjunction with IT architecture and systems. For example, it could mean governance relating to software architecture development and implementation; or it could mean operational rules and standards for administering a deployed production system.

Our use of the word connotes constraints placed upon the business functionality of a business system, harmonized with constraints on the infrastructure (hardware and software) that provisions that functionality. These constraints could include accounting rules that businesses follow, role-based access control on business functionality, corporate policy about the maximum allowable hotel room rate that a nonexecutive employee could purchase when using an online reservation service, rules about peak business traffic that determine when a new virtualized image of an application system should be deployed, and the various infrastructural policies that might give customer *A* preference over customer *B* should critical resource contention require such.

Policy extension points, as noted above, provide the means by which policy constraints are exposed to business and corresponding infrastructural functionality and incorporated into their execution. They are not configuration points that are usually known in advance of when an application execution starts and that stay constant until the application restarts. Rather, policy extension points are dynamic and late bound to business and infrastructural functionality, and they provide the means to *dynamically shape* execution of this functionality. The sense of the word *shape* is consistent with how policy is applied in the telecom world where, for example, bandwidth might be made available to users during particular times in the day as a function of total number of users present. Just as policy is used in the telecom world to *shape* use of critical resources, policy can be used to shape execution of business functionality.

For example, suppose that an interaction between business partners is started by a partner located in a European country that legally requires all interaction data to remain in that country, whereas this same type of data could be stored anywhere that the deployment platform determines convenient otherwise. A policy extension point on storage could be exposed to ensure that storage systems located in the appropriate European country are used when required. Because policy is externalized as described above, this policy does *not* imply the need for multiple code bases to realize this constraint.

The example above is a simple one that one could imagine implementing at the application *business* layer of an enterprise architecture. Suppose, however, that this type of policy is moved from the business layer *into the network*. From 2005 to date, we have seen the emergence of XML accelerators (e.g., IBM/Data Power, Intel, Layer 7 Technologies) that make such possible by bringing to application protocol management what network protocol analyzers, or *sniffers*, bring to network protocol management. These accelerators are able to inspect, transform, and route both XML and binary data in ways that are conscious of ecosystem and interaction constraints, e.g., constraints like the European storage rule above. Once equipment like this is aware of the business data and the workflow context in which it is communicated, it can carry out networking functions such as forwarding, security, usage analysis, traffic management, and billing, in a much more sophisticated manner in comparison to traditional networking techniques – and it can do this taking into account policy constraints across an entire technology stack.

UTILITY COMPUTING

The raison d'être of autonomic computing is the need to address the growing complexity of IT systems. While loosely coupling architecture components makes them less brittle, it also exposes more moving parts that also must have management and configuration extension points. The authors of *The Vision of Autonomic Computing* worded their concerns over complexity as follows:

"As systems become more interconnected and diverse, architects are less able to anticipate and design interactions among components, leaving such issues to be dealt with at runtime. Soon systems will become too massive and complex for even the most skilled system integrators to install, configure, optimize, maintain, and merge. And there will be no way to make timely, decisive responses to the rapid stream of changing and conflicting demands."

Externalization of policy goes a long way toward making it possible to composite clouds and manage policy compliance. But the structure of the cloud also must be addressed if we expect to manageably scale a cloud. An autonomic computing architecture calls for architecture components to, themselves, be autonomic. This might sound a bit far-fetched unless we consider that we have been solving heterogeneity problems with abstraction layers at the operating system layer for some years now, and that this technique can be used again to manage large collections of computing resources uniformly. In particular, if two clouds are autonomic and essentially support the same management interfaces, then they could be composited into a larger cloud while preserving the identities of the original clouds. Intuitively, this simplifies scaling clouds and reconciling policy differences.

As we see the emergence of Cloud Computing products into the market, we see (at least) two that appear to recognize the need to composite clouds, grids, or meshes of manageable computing resources. Some cloud infrastructure vendors have taken an approach different from Amazon's in that they intend to deal directly with architecture components through a software abstraction layer. One approach taken to manage clouds is to provide a management interface to which all manageable resources, including the cloud itself, conform *so that* management over heterogeneous infrastructure is uniform. The approach that Microsoft has taken acknowledges a need for a uniform management abstraction layer, which it achieves by requiring that the set of manageable resources conform to interfaces in the .NET Framework. In either case, exposing a cloud *and its components* through a well defined management interface enables management policies to be applied *even to the contents of containers like the interaction container discussed earlier*, making it possible to harmonize policies and deliver information in context across business and infrastructure layers.

This is in stark contrast to elastic computing strategies that are virtualization-based, in which the contents of virtual images are not directly manageable by the elastic computing infrastructure. Amazon, with its CloudWatch management dashboard and APIs, provide the ability to manage EC2-based systems using standard systems management metrics. Systems management functionality layered on top of EC2 management could correlate business resources to system resources through systems metrics, though the correlation is made outside of CloudWatch. Recent partnerships between IBM and Amazon allow for containers filled with IBM infrastructure to be managed using Tivoli or similar systems management functionality. However, even in this case, it is important to note that management of *what is in the container* is distinct from *management of Amazon's cloud* unless an integration between the two is ultimately implemented. We have referred earlier in this paper to the mechanism permitting contents of a container to participate in cloud management as *policy extension points*. In practice, policy extension points implement a management interface that makes the resources with which they are associated manageable and makes it possible for these resources to participate in cloud management.

Cloud Composition

The ability of one cloud to participate in managing another will become critical to scaling a cloud. It will provide a means for a private cloud to temporarily use the resources of a public cloud as part of an elastic resource capacity strategy. It also will make it possible to more immediately share functionality, information, and computing resources.

One *real-life* example of a composite cloud is Skype. While Skype may be considered to be just a p2p application, it actually is a global mesh network of managed network elements (servers, routers, switches, encoders/decoders, etc.) that provisions a global VoIP network with voice endpoints that are laptop/desktop computers or handheld devices that run Skype's client application at the edge of the Skype cloud. When the Skype application is not running on a laptop/desktop/handheld device, VoIP calls are not conducted through it. But when the application is running and calls can be conducted, the Skype cloud expands to use the laptop/desktop/handheld device to route traffic and manage network exceptions if needed.

A second *real life* example is FortiusOne's GeoCommons (http://www.FortiusOne.com, http://www.geocommons.com). FortiusOne is developing a next-generation location intelligence platform by blending analysis capabilities of geographic information systems (GIS) with location-based information on the Web. FortiusOne's premise is that it can help organizations make better location-sensitive decisions by simplifying how business information is correlated to visual maps. The technology and data that make up the FortiusOne platform is a combination of open source technology and data that it licenses to complement what it can get from the public domain.

Two applications are made available at GeoCommons: *Finder!* is an application used to find, organize, and share geodata; and *Maker!* is an application used to create maps using GeoCommons and personal data. A simple use case involving both of these tools is the upload of a named data set into *Finder!* that can be linked through postal code, longitude/latitude, or some other location hook to a map, and the subsequent use of *Maker!* to produce a rendering of a map with the named data set superimposed onto it.

FortiusOne has implemented its functionality both as Web applications and services (with a Web service programming interface). It makes this functionality available in its own cloud, which is very similar to Amazon's Elastic Compute Cloud core.

- GeoCommons makes its software-as-a-service platform available through a subscription, with pricing determined by number of reports generated, data set size, and data storage requirements.
- For those who wish to operate in a more secure yet managed *enterprise* context, GeoCommons can be privately hosted for a customer. This version of the platform includes an expanded data set *and* data integration services.

- And for those wishing the ultimate in data privacy who simply do not trust on-line secure environments, FortiusOne packages its GeoCommons functionality and supporting data on a linux appliance and updates data and functionality periodically as required.

The potential for two types of *cloud composition* can be seen in the FortiusOne offerings. First, Amazon's Elastic Computing offering can be used should FortiusOne require additional resources beyond its current capacity. Second, the GeoCommons is accessible via a Web service programming interface, which makes it possible to invoke the services provisioned in the FortiusOne cloud from another cloud. Invoking services of one cloud by another does not require cloud composition, but a need to manage multiple clouds with the same policy set could.

With these examples in mind, we characterize Utility Computing as follows:

