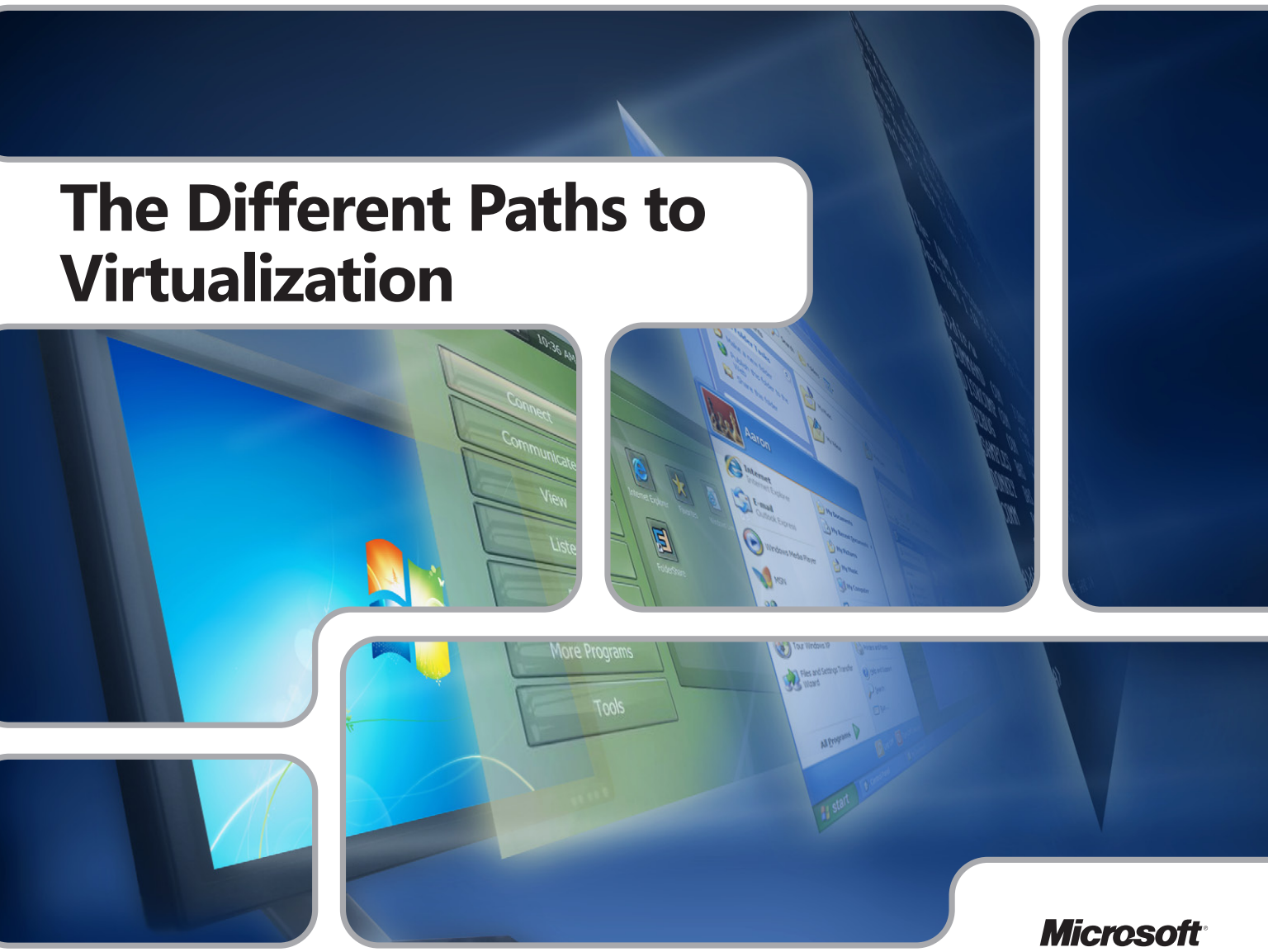
The Architecture Journal

Learn the discipline, pursue the art, and contribute ideas at www.architecturejournal.net

24

input for better outcomes

The Different Paths to Virtualization



Microsoft

Contents

Dear Architect,

Welcome to issue 24 of *The Architecture Journal*, themed on virtualization strategies. This topic has been growing in importance, as its initial success in the operating-system field—with the advent of new companies such as VMWare that were born around the concept—encouraged further exploration into several other contexts such as desktop, application, platform, hardware, and even data, just to mention a few.

