DEMYSTIFYING CLOUDS

EXPLORING CLOUD AND SERVICE GRID ARCHITECTURES

THOMAS B WINANS AND JOHN SEELY BROWN

23 APRIL 2009

INTRODUCTION

Cloud Computing is in vogue. But what is it? Is it *just* the same thing as *outsourcing the hosting of web application*? Why might it be useful and to whom? How does it change the future of enterprise architectures? How might clouds form the backbone of 21st 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 21st 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 the size of the management team or other costs, 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; and
- ❖ 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.

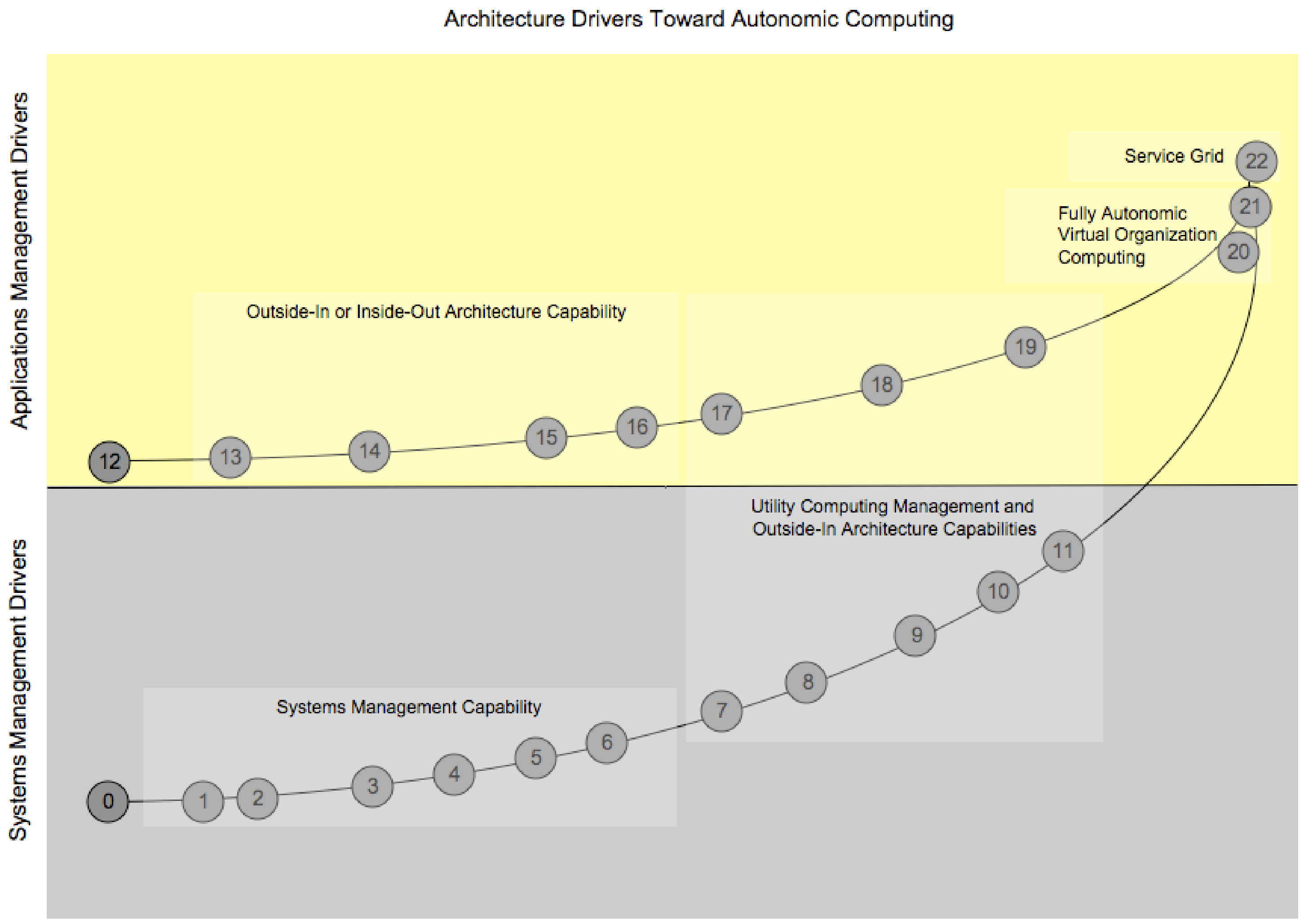
- ❖ The way that users and access control are managed in typical applications today is no longer flexible enough to express roles and responsibilities that people will play in next generation business interactions. Roles will be played by people *outside of* or *across corporate boundaries* in an online 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.
  - o A cloud should be able to dynamically provision itself and optimize its own construction and resource consumption over time.
  - o A cloud must be able to recover from routine and extraordinary events that might cause some or all of its parts to malfunction.
  - o 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 multi-tenant 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.