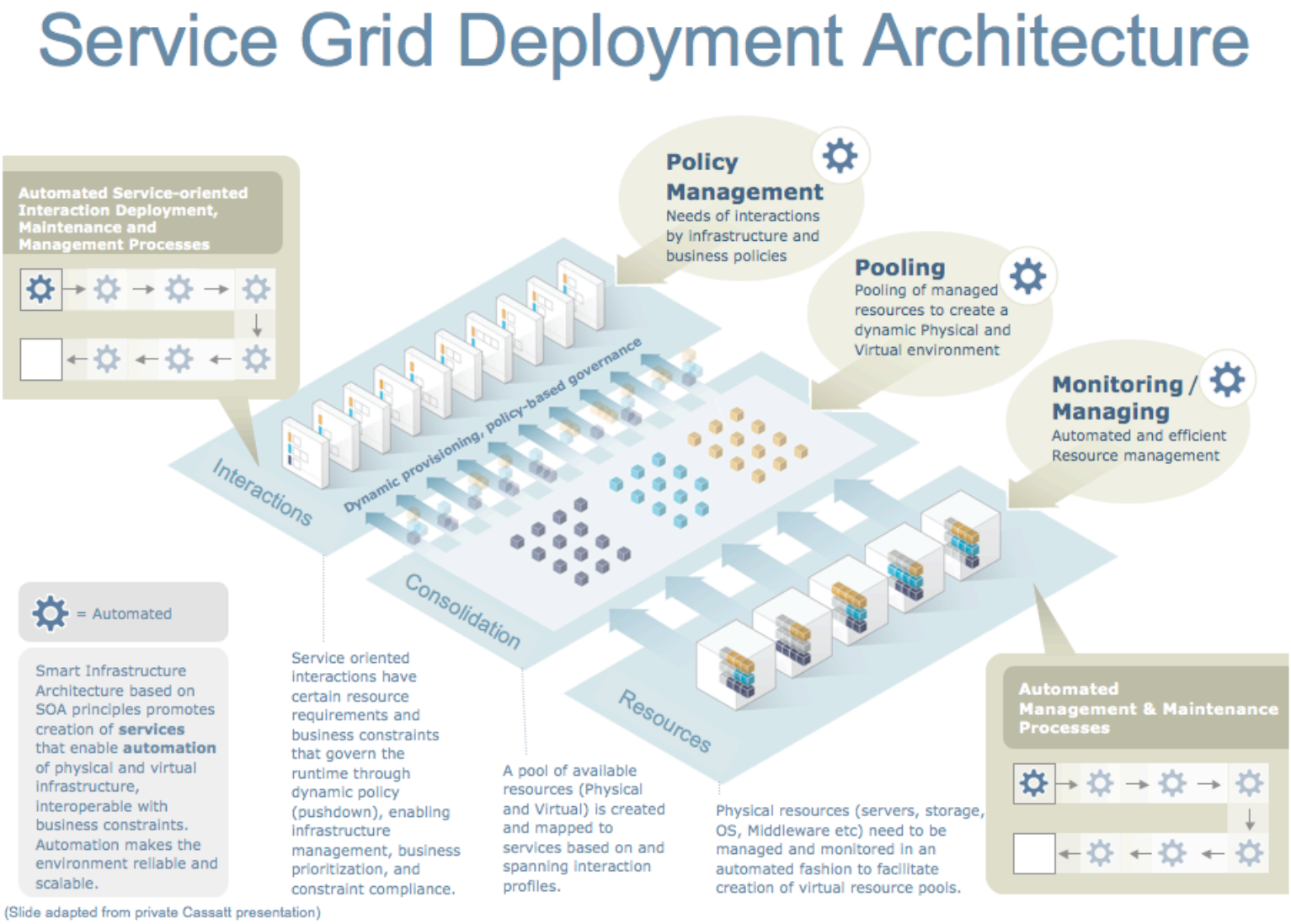
- An OS management layer that transforms hosted resources in a data center into a policy-managed cloud, extensible beyond data center boundaries.
    - It sits over (possibly components physically run on) production hardware.
- It enables clouds conforming to the same cloud management interface to be composited while maintaining cloud identity.
- It knows and manages all resources in a cloud (recursively, as dictated by policy).
- It reallocates (in an autonomic sense) resources in a cloud, as permitted by policy, to accommodate real-time and business-oriented usage patterns.
- It meters use of all resources managed within a cloud.
- It provides security and access control that can federate across cloud boundaries.
- It participates in adjudication of policy collisions across all cloud architecture layers where appropriate.

Utility computing can be considered an overlay on a cloud to make it and its elements manageable and compositional. Preservation of cloud identity also is a nod toward the ability to federate clouds, which has been elaborated in *Service Grid: The Missing Link in Web Services*, together with early thinking of the foundational nature of service grids vis-à-vis business computing ecosystems[v].

SERVICE GRID – THE BENEFIT *AFTER* THE AUTONOMIC ENDPOINT

Before the term *cloud*, the term *service grid* was sometimes used to define a managed distributed computing platform that can be used for business *as well as* scientific applications. Said slightly differently, a service grid is a manageable ecosystem of specific services deployed by service businesses or utility companies. Service grids have been likened to a *power* or *utility grid*: always on, highly reliable, a platform for making managed services available to some user constituency. When the term came into use in the IT domain, the word *service* was implied to mean *Web service*, and *service grid* was viewed as an infrastructure platform on which an ecology of services could be composed, deployed, and managed.

The phrase *service grid* implies *structure*. While grid elements - *servers together with functionality they host within a service grid* - may be *heterogeneous* vis-à-vis their construction and implementation, their presence within a service grid implies *manageability as part of the grid as a whole*. This implies that a capability exists to manage grid elements *using policy* that is *external to* implementations of services in a service grid (at the minimum *in conjunction with* policy that might be embedded in legacy service implementations). And services in a grid become candidates for reuse through service composition; services outside of a grid also are candidates for composition, but the service grid only can manage services within its scope of control. Of course, service grids defined as we have above are autonomic, can be recursively structured, and can collaborate in their management of composite services provisioned across different grids.



A *cloud*, as defined by the cloud taxonomy noted earlier, *is not necessarily a service grid*. There is nothing in cloud definitions that require all services hosted in them to be managed in a predetermined way. There is no policy engine required in a cloud that is responsible to harmonize policy across infrastructure and business layers within or across its boundaries, though increased attention is being given to policy-driven infrastructure management. Clouds are not formed with registries or other infrastructure necessary to support service composition and governance.

A service grid can be formed as an autonomic cloud and will place additional constraints on cloud structure (e.g., external policy management, interaction container-supported composition, a common management interface, support of specific interface standards). These standards will be necessary to manage a service grid both as a technology *and* a business platform.

CONTAINER PERMEABILITY

Clouds and service grids both have *containers*. In clouds, *container* is used to mean *a virtualized image containing technology and application stacks*. The container might hold other kinds of containers (e.g., a J2EE/Java EE application container), but the cloud container is *impermeable*, which means that the cloud does not directly manage container contents, and the cloud contents do not participate in cloud or

container management. In a service grid, *container* is the means by which the grid provides underlying infrastructural services, including security, persistence, business transaction or interaction life cycle management, and policy management. In a service grid, it is possible for contents *in* a container to participate in grid management as a function of infrastructure management policies harmonized with business policies like service level agreements. It also is possible that policy external to container contents can *shape*[4] how the container's functionality executes. So a service grid container's wall is permeable vis-à-vis policy, which is a critical distinction between clouds and service grids[5].

CLOUD VENDORS AND VENDOR LOCK-IN

Vendor lock-in is a concern that will grow as cloud computing becomes more prevalent. Lock-in is best addressed by the implementation of and compliance to standards. In particular, standards for security, interoperability, service composition, cloud and service grid composition, management and governance, and auditing will become especially critical as clouds become embedded into the way that corporations conduct business[6].

Standards for cloud management are emerging as vendors like Amazon, Google, and Microsoft make their offerings available for use. The Web Services community has developed a set of standards for Web service security, Web service management, and Web service policy management, and so forth, that can serve as a basis for standards to be supported in cloud computing. And software vendors[7] are implementing Web service management platforms based on such standards that provide the means to define service level agreements that, when integrated with Web service and supporting infrastructure, govern end-to-end Web service-based interactions, ensure qualities of service, throttle Web service use to ensure performance minimums, etc.

With all this said, however, the fact is that comprehensive standards for cloud computing do not yet exist, since cloud computing is nascent. And until (and probably even after) such standards exist, cloud users should expect to see features and capabilities that justify lock-in – just as one does with other software and utility platforms. Externalizing policy (discussed later in this paper) and implementing services from an outside-in perspective will result in getting benefits from clouds while ameliorating at least some of the aspects of vendor lock-in through loose couplings and manageable interfaces.

VIRTUAL ORGANIZATIONS AND CLOUD COMPUTING

Social networks are examples of platforms that use a somewhat amorphous definition of *organization* similar to a *virtual organization*, which is defined by the National Science Foundation as "a group of individuals whose members and resources may be dispersed geographically and institutionally, but who function as a coherent unit through the use of cyberinfrastructure."[vi] Virtual organizations can form in a variety of ways, usually as a function of roles/responsibilities played in interactions and less as a function

[4] The sense of the word *shape* is consistent with how policy is applied in the telecom world where, for example, bandwidth might be made available to users during particular times in the day as a function of total number of users present.

[5] Cloud management typically is exposed by the cloud vendor through a dashboard. Vendors like Amazon also make functionality underlying the dashboard available as Web services such that cloud users' functionality could programmatically adjust resources based on some internal policy. A service grid is constructed to actively manage itself as a utility of pooled resources and functionality for all grid users. Hence, a service grid will require interaction with functionality throughout the grid and determine with the use of policy extension points whether resource supply should be adjusted.

[6] Note the absence of portability in this list. Interoperability is far more important than portability, which more often leads to senseless technology wars. It is unlikely to be possible to port applications from one cloud to another if these applications make use of cloud APIs. Since clouds are not standard as yet, neither will the APIs be for some time. However, making policy explicit, and providing APIs in the noted areas will go a long way toward enabling interactions to be orchestrated across cloud and service grid boundaries.

[7] See AmberPoint and SOA Software as two examples of Web service management platform vendors.

of title or position in an organization. Roles/responsibilities represent interfaces that have interaction scope and can be used to automate computing and exception handling.

A virtual organization's use of cloud services could vary widely:

- A virtual organization might be a startup company that uses an infrastructure cloud to deploy its computing services because the economic model is right – a pay-for-use model is what it can afford as it gets off the ground, and maybe even throughout its entire corporate existence. This type of organization may be interested in the elastic resources of a cloud, but may not need more advanced capabilities.
- A network of thousands of supply chain partners could be considered to be a virtual organization. It could use a business interaction server hosted in a cloud that manages interactions, ensuring they conform to legal and contract policies and giving all participants in an interaction a record of their participation when that interaction completes. This virtual organization might need the full range of autonomic computing capabilities to manage the complexity of interoperating with many partner systems and accommodating policy differences.
- A network of hundreds of thousands of corporate clients that use travel and entertainment services that comply with corporate standards – all hosted in a cloud – could be considered a virtual organization. One can imagine *transaction consolidations* and *other clearinghouse functions* that are part of this small ecosystem. Interactions might be complex and somewhat long-lived and guided by business policies, though the roles/responsibilities played are likely to be simple.
    - Rearden Commerce (http://www.reardencommerce.com/) implements just such a platform that (as of Jan 2009) serves over 4000 corporate clients and 2 million users (client customers). It brings together corporate business travel policies, reviews of travel/entertainment service providers, expense processing and reporting, etc., in a way that recognizes life of a traveler and makes it easier by eliminating the need to build direct point-to-point traveler to service provider relationships.
- A virtual organization could be composed of scientists who collaborate from their labs across the globe in compute- and data-intensive interactions hosted in a cloud. These organizations typically are not large, but their work requires access to an elastic set of compute resources for hours at a time, and the capability to manipulate huge databases.
- And we could consider a healthcare context as an example of an ecosystem of virtual organizations that scales to be even larger than the user bases of popular social network platforms. Members might include healthcare providers whose credentials must be tracked. Patients must be able to access their health records securely, and authorize access to portions of their charts to others. Healthcare devices and applications or service functionality emit HL7 message streams and related events that result in updating patient charts, informing care providers of procedure results, communicating billing information to hospital billing systems and insurance providers, measuring quality of care, and keeping each member of a care provider group informed of all activities and the corresponding outcomes that occur while they care for a patient who might be physically located in another country.
    - HL7 application messaging protocols are evolving from being ASCII/special character delimited protocols (v2.x) to being XML-based (v3.x). From a technology point of view, HL7's evolution to XML is very complementary to Web service orientation, though it does not force standardization of HL7 messages as yet; we hope that it will bring about standardization as v3.x becomes more widely adopted. Use of XML (and XSLT) also complements a strategy to enrich data passed in messages in a more standard (data extension point) fashion, making it possible for participants in multiparty interactions to pass information that they care about (but maybe no other participant does) along with

standard information useful to all participants in the interaction. Further, because XML structure can be made very explicit, enforcement of business policies is more easily enabled.