The initial benefit of reusing an existing solution by simulating its original environment is too little compared with the myriad benefits that are aggregated, as long as virtualization was spread to different contexts. We might find these benefits from hardware consolidation (to reduce energy consumption and, therefore, the carbon footprint) to desktop-application coexistence with a straightforward, never-before-seen rollout process.

As typically occurs, however, benefits never come without expense, as new implementation challenges emerge. Here, again, is where architects come into play—to ensure that technology does not become an end unto itself, but a business-flow enabler.

For this issue, we counted on Richard Carpenter as guest editor-in-chief. Without his help, we could not have delivered these articles to you today:

- Nishant Thorat writes on how virtualization is already having an impact on IT architecture and its current trends.
- Phil Riccio explains factors, such as risks and mitigations, that one should consider in a virtualization plan.
- Alan Madisson covers the operational challenges of virtualization and how the Microsoft Operations Framework (MOF) overcomes these challenges.
- Clemens Reijnen takes us a bit away from the theme of this issue, as he explains what architects can expect from Visual Studio 2010 for application life-cycle management (ALM).
- Greg Hutch shares lessons that he learned from his experiences in virtualization projects.
- David Ziembicki invites us to look at virtualization from the strategic perspective of dynamic IT—an advanced state of IT maturity that has a high degree of automation, integrated-service management, and efficient use of resources.

That is all for this issue of the *Journal*, dear reader. We hope that you enjoy these articles and, as usual, send us your comments to archjrnl@microsoft.com.

Diego Dagum
Editor-in-Chief

The Impact of Virtualization on Software Architecture

by Nishant Thorat and Arvind Raghavendran

Summary

This article examines the impact of virtualization on software architecture and attempts to identify enabling architecture patterns.

Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem.

–DAVID WHEELER

Introduction

Virtualization has a profound impact on IT environments, as it abstracts the physical characteristics of computing resources from users. Virtualization techniques are mainly used for enhancing capacity (using excess computer resources efficiently), compatibility (running legacy applications on modern platforms), and manageability (easy patch management and deployment). However, one must not confuse virtualization technologies—such as OS virtualization and application virtualization—with virtualization itself. In principle, virtualization can be applied to provide an abstraction layer that helps solve complex problems in software architecture. As Table 1 shows, virtualization technologies are of different types and can be complementary in different architectural styles.

In this article, we examine the impact of virtualization on software architecture to support desired attributes—such as high dynamic scalability, high availability, and geographic distribution—in modern composite applications, as well as the benefits of extending a virtualized environment to traditional applications.

Traditionally, enterprise software was designed and developed to address specific enterprise problems. Hence, the architecture of the enterprise-application software stack would be confined to the enterprise boundary and often restricted to the *n*-tier architectural pattern. With the ever-increasing volume of unstructured data and unprecedented increase in transactional data volume, scalability is becoming an important factor for enterprise applications. The classic *n*-tier applications do not scale linearly, following Amdahl's law.[1]

The scalability problem can be overcome by applying the virtualization principles to the complete application stack. In this approach, each layer of the application stack is mapped to multiple physical resources through virtualization. This leads to logical separation at every level, data, components, and actual devices—hence, enabling scaling out without adding complexity.

Virtualization Attributes for Architects

Virtualization demands changes in the way in which we approach various attributes of software architecture, such as security, deployment, multitenancy, and communication.

Security

From an architectural perspective, there is some confusion about how virtualization differs from existing models of computing and how these differences affect the technological approaches to network- and information-security practices.