Architecture Drivers Toward Autonomic Computing
Applications Management Drivers
Systems Management Drivers
Service Grid
Fully Autonomic Virtual Organization Computing
Outside-In or Inside-Out Architecture Capability
Utility Computing Management and Outside-In Architecture Capabilities
Systems Management Capability
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

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

0. Architecture state: No systems management in place
1. Systems and their all system resources must be identifiable
2. System and all system resources must be manageable
3. Policy-driven secured access to the system and all managed system resources must be provided
4. The system must reallocate managed resources on failures as a function of policy
5. The system must reallocate managed resources on various system-level conditions as a function of policy

6. The system must be managed lights out in a single data center context
7. System management capability must scale across clouds of the same type
8. System management capability must scale across clouds of different types. These clouds must be managed uniformly while maintaining separate cloud identities
9. The 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 must be harmonized across cloud boundaries
11. It must be possible to integrate cloud systems management policy with external constraint management systems
12. Monolithic applications and traditional application integrations exist/are sufficient
13. The application platform must be service oriented
14. Applications must be replaced with business services
15. Security and access control must exist for business services
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. The system must reallocate managed business service on various business-level conditions as a function of policy to accommodate real-time and business-oriented usage patterns
19. An interaction container must be used as *application container* in a multi-tenant environment
20. Business service and systems management policies must be 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 multi-tenant 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 exisiting 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.

    - o 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.

    - o 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 which 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., are they transactional (n-phase commit) or not. 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, it factors out policy as a first class architecture component that can be used to govern transparent performance of service-related tasks, and it 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 operabilty 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.,

[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"[2], so we will not discuss SOA in detail in this paper.

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.

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 on-line 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 21st 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 multi-tiered 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 multi-tenant 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 multi-party 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:

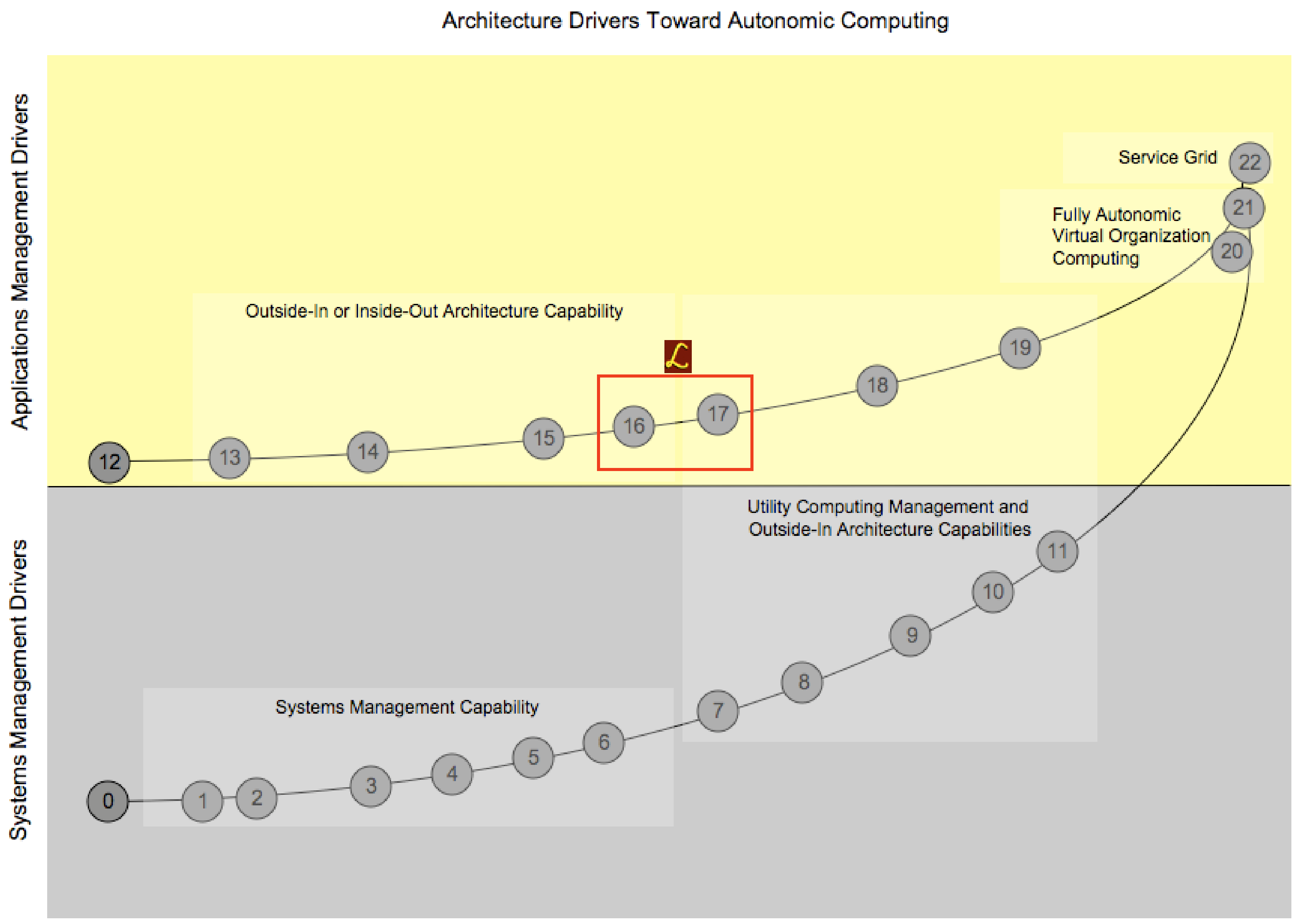
- Policy extension points must be exposed and formally declared in any part of the architecture that must be managed.
- Policy management must support *policy pushdown* to enable extensible and dynamic detection of policy violation and policy enforcement.
- It must be possible to version policy so that 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 occurences *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 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.


Architecture Drivers Toward Autonomic Computing
Applications Management Drivers
Systems Management Drivers
Service Grid
Fully Autonomic Virtual Organization Computing
Outside-In or Inside-Out Architecture Capability
Utility Computing Management and Outside-In Architecture Capabilities
Systems Management Capability
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

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 non-executive employee could purchase when using an on-line 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 which 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, Layer7 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. Recent partnerships between IBM and Amazon allow for containers filled with IBM infrastructure to be managed using Tivoli or similar systems management functionality. However, 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.

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 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 Computing Cloud core.

- ❖ GeoCommons makes its software-as-a-service platform available through a subscription, with pricing determined by number of reports generated, dataset 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 GeoCommon 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:

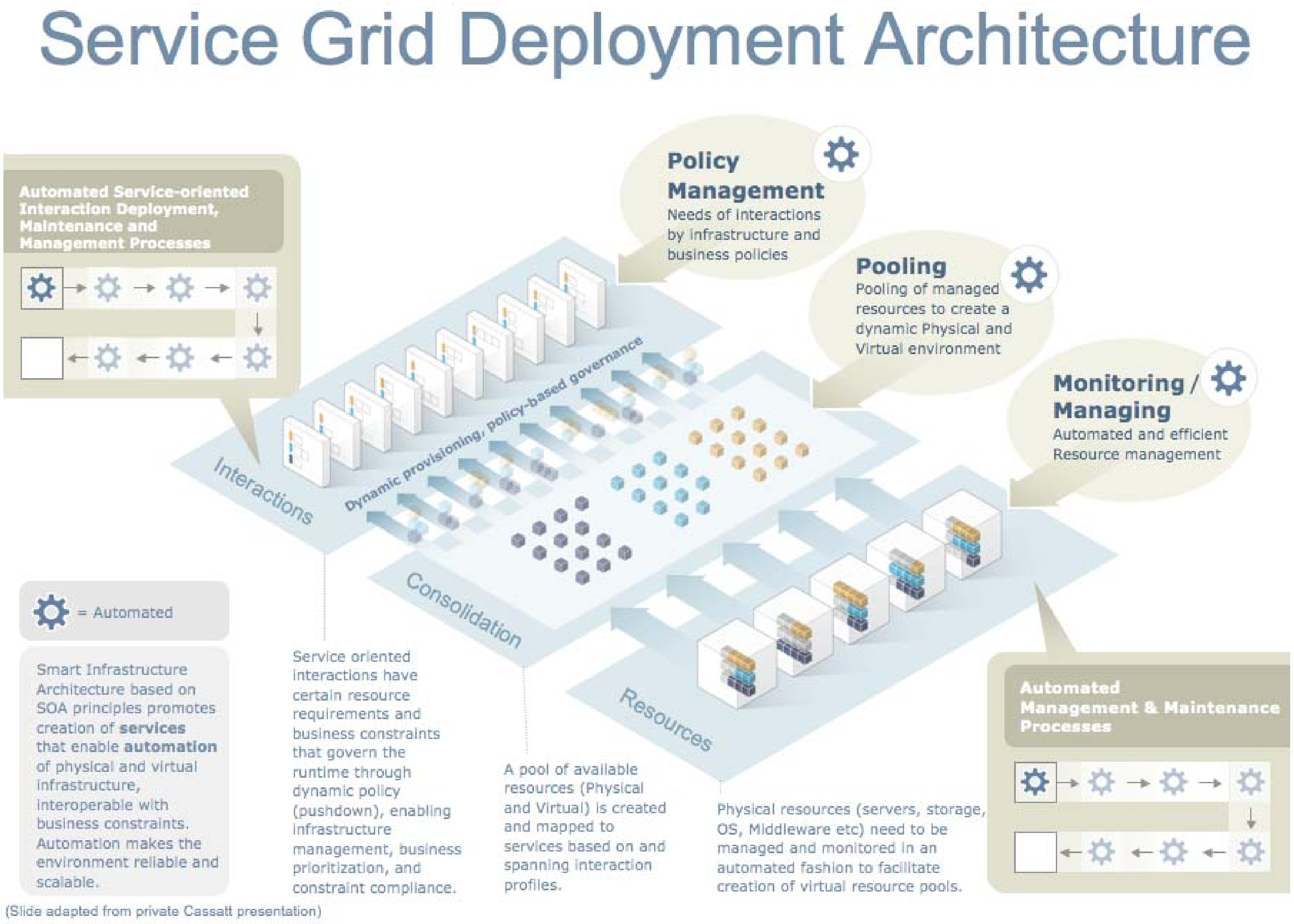
- ❖ An OS management layer that transforms hosted resources in a datacenter into a policy-managed cloud, extensible beyond datacenter boundaries.
    - o It sits over (possibly components physically run on) production hardware.
- ❖ It enables clouds conforming to the same cloud management interface to 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.


Service Grid Deployment Architecture
Automated Service-oriented Interaction Deployment, Maintenance and Management Processes
Policy Management
Needs of interactions by infrastructure and business policies
Pooling
Pooling of managed resources to create a dynamic Physical and Virtual environment
Monitoring / Managing
Automated and efficient Resource management
Interactions
Dynamic provisioning, policy-based governance
Consolidation
Resources
= Automated
Smart Infrastructure Architecture based on SOA principles promotes creation of services that enable automation of physical and virtual infrastructure, interoperable with business constraints. Automation makes the environment reliable and scalable.
Service oriented interactions have certain resource requirements and business constraints that govern the runtime through dynamic policy (pushdown), enabling infrastructure management, business prioritization, and constraint compliance.
A pool of available resources (Physical and Virtual) is created and mapped to services based on and spanning interaction profiles.
Physical resources (servers, storage, OS, Middleware etc) need to be managed and monitored in an automated fashion to facilitate creation of virtual resource pools.
Automated Management & Maintenance Processes
(Slide adapted from private Cassatt presentation)

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/JavaEE 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

[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 a 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 or not 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.

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, yet 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 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 activies 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 – hopefully 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 multi-party 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 Computing Cloud instantiations we see in the market, and makes clear that the demand is imminent for Cloud Computing to serve as the infrastructure and utilty 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.

CONCLUSIONS

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 21st 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.

[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.

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 USC. His web site is at http://www.johnseelybrown.com.

[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