Cloud computing must (and in some cases already does) address technical challenges to accommodate these organizational forms, including the following:

- The number of machines in a cloud serving hundreds of millions of users can reach tens of thousands of machines physically distributed across multiple data centers, where it also may be necessary for data center capacity to spill over to still other data centers. Failed servers in such a large-scale environment have to be discoverable and cannot impact the capabilities of the Cloud in aggregate; failed cloud components must be adopted as *the norm* and not *the exception*.
- Failed computers have to be replaced (virtually) by others that are waiting in inventory for automatic configuration and deployment.
- Storage models will have to be reconsidered, since it may be expedient to use massively distributed storage schemes *in addition to* the centralized relational and hierarchical models now in use. We are seeing the beginnings of such with Amazon's and Microsoft's offerings (using Hadoop-like storage models), and the Google File System. "Backup and Recovery" takes on new meanings with distributed file systems. Storage fault tolerance likely will be implemented differently in large clouds than in smaller enterprise clouds.
- Security management systems might have to be federated. Access control schemes will have to accommodate global user bases securing service methods throughout the cloud. There also are global constraints to be considered: some countries do not wish data relating to their citizens to be hosted outside of their national boundaries.
- We often think of network traffic attributed to systems management to be small in comparison to the traffic generated by user interactions with hosted business functionality. Management of clouds and their components, especially clouds containing business functionality managed with externalized business and infrastructure policies, may have to be federated as a function of the size of the cloud to manage a more appreciable amount of management-related network traffic.
- *Complete* testing will be difficult to impossible to perform in a very large and dynamic cloud context, so it is likely that new test methods and tools will be required.

The range of cloud-related virtual organization use cases noted above leverage the cloud computing instantiations we see in the market, and makes clear that the demand is imminent for cloud computing to serve as the infrastructure and utility service grid for a user constituency that is much larger and varied than we've seen to date. We see the first signs of such in social networking platforms and the success that they enjoy as measured by number of users. It will be only a matter of time when we see that business interactions will be conducted in business network group contexts where business policy, roles, responsibilities, and functionality converge in a new type of cloud architecture.

CONCLUDING REMARKS

Autonomic computing, though viewed with suspicion or disbelief in the past years, can be sensibly applied to Cloud Computing in a way that will be useful when developing cloud architectures capable of sustaining and supporting ecosystem-scaled platforms. We suspect that this will become the norm as adoption of cloud computing increases and as social network platforms transition to include business capabilities.

Cloud computing as we see it emerging today is somewhat amorphously defined, making it difficult to form a point of view about the capabilities of currently available cloud computing instances to manage next-century platforms. While it is clear that they can manage today's common platforms, we see architectural challenges for the future that we believe will be difficult to address using current cloud architectures and architecture styles. We identify technical challenges - including architecture style, user

and access control management, the need to have externally managed business and infrastructure policies through interaction containers, and the need for Utility Computing capabilities – that must be addressed to meet future architecture requirements.

Aiming at implementation of an *ecosystem* platform will take us beyond the management capabilities of current cloud offerings. Adding architecture components like the interaction container and externalized policy engine will improve cloud capabilities, but until these become fundamental components in cloud architecture, it is unlikely that a cloud will be able to manage the concerns of a service grid. It is interesting to note, however, that the construct of a service grid enables it to manage the concerns of a cloud. A service grid, as an autonomic architecture that is hardened to be both a service-oriented technology platform and a business platform, can be expected to scale both downward to support enterprise architectures *and* upward and outward to support the types of architectures likely to be pervasive in twenty-first-century computing.

Healthcare represents an area where we believe service grid computing and next-generation architectures will prove to be invaluable. Healthcare systems world-wide are difficult to manage and architecturally extend, and they certainly are difficult to integrate. Unifying information across healthcare facility boundaries is not only an informatics problem, but also an architecture problem that, if not addressed, will hinder national healthcare agendas in the United States and elsewhere[8]. We will discuss a service grid-enabled healthcare platform architecture in detail in a subsequent paper.

ABOUT THE AUTHORS

Thomas B (Tom) Winans is the principal consultant of Concentrum Inc., a professional software engineering and technology diligence consultancy. His client base includes Warburg Pincus, LLC and the Deloitte Center for the Edge. Tom may be reached through his Website at http://www.concentrum.com.

John Seely Brown is the independent co-chairman of the Deloitte Center for the Edge, where he and his Deloitte colleagues explore what executives can learn from innovation emerging on various forms of edges, including the edges of institutions, markets, geographies and generations. He is also a Visiting Scholar and Advisor to the Provost at The University of Southern California. His Website is at http://www.johnseelybrown.com.

[8] One of the first efforts of which we are aware to solve access control/role-responsibility problems in healthcare systems as these relate to management of biomedical information in a service grid is being conducted by Dr. Carl Kesselman and Dr. Stephan Erberich at ISI/USC's Medical Information Systems division. Without doubt, ISI's work will be critical to the implementation of service grid-based next-generation healthcare systems.

[i] The Vision of Autonomic Computing, by Jeffrey O Kephart and David M Chess, IBM Thomas J Watson Research Center, 2001

[ii] Autonomic Computing Manifesto, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, International Business Machines Corporation 2001

[iii] Web Services 2.0, by Thomas B Winans and John Seely Brown, Deloitte, 2008

[iv] Identity Management, by Bill Coleman, Working Paper, 2009

[v] Service Grids: The Missing Link in Web Services, by John Hagel III and John Seely Brown, Working Paper Series, 2002

[vi] Beyond Being There: A Blueprint for Advancing the Design, Development, and Evaluation of Virtual Organizations, National Science Foundation, May 2008

MOVING INFORMATION TECHNOLOGY PLATFORMS TO THE CLOUDS

INSIGHTS INTO IT PLATFORM ARCHITECTURE TRANSFORMATION

THOMAS B WINANS AND JOHN SEELY BROWN

23 APRIL 2009

INTRODUCTION

The Long-Term Credit Bank (LTCB) of Japan underwent a very traumatic reorganization beginning in 1998 following Japan's economic collapse in 1989. The bank was beset with difficulties rooted in bad debts. Possible mergers with domestic banks were proposed, but the bank eventually was sold to an international group, which set about putting the bank back together, launching it in June 2000 as Shinsei Bank, Limited (Shinsei).

LTCB's IT infrastructure was mainframe based, as many banks' infrastructure was at the time (and still is). Acquisitions and organic growth resulted in a variety of different systems supporting similar bank card products. Among the many challenges with which the bank had to grapple as it began its new life was IT infrastructure consolidation, which, in part, translated into deciding how to consolidate bank card products and their supporting IT systems *without further disruption to its bank clientele*. The bank *could* have issued a *new* card representing bank card features and benefits of its individual products consolidated into a single one, but this would have violated the constraint to not further disrupt its client base, risking loss of more clients. Or it could have continued to accept the entire bank card products as it had in the past but, at the same time, find a way to transparently consolidate systems and applications supporting these cards into a very reduced set of systems – ideally one system – that would enable the retirement of many others.

In sum, Shinsei took this second path. Conceptually, and using IT terminology, the bank viewed its various card products as business interfaces to Shinsei that it had to continue to support until a card type no longer had any users, after which the product (the card type) could be retired. Further, to effect consolidation, the bank had to implement an IT application platform supporting both its future and its legacy. The bank IT group set about this mission, empowered by the freedom its business interfaces provided, and, over the next three to five years, replaced many (potentially all) of its mainframe legacy systems using applications constructed with modern technologies and hosted on commodity hardware and operating systems.

Shinsei's example is a direct analog to what IT teams in corporations today must do to transform legacy/existing *Inside-Out* application platforms into *Outside-In* service oriented ones that effectively leverage the capabilities that are afforded through use of cloud and service grid technologies. We begin this paper with a very brief explanation of Outside-In versus Inside-Out architecture styles, clouds, and service grids. Then we explore strategies for implementing architecture transformations from Inside-Out to Outside-In and issues likely to be encountered in the process.

GOING-FORWARD ASSUMPTIONS AND DISCLAIMERS

Globalization, economic crises, technology innovations, and many other factors are making it imperative for businesses to evolve away from current core capabilities toward new cores. Further, there appear to

be indicators that these businesses – if they are to participate in twenty-first-century business ecosystems for more than just a few years – will have to make more core transitions during their corporate life than their twentieth-century counterparts, so the capability to leverage technology to efficiently transform is important to corporate survival.

We believe that clouds, service grids, and service oriented architectures having an Outside-In architecture style are technologies that will be fundamental to successfully making such corporate transformations. There are near-term objectives, like the need for cost and resource efficiency or IT application portfolio management that justify use of these technologies to rearchitect and modernize IT platforms and optimize the way corporations currently deploy them. But there are longer-term business imperatives as well, like the need for a company to be agile in combining their capabilities with those of their partners by creating a distributed platform, and it is at these corporations we specifically target this paper.