An application-delivery infrastructure that is based on virtualization is a new approach that is made possible by advances in data-center virtualization, the availability of high-speed bandwidth, and innovations in endpoint sophistication:

Table 1: Virtualization technologies

Virtualization type	Resource	Enabling layer
Software-as-a-Service (SaaS)	Applications accessed as Web services	Multitenant architectures
Utility computing (on-demand, cloud)	Distributed data-center software and hardware	Distributed resource and workload managers
Computational grids	Computers across locations or organizations	Grid middleware
Transaction grids (fabrics)	Hardware and software within an organization	Fabric middleware
Data grids	Data sources across locations or organizations	Data-source resource broker
Storage grids or utilities	Storage hardware	Storage broker
Application virtualization	Execution environment	Run-time support
Virtual server	OS and CPU	Virtual-server support
Virtual-machine monitor (VMM)	CPU	Hypervisor (possibly, with CPU support)
Virtual appliance	CPU	Application virtualization Packaging and runtime

- **Should data be kept in the data center?** Centralizing the data into a data center facilitates easier maintenance of applications and better protection of confidential data. Also, operating costs are reduced, as less time is spent on managing employee endpoint resources.
- **Should application streaming be used?** Application streaming provides business value to remote offices and retains control of confidential data. Virtualization of desktop applications reduces the amount of time that an application is exposed to potential vulnerabilities, as it transparently streams compliant images from the data center and reduces the risk of malicious code that might be lingering on corporate endpoints.
- **Should a wide variety of endpoint devices be supported?** A greater scale of application virtualization can be achieved by providing the user with a wide variety of endpoint devices, such as browser, remote display, or streamed-application access.

Deployment

Application virtualization has a huge impact on how most enterprise solutions are packaged and deployed:

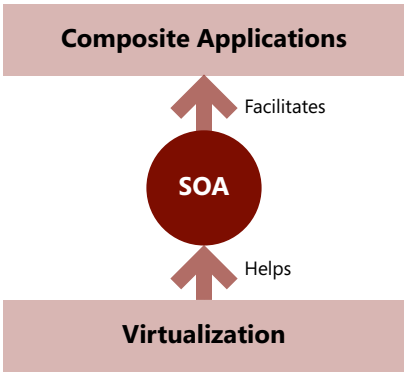
- **Handling compatibility issues**—Software appliances can provide the answer, as they have several benefits over traditional software applications. A software appliance dramatically simplifies software deployment by encapsulating an application's dependencies in a pre-integrated, self-contained unit. It frees users from having to worry about resolving potentially complex OS-compatibility issues, library dependencies, and undesirable interactions with other applications.
- **Impact on current systems**—Do the virtualized applications integrate seamlessly with the existing products that are used for testing, deployment, and troubleshooting? If not, contingency plans for this new way of packaging applications must be incorporated. Ensure that the choice of virtualization technology allows for easy integration with the existing enterprise-application integration (EAI) strategies.
- **Additional hardware requirements**—When we deploy multiple versions side by side on a machine, we must consider issues such as port conflicts, network capacity, and other hardware-related questions.

Multitenancy

Multitenancy—in which a single instance of the software, running on a service provider's premises, serves multiple organizations—is an important element of virtualization and implies a need for policy-driven enforcement, segmentation, isolation, governance, service levels, and chargeback/billing models for different consumer constituencies:

- **Isolation and security**—Because tenants share the same instance of the software and hardware, it is possible for one tenant to affect the availability and performance of the software for other tenants. Ensure that application resources such as virtual portals, database tables, workflows, and Web services are shared between tenants, so that only users who belong to a tenant can access the instances that belong to that tenant.
- **Database customizability**—Because the software is shared between tenants, it becomes difficult to customize the database for each tenant. Virtualization of the data layer and effective use of data partitioning—coupled with selection of the best approach among separate databases, a shared database with separate schemas, and a shared database with shared schema—will enable you to achieve your database-customization goals.
- **Usage-based metering**—There are several methods for associating usage-metering data with a corresponding tenant. Authentication-based mapping, general HTTP- or SOAP-request parameter-based mapping, and application separation are some of the methods that can be used.