OUTSIDE-IN AND INSIDE-OUT ARCHITECTURE STYLES

Architecture styles define families of software systems in terms of patterns for characterizing how architecture components interact. They define what types of architecture components can exist in architectures of those styles, and constraints on how they may be combined. They define how components may be combined together for deployment. They define how units of work are managed, e.g., whether they are transactional (n-phase commit). And they define how functionality that components provision may be composited into higher order functionality and how such can be exposed for use by human beings or other systems.

The *Outside-In* architectural style is inherently top-down and emphasizes decomposition to the functional level but not lower, is service-oriented rather than application-oriented; factors out policy as a first-class architecture component that can be used to govern transparent performance of service-related tasks; and emphasizes the ability to adapt performance to user/business needs without having to consider the intricacies of architecture workings[1].

The counter style, what we call *Inside-Out,* is inherently bottom-up and takes much more of an infrastructural point of view as a starting point, building up to a business functional layer. Application platforms constructed using *client server*, *object-oriented*, and *2/3/n-tier* architecture styles are those to which we apply the generalization *Inside-Out* because they form the basis of enterprise application architectures today, and because architectures of these types have limitations that require transformation to scale in a massive way vis-à-vis Outside-In platforms.

Implementation of an Outside-In architecture results in better architecture layering and factoring, and interfaces that become more *business* than *data* oriented. Policy becomes more explicit, and is exposed in a way that makes it easier to change it as necessary. Service orientation guides the implementation, making it more feasible to integrate and interoperate using commodity infrastructure rather than using complex and inflexible application integration middleware.

As a rule, it is simpler to integrate businesses at functional levels than at lower technology layers where implementations might vary widely. Hence we emphasize decomposition to the functional level, which often is dictated by standards within a market, regulatory constraints on that market, or even accounting (AP/AR/GL) practices.

For a much more detailed discussion of Outside-In versus Inside-Out architecture styles, please see the working paper we call "Web Services 2.0"[i].

[1] An Outside-In architecture is a kind of service-oriented architecture (SOA) which is fully elaborated in Thomas Erl's book called "Service-Oriented Architecture: Concepts, Technology, and Design," so we will not discuss SOA in detail in this paper.

CLOUDS AND SERVICE GRIDS

Since a widely accepted industry definition of cloud computing - beyond a relationship to the Internet and Internet technologies – does not exist at present, we see the term used to mean hosting of hardware in an external data center (sometimes called infrastructure as a service), utility computing (which packages computing resources so they can be used as a utility in an always on, metered, and elastically scalable way), platform services (sometimes called middleware as a service), and application hosting (sometimes called software or applications as a service).

The potential of cloud computing is not limited to hosting applications in someone else's data center, though cloud offerings can be used in this way to elastically manage computing resources and circumvent the need to buy new infrastructure, train new people, or pay for resources that might only be used periodically. Special file system, persistence, data indexing/search, payment processing, and other cloud services can provide benefits to those who deploy platforms in clouds, but their use often requires modifications to platform functionality so that it interoperates with these services.

Before the term cloud, the term service grid was sometimes used to define a managed distributed computing platform that can be used for business as well as scientific applications. Said slightly differently, a service grid is a manageable ecosystem of specific services deployed by service businesses or utility companies. Service grids have been likened to a power or utility grid … always on, highly reliable, a platform for making managed services available to some user constituency. When the term came into use in the IT domain, the word service was implied to mean Web service, and service grid was viewed as an infrastructure platform on which an ecology of services could be composed, deployed, and managed.

The phrase service grid implies structure. While grid elements, servers together with functionality they host within a service grid, may be heterogeneous vis-à-vis their construction and implementation, their presence within a service grid implies manageability as part of the grid as a whole. This implies that a capability exists to manage grid elements using policy that is external to implementations of services in a service grid (at the minimum in conjunction with policy that might be embedded in legacy service implementations). And services in a grid become candidates for reuse through service composition; services outside of a grid also are candidates for composition, but the service grid only can manage services within its scope of control. Of course, service grids defined as we have above are autonomic, can be recursively structured, and can collaborate in their management of composite services provisioned across different grids.

Clouds and service grids both have *containers*. In clouds, *container* is used to mean *a virtualized image containing technology and application stacks*. The container might hold other kinds of containers (e.g., a J2EE/Java EE application container), but the cloud container is *impermeable*, which means that the cloud does not directly manage container contents, and the cloud contents do not participate in cloud or container management. In a service grid, *container* is the means by which the grid provides underlying infrastructural services, including security, persistence, business transaction or interaction life cycle management, and policy management. In a service grid, it is possible for contents *in* a container to participate in grid management as a function of infrastructure management policies harmonized with business policies like service level agreements. It also is possible that policy external to container contents can *shape*[2] how the container's functionality executes. So a service grid container's wall is permeable vis-à-vis policy, which is a critical distinction between clouds and service grids[3].

[2] The sense of the word *shape* is consistent with how policy is applied in the telecom world where, for example, bandwidth might be made available to users during particular times in the day as a function of total number of users present.

[3] Cloud management typically is exposed by the cloud vendor through a dashboard. Vendors like Amazon also make functionality underlying the dashboard available as Web services such that cloud users' functionality could programmatically adjust resources based on some internal policy. A service grid is constructed to actively manage itself as a utility of pooled resources and functionality for all grid users. Hence, a service grid will require interaction

A *cloud*, as defined by the cloud taxonomy noted earlier, *is not necessarily a service grid*. There is nothing in cloud definitions that require all services hosted in them to be manageable in a consistent and predetermined way[4]. There is no policy engine required in a cloud that is responsible to harmonize policy across infrastructure and business layers within or across its boundaries, though increased attention is being given software vendors to policy-driven infrastructure management. Clouds are not formed with registries or other infrastructure necessary to support service composition and governance.

However, a service grid can be formed by implementing a cloud architecture, adding constraints on cloud structure, and adding constraints on business and infrastructure architecture layers so that the result can be managed as both a technology *and* a business platform.

For a much more detailed discussion of architectures in clouds and service grids, please see the working paper we call "Demystifying Clouds: Exploring Cloud and Service Grid Architectures"[ii].

ARCHITECTURE TRANSFORMATION

How to construct an Outside-In architecture that meets next century computing requirements is a topic that requires debate. Should we leverage our past investments in infrastructure, bespoke software development, and third party software products? If so, how can we self-fund this and how long will it take? Or do we go back to the IT funding well with rationale that defends our need now to develop a new service platform and jettison that multimillion-dollar investment we just barely finished paying off?

The answer is *it depends*. We've seen both approaches taken. And we've seen that development of a new platform is no longer as drastic as it sounds.

TRANSFORMING AN EXISTING ARCHITECTURE

It is enticing to think that one could implement an Outside-In architecture simply by wrapping an existing Inside-Out application platform with Web service technologies to service-enable it.

Not quite.

It *is* possible to do that *and then* evolve the Inside-Out architecture to an Outside-In one as budget and other resources allow using a strategy very similar to Shinsei's *business interface strategy* discussed in the introduction of this paper. But the fact that an Inside-Out architecture typically is *not* service-oriented – even though it might be possible to access application functionality *using* Web services – suggests that just using the wrapper strategy will not yield the benefits of a full Outside-In architecture implementation, and compensation for Inside-Out architecture limits may even be more costly than taking an alternative approach.

To illustrate the process of converting an Inside-Out architecture to an Outside-In one, we consider how a typical Web application platform could be converted to an Outside-In architecture in which some Web application accesses all critical business functionality through a Web services layer, and Web services are hosted in a cloud, a service grid, or internally.

From a layered perspective, a Web application usually can be described by a graphic of a three-tiered architecture like the one below.

---

with functionality throughout the grid and determine with the use of policy extension points whether resource supply should be adjusted.

[4] This should not suggest that clouds and elements in them are not managed, because they are. Service grids, however, impose an autonomic, active, and policy-based management strategy on all of the elements within their scope of control so that heterogeneous application and technology infrastructure can be managed through a common interface that can be applied to fine-grained grid elements as desired or necessary.

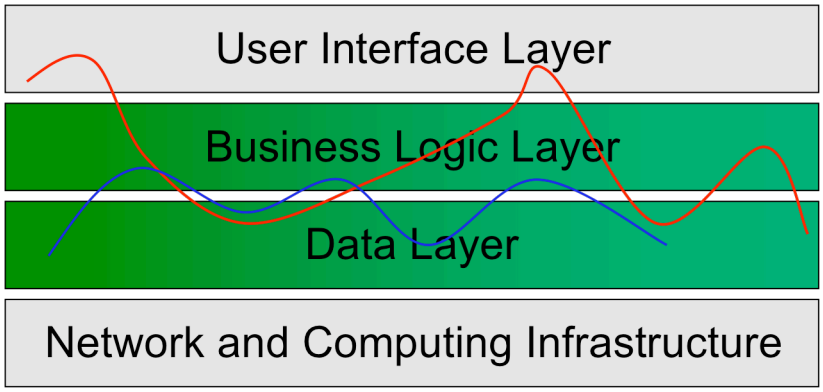


Figure 1

At the top of the graphic we see a user interface layer, which usually is implemented using some Web server (like Microsoft's IIS or Apache's HTTP Web server) and scripting languages or servlet-like technologies that they support. The second layer, the business logic layer, is where all business logic programmed in Java, C#, Visual Basic, and php/python/perl/tcl (or pick your favorite programming language that can be used to code libraries of business functionality) is put. The data layer is where code that manipulates basic data structures goes, and this usually is constructed using object and/or relational database technologies. All of these layers are deployed on a server configured with an operating system and network infrastructure enabling an application user to access Web application functionality from a browser or rich internet client application.

The blue and red lines illustrate that business and data logic sometimes are commingled with code in other layers of the architecture, making it difficult to modify and manage the application over time (code that is spread out and copied all over the architecture is hard to maintain). Ideally, the red and blue lines would not exist at all in this diagram, so it is here where we start in the process of converting this Inside-Out architecture to an Outside-In one.

Addressing Architecture Layering and Partitioning

The first step of transitioning from one architecture style to another is to correct mistakes relating to layering wherever possible. This requires code to be cleaned and commented, refactored, and consolidated so that it is packaged for reuse and orderly deployment, and so that cross-layer violations (e.g., database specifics and business logic are removed from the UI layer, or business logic is removed from the data layer) are eliminated.

Assuming layering violations are addressed, it makes sense then to introduce a service application programming interface (API) between the User Interface Layer and the Business Logic Layer as shown in the slightly modified layer diagram below:

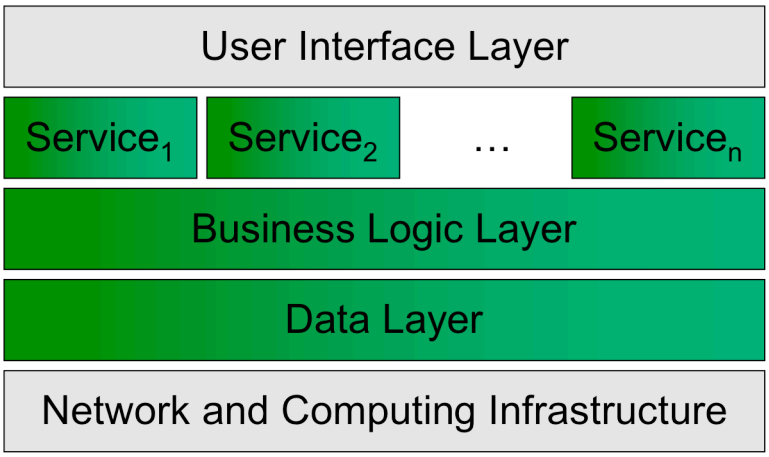


Figure 2

The service layer illustrated here is positioned between the User Interface and lower architecture layers as the *only* means of accessing lower level functionality. This means that the concerns of one architecture layer do not become or complicate the concerns at other levels.

But while we may have cleaned up layering architecture violations, we may not have cleaned up *partitioning* violations. *Partitioning* refers to the "componentizing" or "modularizing" of business functionality such that a component in one business functional domain (e.g., order management) accesses functionality in another such domain (e.g., inventory management) through a single interface (ideally using the appropriate service API). Ensuring that common interfaces are used to access business functionality in other modules eliminates the use of private knowledge (e.g., private APIs) to access business functionality in another domain space. Partitioning also may be referred to as *factoring*. When transitioning to a new architecture style, the first stage of partitioning often is implemented at the Business Logic Layer, resulting in a modified architecture depicted as follows:

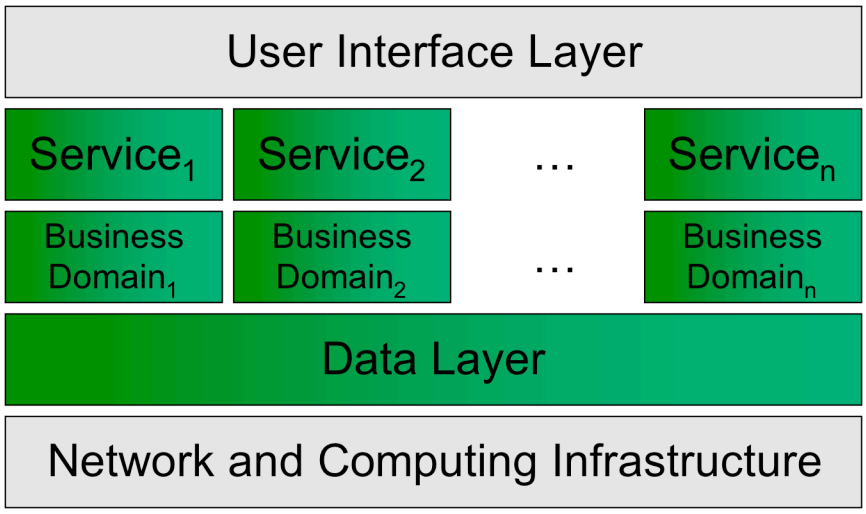


Figure 3

The next phase of transformation focuses attention on partitioning functionality in the database so that, for example, side effects of inserting data into the database in an area supporting one business domain does not also publish into or otherwise impact the database supporting other business domains.

Why go to such trouble?

Because it is possible to transition the architecture in Figure 1 to become like one of the depictions below. Figure 4 illustrates a well-organized platform that might be centrally hosted.

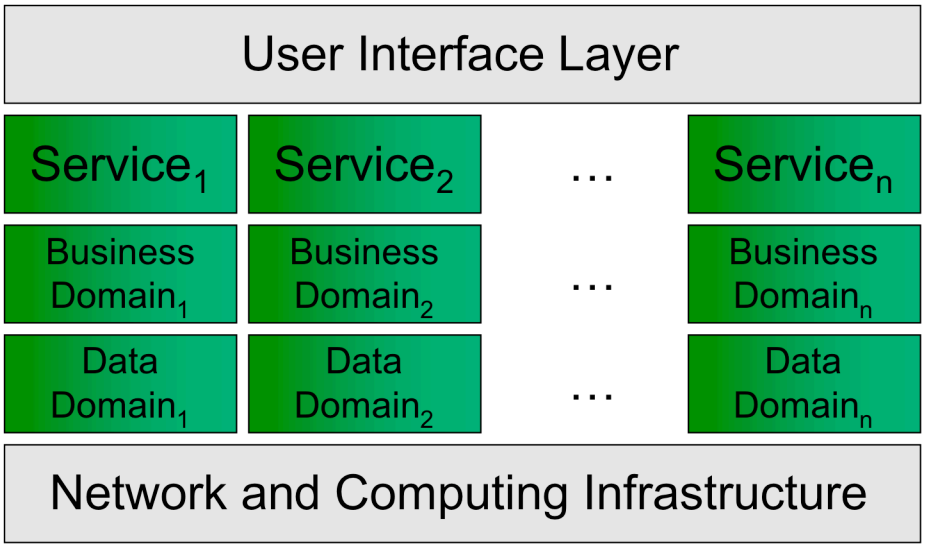


Figure 4

Figure 5 illustrates a well organized platform that could be hosted in a service grid or even many service grids.

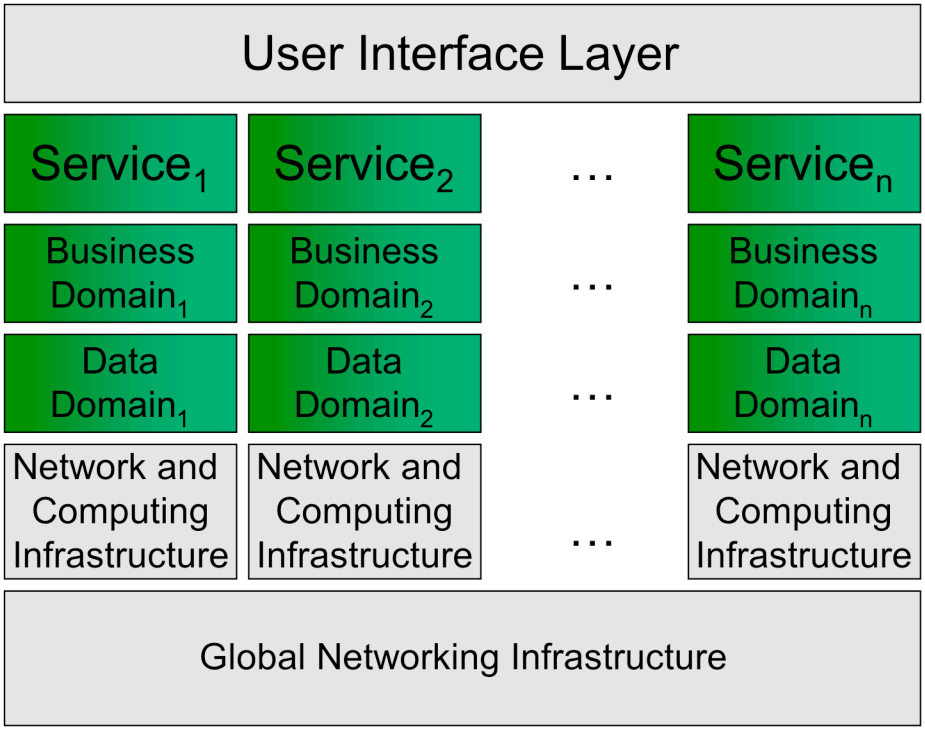


Figure 5

Figures 4 and 5 make it simple to see that services and their supporting business logic and data functionality could be replaced easily with an alternative service implementation without negatively impacting other areas of the architecture, *provided that* functionality in one service domain is accessed by another service domain *only through the service interface*. And such capability is required in order to simplify management of an application portfolio implemented on such an architecture as well as distribute and federate service implementations.

Externalizing Policy

The next step toward implementing an Outside-In architecture is to external both business and infrastructure policies from any of the functionality provisioning services illustrated in the figures above.

Our use of the word *policy* connotes constraints placed upon the business functionality of a system, harmonized with constraints on the infrastructure (hardware and software) that provisions that functionality. These constraints could include accounting rules that businesses follow, role-based access control on business functionality, corporate policy about the maximum allowable hotel room rate that a nonexecutive employee could purchase when using an online reservation service, rules about peak business traffic that determine when a new virtualized image of an application system should be deployed, and the various infrastructural policies that might give customer *A* preference over customer *B* should critical resource contention require such.

Policy extension points provide the means by which policy constraints are exposed to business and corresponding infrastructural[5] functionality and incorporated into their execution. They are not

[5] Rob Gingell and the Cassatt team are incorporating policy into their next-generation utility computing platform, called Skynet. In their parlance, policy primitives represent metrics used by policy extension points in support of management as a function of application demand, application service levels, and other policy-based priority inputs, such as total cost. The policy-based approach to management is being implemented so that infrastructure policy can

configuration points that are usually known in advance of when an application execution starts and that stay constant until the application restarts. Rather, policy extension points are dynamic and late bound to business and infrastructural functionality, and they provide the potential to *dynamically shape* execution of it within the deployment environment's runtime.

Externalizing policy highlights a significant distinction between Inside-Out and Outside-In architecture styles. Inside-out architectures usually involve legacy applications in which policy is embedded and thus externalizing it is – at best – very difficult. Where application policies differ in typical corporate environments, it becomes the responsibility of integration middleware to implement policy adjudication logic that may work well to harmonize policies over small numbers of integrated systems, but this will not generalize to manage policy in larger numbers of applications as would be the case in larger value chains. To illustrate the problem of scaling systems where policy is distributed throughout it, consider the system illustrated in Figure 6.