Figure 1: Virtualization–SOA–composite-application relationship



Communication

Communication plays a key role in the new world of virtualization. With more and more applications acting as services, and composition coming into play, it is very important that there be a common integration platform to help these disparate services communicate with each other.

In traditional architectures, synchronous communication patterns are used. Synchronous communication requires both endpoints to be up and running. However, because of load-balancing or operational reasons such as patching, virtual endpoints could become unavailable at any time. Thus, to ensure the high-availability asynchronous communication, patterns should be used.

In recent years, the enterprise service bus (ESB) has permeated the IT vernacular of many organizations that are looking for a new "magic bullet" to deal with the ongoing challenge of connecting systems. In computing, an ESB consists of a software-architecture construct that provides fundamental services for complex architectures via an event-driven and standards-based messaging engine (the bus):

- **Brokered communication**—ESBs send data between processes on the same or different computers. As with message-oriented middleware, ESBs use a software intermediary between the sender and the receiver to provide a brokered communication between them.
- **Address indirection and intelligent routing**—ESBs typically resolve service addresses at runtime by maintaining some type of repository. They usually are capable of routing messages, based on a predefined set of criteria.
- **Basic Web-services support**—ESBs support basic Web-services standards that include the Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL), as well as foundational standards such as TCP/IP and XML.

Virtualization, SOA, and Composite Applications

Composite applications are built from *servicelets* or *microservices*, which by nature are distributed, federated, and replicated. *Servicelets*

Table 2: Application life-cycle patterns

Life-cycle stage	Pattern	Intent/Usage
Model	Service Wrapper	The Service Wrapper pattern facilitates the abstraction of native applications as a service. The service interface can be exposed as WSDL.
Assemble	Service Discovery	The Service Discovery pattern facilitates the automatic discovery of services, based on the metadata that accompanies the service in a service catalog.
	Service Negotiation	The Service Negotiation pattern facilitates the abstraction of services through *service-contract templates* that are built around different numbers of *contract parameters*. The negotiation could be manual, semi-automated, or fully automated. The autonomic quotient is governed by the number of static numerical parameters.[3]
	Service Composition	The Service Composition pattern facilitates dynamic composition of preexisting and composite services. It results in a composite-service description that can enable identification of a suitable execution model.
Execute	Composite Service Execution	The Composite Service Execution pattern facilitates the execution of a composite application, provided that the execution model is identified. There are two possible patterns: • **Central Authority Execution**—In this pattern, the composite service holds the scheduler. • **P2P Execution Pattern**—In this pattern, the responsibility of coordinating the execution of a composite service is distributed among services. A coordinator that is installed on each service is responsible for execution of initialization.

are self-contained hardware and software building blocks. They could possibly communicate by embracing service-oriented architecture (SOA). It would be wrong to assume that composite applications are by definition SOA, although SOA is a major technology for implementing them. However, it would be interesting to infer that virtualization is the platform that could enable SOA-based composite applications. The inherent nature of virtualization provides a necessary indirection that is needed both for SOA and to build composite applications. (See Figure 1 on page 3.)

Composite-Application Life Cycle

In a typical scenario, composite applications follow the **Model-Assemble-Deploy-Manage**[2] SOA life-cycle pattern. However, for the purposes of our discussion, we propose a modified version of the pattern: **Model [with optional Wrapper for legacy applications]-Assemble [through Service Discovery/Publish/Negotiation]-Deploy-Execute-Manage**. We use this model to identify the patterns for each phase of the composite-application life cycle, as Table 2 shows.

Virtualization Patterns

For the purposes of this discussion, we consider access virtualization, workload virtualization, information virtualization, and platform virtualization (see Figure 2).

Access Virtualization and Related Patterns

Access virtualization is responsible for authentication and makes it possible for any type of device to be used to access any type of application. This gives developers an opportunity to develop applications without being restricted to any specific device.