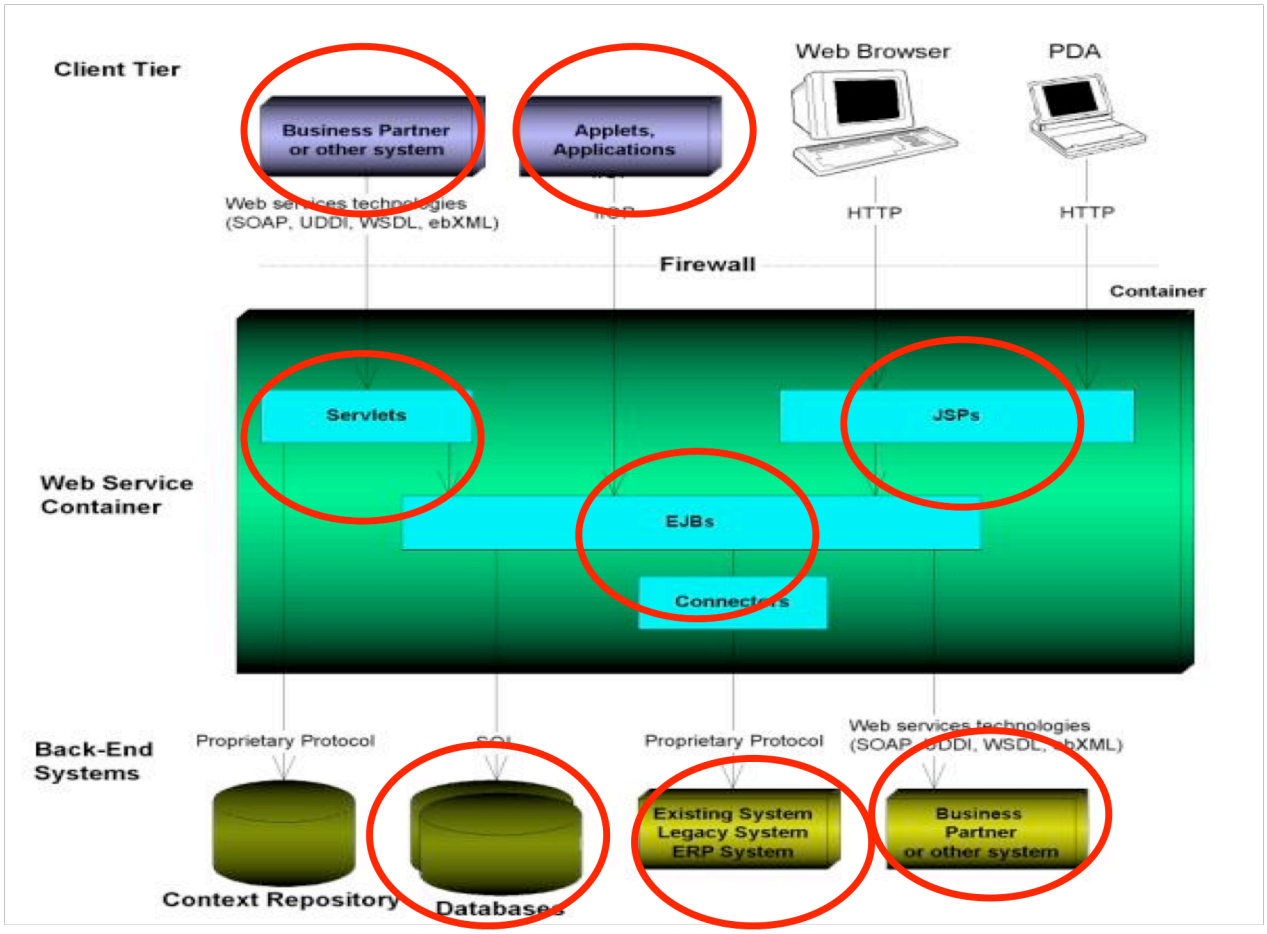


Figure 6

Figure 6 illustrates a system where business policy exists in multiple locations of the architecture as indicated by areas outlined in red. Scaling this architecture would be disastrous because policy would be distributed as copies (or, worst case, as different code bases) over a very complex deployment

be connected to business service level agreements. This will be fundamental to automating resource allocation, service prioritization, etc., when certain business functionality is invoked or when usage trends determine need. Such capability will prove invaluable as the number of elements within a cloud or service grid becomes large.

environment. But a well-factored environment like the ones illustrated in Figures 4 and 5 have business logic located in a single logical architecture layer and, from it, policy can be externalized with the development of adapters or similar architecture components that play the role of policy extension points described above. Once this is accomplished, the architecture we started with now begins to resemble the architecture illustrated in Figure 7 below, in which policy has been externalized, possibly federated, and put under the control of policy management services. Once policy from business functionality is externalized, it can be harmonized with infrastructure policy as feasible/desired.

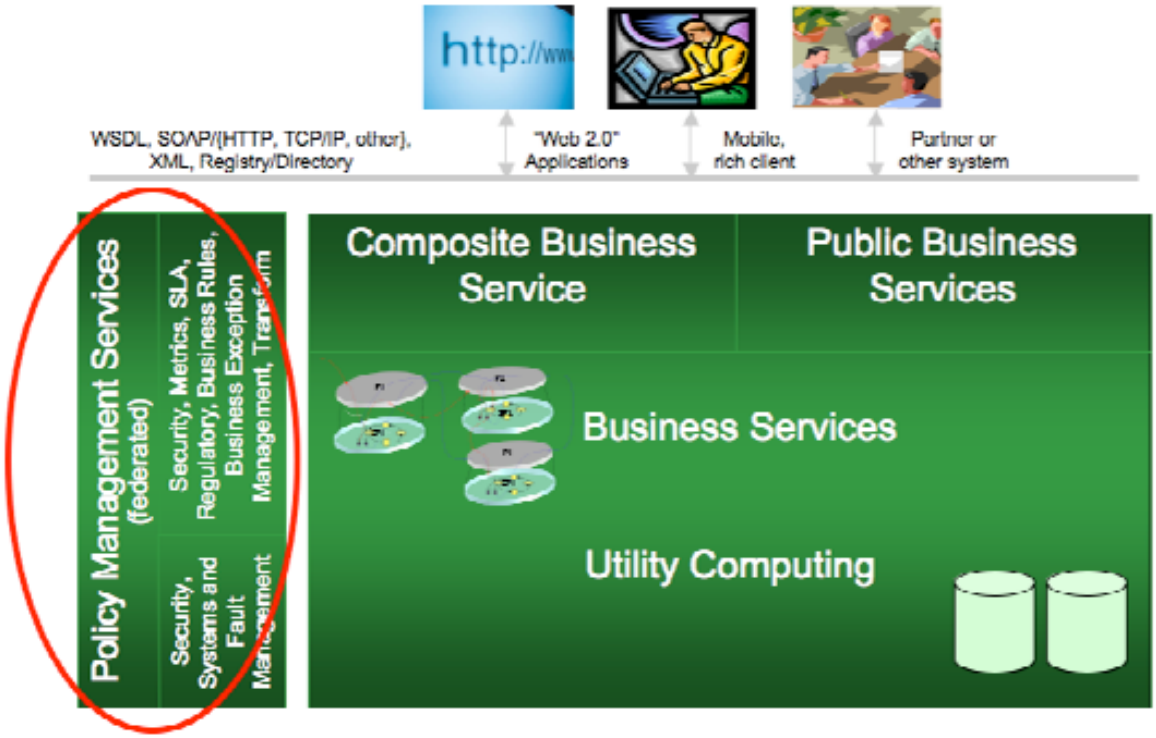


Figure 7

Replacing Application Functionality with (Composite) Services

The final step in transforming an Inside-Out platform to an Outside-In platform is to replace *business application code that coordinates invocation of multiple services* with *composite service* if this is possible.

In Figure 7 we use the term *composite service* to mean business services formed by combining other business services (or methods thereof) together to form coarse (larger) business functions that are peer with application functionality. For example, we might see services to manage order fulfillment, invoice submission and payment processing, orchestrations with which billing staff use to prepare for invoicing, logistics planning, and so forth. As a kind of mental mapping between Figures 1 and 7, the composite service functionality in Figure 7 maps to business logic that has *leaked* into Web pages of the Web application in Figure 1 (shown with red and blue lines) that are used to manage order fulfillment, invoice submission, etc.

Orchestration is often equated to workflows used to coordinate some ordering of service method invocations. Workflow and other business process management technologies are now well-known within today's corporations. Workflow engines for Web services have been commoditized through open source initiatives and by commercial software vendors. These engines make it possible to implement composite

Web services as either state machine or sequential workflows. Use of state machine flows makes it possible to avoid prescriptively dictating how systems interoperate. They also provide the opportunity to incorporate human intelligence tasks to help resolve exception conditions that often emerge from composite services or straight through processing flows[6].

STARTING FROM SCRATCH – MAYBE EASIER TO DO, BUT SOMETIMES HARD TO SELL

Many CIOs and IT executives hope that the costs and risks of transforming a legacy platform architecture to an Outside-In one can be amortized over time, and who can blame them. Most have probably spent a considerable sum developing the current architecture, so the last thing any IT executive wants to ask for is new budget sufficient to fund still more infrastructure-level activities or require their companies to choose between new functionality or resolved infrastructure issues.

But we have experienced many changes in the technology world during the last 20 years that strongly suggest there is value in at least considering whether implementing Outside-In architectures from scratch would be worthwhile. An interesting catch here is that this argument could have been made and *was* made at each new stage of development over the last 20 years. *Why is the story now so different?* Because today's context versus just a few years ago is qualitatively different. Significant broadband capacity, economic storage (both self- and cloud-hosted), cheap memory and modern caching services, commodity 64-bit operating systems, XML accelerators and sophisticated application protocol management capabilities, commoditized integration/interoperability technologies, virtualization and utility computing, cloud and service grid computing, and other relatively recent innovations challenge the traditional wisdom that it is better to evolve and extend an existing platform than it is to create a new one that could circumvent problems from retrofitting an existing architecture in ways quite counter to its original design.

Coupled with these advances are elaborations of industry domains in the form of *industry* or *business solution maps*. These maps are used by consulting companies and software vendors to provide business process oriented views of industry, define roles played and responsibilities performed within business processes, begin (at least) to build out functional decompositions of the industry domain, and map processes to technology solutions where feasible. Using these maps as starting points streamlines process and data mapping efforts that used to take months to even several years to perform (in larger companies), and results in a detailed functional view that is necessary to build a well-formed Outside-In architecture.

Building from scratch is really not the same as starting with nothing but a blank sheet of paper. While it is unusual to find a company able to take a purely greenfield approach (unless it is a startup), there are ways for established businesses to get comfortable with taking a greenfield approach to developing an Outside-In architecture, and subsequently developing a strategy to implement it even if using components of existing platforms.

CONCLUDING REMARKS

Transforming an Inside-Out architecture to an Outside-In architecture can be a lengthy process – it is a function of existing system complexity, size, and age. One company who shared with us its experiences when making such a transition was Rearden Commerce (Rearden). Prior to three and a half years ago, Rearden's architecture was composed like many of the Web applications we see today: three-tiered, open source Web and application server technologies, and a relational database. Rearden's Web application exposed a framework to which merchant clients could interface to Rearden "services" or functions. Rearden's management team had the foresight to recognize the company's need to create a platform (not just an application), and the corresponding need to make architecture changes to support more rapid

[6] Ultimately, it may prove necessary to incorporate a constraint engine into the way that services are composited to harmonize policies and dynamically govern execution of the composite.

development and simpler deployment of new services. By this time, Rearden already had clients, so it understood that change had to be made transparently to its user base whenever possible or in a way that the user base viewed as a positive upgrade of capability to which they could migrate as doing so became expedient to their business.

Rearden strengthened its leadership team with technologists who had participated in Web service infrastructure companies and could guide in Rearden's architecture modernization. This new leadership team undertook a transformation of the company's three-tiered architecture to a service-oriented one over a two-year period using a process like the one described above. At the end of the two-and-a-half-year period, Rearden had transformed its traditional Web application architecture to a service oriented one with externalized policy management.

When performing an architecture transformation, is it necessary that *all* architecture components are *entirely* transformed – as was the case with Rearden? If there was queue-based middleware in the old architecture, should it be replaced? Should all *old* applications be replaced with custom applications having appropriate policy extension points?

The answer to these questions is *it depends*. Certainly it is possible to replace enterprise application integration technologies with commodity or open source technologies, simplify them, or maybe – in some cases - even eliminate them. It is unlikely that middleware supporting reliable messaging and long-lived business transactions between business partners needs to be totally replaced in or removed from an Outside-In architecture. But its use can be couched in ways that eliminate tight coupling between partners, and commingling of business policy with integration functionality that makes partner integration difficult to change as policies change or as a partner networks expand.

Taking an Outside-In point of view requires that we separate concerns from the start. Application platforms should be viewed as distributed from their beginning rather than be made so after the fact by attaching some distribution layer to them. We must understand how we have permitted business security and access control models to be built into our architectures and how, now that technology innovations enable us to challenge these limits, we must remove them from our computing platforms to realize business agility goals that will be demanded of an architecture in the twenty-first-century. Technologies we've used in the past can be useful to us in the future. Success in implementing an Outside-In architecture is less a function of technology than it is of a business and technology architecture vision that forces business and technology architects to view business capabilities from a global, outside in and top down perspective.

ABOUT THE AUTHORS

Thomas B (Tom) Winans is the principal consultant of Concentrum Inc., a professional software engineering and technology diligence consultancy. His client base includes Warburg Pincus, LLC and the Deloitte Center for the Edge. Tom may be reached through his Website at http://www.concentrum.com.

John Seely Brown is the independent co-chairman of the Deloitte Center for the Edge, where he and his Deloitte colleagues explore what executives can learn from innovation emerging on various forms of edges, including the edges of institutions, markets, geographies, and generations. He is also a Visiting Scholar and Advisor to the Provost at The University of Southern California. His Website is at http://www.johnseelybrown.com.

[i] Web Services 2.0, by Thomas B Winans and John Seely Brown, Deloitte, 2008

[ii] Demystifying Clouds: Exploring Cloud and Service Grid Architectures, by Thomas B Winans and John Seely Brown, Deloitte, 2009

MOTIVATION TO LEVERAGE CLOUD AND SERVICE GRID TECHNOLOGIES

PAIN POINTS THAT CLOUDS AND SERVICE GRIDS ADDRESS

THOMAS B WINANS AND JOHN SEELY BROWN

26 MAY 2009

INTRODUCTION

It was September 2008 when Larry Ellison was asked about whether Oracle would pursue a cloud vision. His response at the time was that Oracle would eventually offer cloud computing products. But in the same conversation, Mr. Ellison also noted - in his inimitable fashion - that cloud computing was such a commonly used term as to be "*idiocy*."

Fair comment, actually, given that cloud computing *is* such an overloaded term. It can be used as a synonym for *outsourced data center hosting*. It can be used to define what Salesforce.com and NetSuite do – they offer *software as a service*. Some have referred to the Internet as *the cloud …* an uber-cloud containing all others. Cloud computing sometimes is imprecisely used to reference *grid computing* for database resource management or massively parallel scientific computing. And cloud computing has been taken to mean *time sharing*, which is both a style of business and a technology strategy that sought to share expensive computing resources across corporate boundaries at attractive price points. It appears *on the surface* that the IT industry *has* redefined much of what it does now *and has done for quite a while* as cloud computing, and that Oracle indeed might need only to change verbiage in a few of its ads to align them with a well-formed cloud vision.

But today's IT leaders are operating in a business climate in which intense commoditization and change force deployment of *new* IT-enabled business processes and require acknowledgement that business processes and architectures that are fixed/rigid in their definition will not scale to large networks of practice, that IT budgets may have reached the point where conventional internal cost cutting can wring out only nominal additional value unless business and IT processes change, and that *doing business internationally* is *not* the same as *conducting global business* (i.e., outsourcing is not equivalent to organizing and conducting global business). So, while Mr. Ellison's remarks may express the sentiments of many IT leaders today who have spent a considerable amount on infrastructure, they *cannot* be correct *unless* the processes and techniques developed in IT during the past 20 years will be the foundation for processes and techniques of the next 20 years.

We do not believe that twentieth-century IT thinking can or should be the de facto foundation for twenty-first-century IT practices. We believe that now, more than ever before, IT matters, and it has already become the critical center of business operations today. As such, IT leaders have no choice but to continue to chase cost and margin optimization. They *also* have no choice but to carefully set and navigate a course to renovate and/or replace *twentieth-century* practices with *for twenty-first-century* practices and technologies so that product lines and services that companies offer today can remain relevant through significant market transitions.

This paper is the first in a set of three that attempt to establish a thought framework around cloud computing, its architectural implications, and migration from current computing architectures and hosting capabilities to cloud computing architectures and hosting services. We begin by exploring three IT pain

points that can be addressed by cloud and service grid computing. Subsequent papers more completely elaborate these pain points and methods to handle them.

IT PAIN POINTS

There probably is a very large number of *IT pain points* that IT leaders in today's corporations would want to see addressed by cloud and service grid computing. We highlight three in this paper:

- Data Center Management
- Architecture Transformation and Evolution (evolving current architectures or beginning from scratch)
- Policy-based Management of IT Platforms

PAIN POINT: DATA CENTER MANAGEMENT

Summary: The challenges of managing a data center, including network management, hardware acquisition and currency, energy efficiency, and scalability with business demands, all are costly to implement in a way that easily expands and contracts as a function of demand. Also, as businesses aggregate and collaborate in global contexts, data center scalability is constrained by cost, the ability to efficiently manage environments and satisfy regulatory requirements, and geographic limitations. Adding to this complexity, the need to manage change demanded by today's changing global and corporate climates underscores the need for transforming the ways that we manage data centers: methods of the past 5 to 10 years do not scale and do not provide the requiste agility that is now needed.

Cloud solutions: Cloud solutions can form the basis of a next generation data center strategy, bringing agility, elasticity, and economic viability to the fore:

- Affordable elasticity, scalability
  - Resource optimization through virtualization
  - Management dashboards simplify responses evoked by seasonal peak utilization demands or business expansion
  - Finer-grained container management capabilities (e.g., Cassatt's) will serve to fine tune elasticity policies
  - Capability to affordably deploy many current technology-based applications as they exist today, possibly to rearchitect them over time
  - Minimized capital outlay, which is especially important for startups, where initial funding is way too limited to use to capitalize infrastructure
  - Extreme elasticity – handling spikes of traffic stemming from something catching on or "going viral" (e.g., having to scale from 50 to 5000 servers in one day because of the power of social media)
- Affordable and alternative provisioning of disaster recovery
  - Cloud data storage schemes provide a different way to persistently store certain types of information that make explicit data replication unnecessary (storage is distributed/federated transparently)[1]
  - Creation of virtual images of an entire technology stack streamlines recovery in the event of server failure

[1] It is interesting to consider the implications that new cloud persistence schemes can have on registries like public DNS and Web service registries. Were these registries to be hosted in a cloud, it might be possible to rethink their implementation so as to simplify underlying database replication and streamline propagation of registry updates.