Identity management and resource access are the main aspects of access virtualization.

Federated Identity Pattern

Identity in a virtualized environment must be implemented based on standards and should be federated across unrelated security domains. In a virtualized environment, the application components can be anywhere; hence, the identity should be context-aware. For example, for load-balancing purposes, your application component might move into a different security domain; to work it seamlessly in a new environment, the original identity should be federated into the new environment.

Figure 2: Virtualization patterns

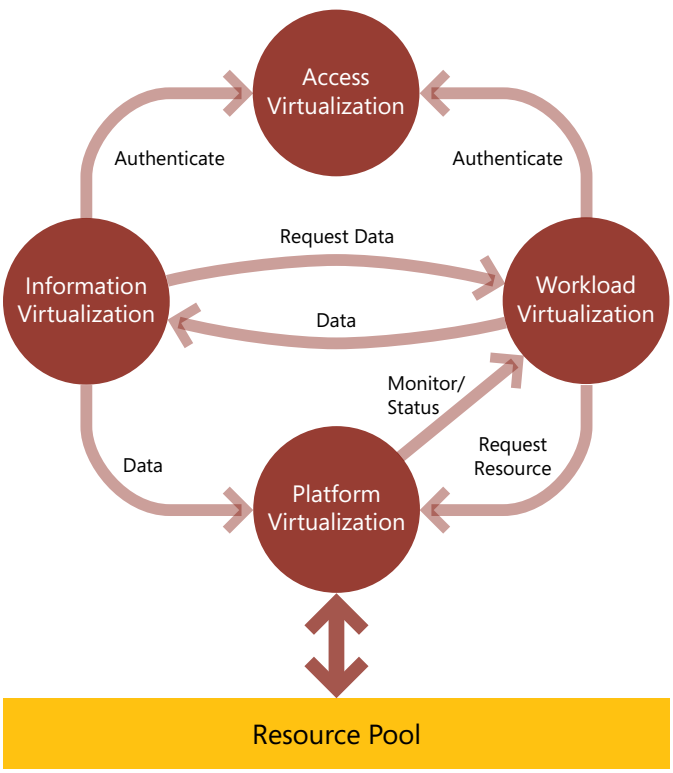
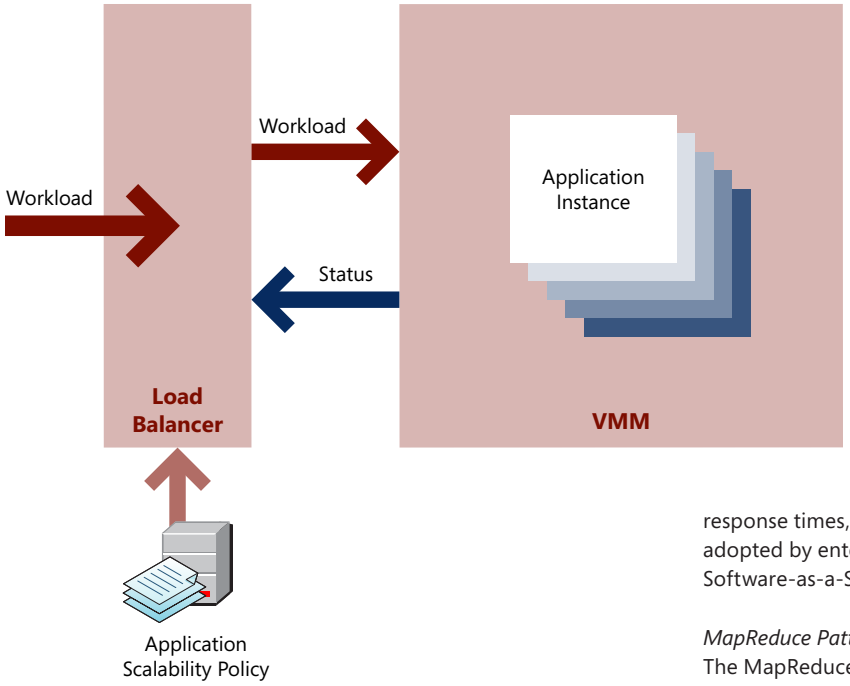


Figure 3: Virtual Scaling pattern



Endpoint Virtualization Pattern

The Endpoint Virtualization pattern is used to hide client-device details—delivering a robust operating environment, irrespective of equipment, connectivity or location. This pattern provides application-streaming capabilities, application isolation, and licensing control.