- Utility computing management platforms enable consistent management across data center boundaries. Evolution of utility computing to enable cloud composition will simplify implementation of failover strategies

❖ Affordable state-of-the-art technology

- The exponential nature of digital hardware advances makes the challenge of keeping hardware current particularly vexing. Buying cloud services transfers the need to keep hardware current to the cloud vendor – except, possibly, as this applies to mainframe or other legacy technologies that remain viable
  - It is reasonable to expect cloud vendors to offer specialized hardware capabilities (e.g., DSP/GPU/SMP capabilities) over time in support of gaming/graphics, parallel/multithreaded applications, etc.
  - Specialized hardware needs (e.g., mainframe-based) probably will not be the responsibility of the cloud vendor, but there is no reason why a private cloud/service grid should not be able to be composed with a public cloud/service grid

❖ Operational agility and efficiency – cloud vendors will oversee management of hardware and network assets at a minimum. Where they provide software assets or provision a service grid ecosystem, they likely will provide software stack management as well

- Management of assets deployed into a cloud is standardized. Management dashboards simplify the management and deployment of both hardware and software

❖ Energy efficiency becomes the cloud vendor's challenge. The scale of a cloud may well precipitate the move to alternative cooling strategies (e.g., water vs. fan at hardware (board/hardware module) levels), air and water cooling of data centers, increased management software capabilities to interoperate with data center policies to control power up/down of resources and consolidate (on fewer boxes) processes running in a cloud given the visibility to utilization, service level agreements, etc. One could even imagine implementing a power strategy that continuously moves resource intensive applications to run where power is less expensive (e.g., power might be less expensive at night than the day so keep running this application on the dark side of the earth)

❖ Big enterprise capabilities for small company prices

- Startups can use and stay with cloud solutions as they grow and become more established

❖ Hardware and data center cost-savings and staff cost optimizations enable businesses to self-fund innovative IT initiatives

- Those who wish to leverage the cloud's functional capabilities will have to build their own capabilities (e.g., services and/or applications) to interoperate with resources in the cloud provided that they wish to do more than simply use a cloud as outsourced hosting

❖ Security compliance (e.g., security of information and data center practices, PCI data security compliance) will increasingly become the responsibility of cloud vendors

- This will be true especially as clouds become/evolve into service grids, as cloud vendors geographically distribute their capabilities, and as specialized clouds are provisioned to serve specific industries

PAIN POINT: ARCHITECTURE TRANSFORMATION/EVOLUTION (THE BROWNFIELD VS. GREENFIELD CONUNDRUM)

Summary: Significant investment in application platforms in the last 10 years have resulted in heterogeneous best-of-breed application systems that have proved hard and costly to integrate *within corporate boundaries*. Scaling them *beyond* corporate boundaries into corporate networks of practice takes integration to a level of complexity that appears insurmountable, but the perceived costs of starting fresh seem equally so. IT leadership knows it must do something about the application portfolio it has assembled to make it interoperable with partner networks without requiring massive technology restructure in unrealistically short time periods. It also knows the business must quickly respond to global market opportunities when they present themselves. How does IT Leadership guide the architectural evolution and transformation of what exists today to enable rapid-fire response without starting from scratch or trying to change its application platform in unrealistic time periods?

Cloud solutions: Cloud solutions can form the basis of an application portfolio management strategy that can be used to address tactical short-term needs, e.g., interoperability within a business community of practice using the cloud to provision community resources, and to address the longer-term needs to optimize the application portfolio and possibly rearchitect it.

- Cloud vendors offer the capability to construct customized virtualized images that can contain software for which a corporation has licenses. Hosting current infrastructure in a cloud (where such is possible) provides an isolated area in which a corporation or corporate partners (probably on a smaller scale due to integration complexities associated with older infrastructure and application technologies) could interoperate using existing technologies
    - Why would companies do this?
        - To move quickly with current platforms
        - To economically host applications, minimize private data center modifications, and, in so doing, self-fund portfolio optimization and/or rearchitecture work
        - Use current capabilities, but shadow them with new capabilities as they are developed – ultimately replacing new with old
        - Simplify architecture by removing unnecessary moving parts
- Cloud vendors offer application functionality that could replace existing capabilities (e.g., small-to-large ERP, CRM systems). Incorporating this functionality into an existing application portfolio leads to incremental re-architecture of application integrations using newer technologies and techniques (Brownfield), which, in turn, should result in service-oriented interfaces that can become foundational to future state. An incremental move toward a rearchitected platform hosted using cloud technologies may prove to be the only way to mitigate risks of architectural transformation while keeping corporate business running. Conversely, clouds also represent locations where Greenfield efforts can be hosted. Greenfield efforts are not as risky as they sound given the maturity (now) of hosted platforms like SalesForce, NetSuite, etc.
    - How quickly can transformation of an existing platform be accomplished? This depends upon the architectural complexity of what is to be transformed or replaced. A very complex and tightly coupled architecture might require several years to decouple so that new architecture components could be added – assuming no Greenfield scenario is desired or feasible – whereas it might be possible to move a simply structured Web application in a matter of hours. A platform that has specialized hardware requirements (e.g., there is mainframe dependency, or digital signal processing hardware is required) might have to be privately hosted, or be hosted partly in public and partly in private clouds

- Cloud application programming interfaces (APIs), together with the concepts of distribution, federation, and services that are *baked in*, provide a foundation on which to implement loosely coupled, service-oriented architectures and can logically lead to better architecture
    - Web services, reliable queuing, and distributed storage (nonrelational with somewhat relational interfaces, and relational) provide foundational infrastructure capabilities to implement modern architectures using standardized APIs (e.g., WS-*)
    - Standardized interfaces, loose architecture couplings, and standardized deployment environments and methods increase reuse potential by making it easier to compose new services with existing services
- Clouds provide a means to deal with heterogeneity
    - Initially, heterogeneity is dealt with through management layers
    - Better architecture as noted above further enhances this as heterogeneity is encapsulated beneath standardized and service-oriented APIs
    - Once heterogeneity is contained, a portfolio optimization/modernization strategy can be put into place and implemented

PAIN POINT: POLICY-BASED MANAGEMENT OF IT PLATFORMS

Summary: Policy constraints are difficult to impossible to implement especially in a rapidly changing world/context. Business processes and policies are embedded in monolithic applications that comprise corporate business platforms today. Even the bespoke applications constructed in the past 10 years share this characteristic since policy and process were not treated as formal architecture components. Consequently, application scalability vis-à-vis provisioning policy-driven business capabilities is limited. Organizational model changes, e.g., mergers and acquisitions (or divestitures) or corporate globalization into very loosely coupled business networks, underscore the need for policy to be made explicit. The ability to conduct business in a quickly changing world will be a direct function of the capability to manage using policy.

Service grid solutions: In one sense, this pain point can be considered to be related to the Brownfield/Greenfield Conundrum that, if addressed, results in a distributed/federated, service-oriented, loosely coupled architecture in which policy *can* be factored out and considered a first-order architecture component. However, it also is clear that: (1) *everyone* does not need policy factored like this; and (2) where policy must be exposed may vary by domain, community, geography, etc. Hence we deal with this pain point separately and consider the need to address it a prerequisite condition to leveraging the full capabilities of a service grid.

Corporations require enterprise qualities in architectures, and they will have the same expectations of clouds where they will deploy critical platforms. Scaling architectures for use in increasingly larger communities as well as simply making platforms that are far more configurable and compositional requires the ability to expose policy extension points that can be incorporated into a management scheme that can implement and enforce policy across the entire technology stack. The result of such architecturally pervasive policy management is that control over environmental as well as business constraints is provisioned. When corporate architectures are deployed into service grids, these policy extension points must be used even in grids composed of other grids.

- Cloud vendors are implementing *utility computing* management capabilities, which are, themselves, policy driven. As these capabilities are further refined, it will become feasible to integrate business policies with infrastructure management policies on which business and infrastructure service level agreements/processes can be based
- Outside-In architectures (in our view, service-oriented architecture properly done) provision interfaces that easily align with business processes and minimize architecture complexity,

resulting in a simpler architecture in which policy is externalized. Externalized policy provides the opportunity for business policy to join with infrastructure policy

- Externalized policy provides the foundation on which policy-driven business processes can be constructed and managed. This results in increased business agility because it simplifies how businesses interoperate: they interoperate at the business process level, and not at the technology level; policies can be changed with significantly less impact on the code that provisions business functionality
- Cloud vendors use containers to deploy functionality. As these containers become permeable such that their contents can both be managed and expose policy extension points, then policies can span the entire cloud and grid technology stacks
- Service grids provision architecture components, e.g., policy engines and interaction services, that enable policy to be managed/harmonized explicitly and separately from other business functionality – across architecture layers, across business networks of practice - and used as the foundation of business interactions
    - It is important to note that policy is viewed as a constraint continuum covering infrastructure management to domain (regulatory, industry/market sector) policy constraints

CONCLUDING REMARKS: TO THE 21st CENTURY AND BEYOND

We see, in this paper, that cloud computing can be used to address tactical problems with which IT continually deals, like resource availability and reliability, data center costs, and operational process standardization. These near-term objectives represent sufficient justification for companies to use cloud computing technologies even when they have no need to improve their platforms or practices. But there are longer-term business imperatives as well, like the need for a company to be agile in combining their capabilities with those of their partners by creating a distributed platform that will drive aggressively toward cloud and service grid computing. We believe that clouds, service grids, and service-oriented architectures are technologies that will be fundamental to twenty-first-century corporations' successfully navigating the changes that they now face.

The pain points discussed above illustrate a progression of change that most corporations have already begun, whether they are just starting up or are well established. We began with use of cloud hosting services as an alternative to self-hosting, or as an alternative to other current day third-party hosting arrangements that do not offer at least the potential of cloud computing. For those companies that need to pursue implementation and management of a service-oriented architecture, we discussed pain points relating to rearchitecting current platforms to leverage cloud computing, and the possible need to formalize the way that policy is used to manage IT platforms within and across service grid boundaries.

Many of the concepts mentioned in the pain point discussions are architectural, and are not defined at all in this paper. However, they are more completely elaborated in our other papers, called *Demystifying Clouds: Exploring Cloud and Service Grid Architectures*, and *Moving Information Technology Platforms To The Clouds: Insights Into IT Platform Architecture Transformation*.

ABOUT THE AUTHORS

Thomas B (Tom) Winans is the principal consultant of Concentrum Inc., a professional software engineering and technology diligence consultancy. His client base includes Warburg Pincus, LLC and the Deloitte Center for the Edge. Tom may be reached through his Website at http://www.concentrum.com.

John Seely Brown is the independent co-chairman of the Deloitte Center for the Edge, where he and his Deloitte colleagues explore what executives can learn from innovation emerging on various edges, including the edges of institutions, markets, geographies, and generations. He is also a Visiting Scholar and Advisor to the Provost at USC. His Website is at http://www.johnseelybrown.com.