Application installation and configuration can be captured and streamed back in virtualized layers. The deployment process should complement the capturing process, identify the component dependency, and so on. Although complete isolation is an idealistic view, an application might require sharing OS resources and might also have linking to coexisting applications and resources. For example, it is unreasonable to expect that every .NET application would be virtualized with its own copy of .NET.

Also, while we are designing applications, we should refrain from making any specific assumptions about endpoint capabilities.

The Endpoint Virtualization pattern can also be a key component to various thin-client approaches, as it allows for the distribution of application layers and the capture of user sessions to maintain the personal preferences of the individual user.

Workload Virtualization and Related Patterns

Workload virtualization is responsible for selecting the most appropriate system resources to execute the workload tasks. This selection could be policy-driven and based on different criteria, such as resource availability and priority.

Workload virtualization uses the scheduler to optimize resource usage and introduces multiple patterns for resource management and isolation.

Virtual Scaling Pattern

Application scaling is the ability of an application to adapt to enterprise growth in terms of IO throughput, number of users, and data capacity. The software does not need to change to handle these increases. Increased capacity demands can be met by creating a new virtual machine (VM) and running a new instance of the application. The load-balancing component chooses the appropriate VM that will execute the tasks.

The Virtual Scaling pattern (see Figure 3) allows on-demand applications to be scaled up and down, based on policies. These policies could be set to set up new VMs, to maintain the service quality in peak usage while decommissioning the VMs when demand subsides. The application should be able to communicate to the virtual-machine monitor/Virtual Machine Manager (VMM) to report its status. This pattern simplifies the scalable application design. The VMM hides all of the resource details from the application. The application resource status could be monitored by analyzing the usage statistics of queues, threads, response times, and memory in real time. This strategy is mostly adopted by enterprise applications that are being targeted for the Software-as-a-Service (SaaS) model.

MapReduce Pattern

The MapReduce pattern divides processing into many small blocks of work, distributes them throughout a cluster of computing nodes, and collects the results.

Applications can be encapsulated into VMs, and—because VM images are used as templates to deploy multiple copies, when needed—only the template images must be maintained. The nodes can communicate through virtual networks. Typically, in a virtualized environment that employs the MapReduce pattern, a VM image that has the necessary software is prepared and uploaded to the virtual-workspace factory server. When a new node is requested, the virtual-workspace factory server in each site selects the available physical nodes and starts the VM cloning process. VM clones are booted and configured by using the appropriate network addresses.

Information Virtualization and Related Patterns

Information virtualization ensures that appropriate data is available to processing units (PUs). The data could be cached on the system, if required. Storage virtualization and I/O virtualization, together with a sophisticated distributed-caching mechanism, form the information-virtualization component.

Information virtualization could provide information from heterogeneous storage and I/O technologies. In a virtualized environment, applications are not bound to specific resources or endpoints. Applications could be provisioned on different endpoints, based on business requirements. To ensure that computation resources are utilized properly and are not wasted due to network-latency problems, the proper data should be made available to applications in a ubiquitous manner.

With the growth of Web 2.0 applications, there is an exponential rise in unstructured data, in addition to classic structured data. The unstructured data is by nature not suitable for the Relational database management system (RDBMS). Different types of storage facilities must be used, so that even unstructured data is indexed and quickly available. Also, the data must be replicated across systems; due to the network-latency data, it could become stale over a period of time.

Federated Virtualized Storage Pattern

The goal of the Federated Virtualized Storage pattern (see Figure 4) is to make data available across disparate systems through a pool of storage facilities. In this pattern, each data source is represented by its proxy data agent, which acts as an intermediary. The data agent can federate multiple data sources. It is also responsible for replication, the staging of data, caching, transformation, and encryption/decryption. The data agent has its low-level metadata that can contain the information regarding underlying data sources. The application maintains a data catalog that contains the higher level of metadata, such as replica locations. The data that is provided by the data agent can be stored in a local cache.

However, performance characteristics should be optimized and closely followed, as the aggregation logic takes place in a middle-tier server, instead of in the database.

Distributed Caching Pattern

For information virtualization, caching is an important aspect, as the performance is affected by network latency. Latency could be attributed to three factors: network, middleware, and data sources. When these factors are within the network boundary, the solution could be designed by considering the maximum latency.

The Distributed Caching pattern is ideal when applications are distributed remotely. In this pattern, a central cache repository that is synched with local caches could be maintained. Local caches do not need to maintain the full copy.

Another variant of the system in which caches are maintained with each instance is complicated, however, because of high transaction costs, especially when the number of application instances increases exponentially.

Virtual Service Pattern

The Virtual Service pattern uses the service intermediary to mask the actual service. The service metadata is used to model the virtual services. Service models are executed by the service-virtualization engine to provide different service behaviors at runtime.

Because all communication travels through the service intermediary, it is possible for this central node to intercept each service invocation and introduce additional service behaviors (for example, operation versioning, protocol mapping, monitoring, routing, run-time-policy enforcement, and custom behaviors), without having any impact on the client or service code.

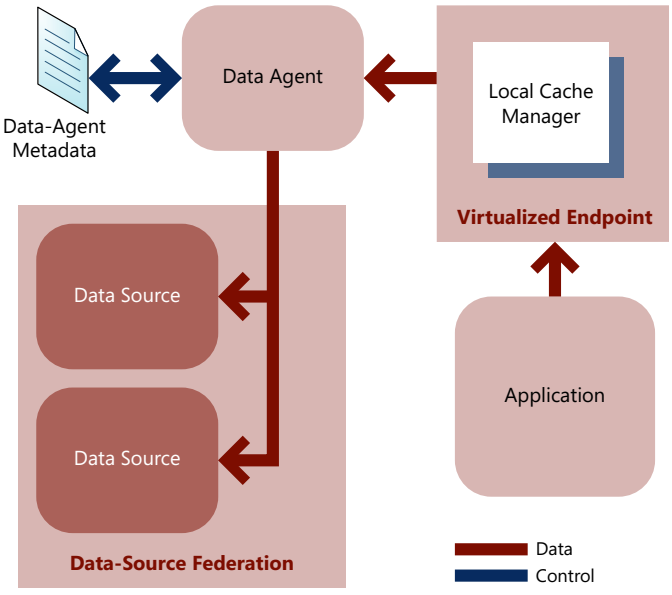
Thus, it is possible to provide service isolation by using this pattern. Machine-level isolation is ensured through the use of virtual machines. Machine-level and service-level isolations are in fact complementary. For example, all services from an organization can be configured to run at the service-isolation level, while the actual service implementations (which might be from different applications) could be isolated at the VM level.

Platform Virtualization and Related Patterns

Platform virtualization is made up of the actual virtualization layer and its monitoring component. It is responsible for management and orchestration of the workload on different virtual systems. This component ensures that the proper OS, middleware, and software stack is provisioned on the end systems that are used for execution.

The software architecture should complement the orchestration by providing application resource–usage metrics (feedback loop), as well as supporting deployment technologies for dynamic (de)provisioning.

Figure 4: Federated Virtualized Storage pattern



Dynamic Resource Manage Pattern

In the Dynamic Resource Manage pattern, service level agreement (SLA)–based provisioning is supported, based on application characteristics (such as application criticality, and current and historic resource usage). The Provisioner and Orchestrator components work with workload virtualization to ensure that the business requirements are met. The Provisioner builds the software stack and allocates storage and network bandwidth, based on the attribute information—such as OS, middleware, and storage—that is provided by the Application Profile. The Orchestrator uses business policies to acquire new endpoints, based on the workload metrics—such as current load and queue sizes—that are provided by the applications.

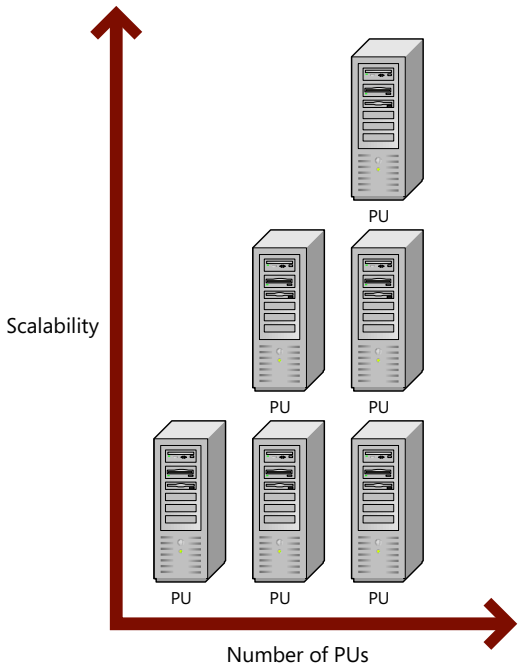
Architectural Styles for Virtualization

Traditionally, we have been using the *n*-tier model for designing enterprise applications. This model no longer meets our needs, due to the virtualized nature of modern systems, which introduces a high degree of agility and complexity. To address the new challenges—such as demand for dynamic linear scaling, high availability, and model-driven applications—we must explore new architectural styles. We examine two prominent styles: event-based architecture and space-based architecture.

Event-Based Architecture

In event-based architecture, services that constitute applications publish events that might or might not be consumed by other services. The subscribing services can use the event to process some task. The data to process this task might be made available through the Information Virtualization layer. The publishing service is decoupled from the services that are dependent on it. This makes the architecture so resilient that any part of the system can go down without a disruption to the overall availability. To ensure proper distributed transaction support without the addition of much overhead, apology-based computing could be used.

Figure 5: Linear scalability in SBA



By making heavy use of asynchronous communication patterns, durable queues, and read-only data caches, event-based architecture allows us to cope with the temporal unavailability of parts of the system. For example, the Windows Azure Platform provides the .NET Service Bus for queue-based communication.

For additional information, please refer to http://msdn.microsoft.com/en-us/library/dd129913.aspx.[4]

Space-Based Architecture (SBA)

The space-based architecture style is useful for achieving linear scalability of stateful, high-performance enterprise applications by building self-sufficient PUs. Unlike traditional tier-based applications, scalability is easily achieved by combining PUs.

Each of the PUs has its own messaging and storage facilities—thus, minimizing the external dependencies. Because each PU is an independent component, the sharing of state information across different software components is no longer needed. To ensure high availability and reliability, each of the PUs can have its failover instance running on different machines. Messages and data can be replicated asynchronously between PUs. (See Figure 5.)

A built-in SLA-driven deployment framework automatically deploys the PUs on as many machines as is necessary to meet the SLA. Deployment descriptors define the services that each PU comprises, their dependencies, their software and hardware requirements, and their SLAs.

The SBA architecture could be used to extend the tier-based applications to support linear scalability.

Conclusion

There is plenty more to be said about each of the topics that this article addresses. By this point, however, we hope that you have read enough to begin developing a software architectural stack on the constructs of virtualization, and see how you and your customers might benefit from it. Virtualization represents a new paradigm in software architecture—a model that is built on the principles of multitenant efficiency, massive scalability, and metadata-driven configurability to deliver good software inexpensively to both existing and potential customers.

This article is by no means the last word in software patterns that are based on virtualization. However, these approaches and patterns should help you identify and resolve many of the critical questions that you will face.

References

1. Amdahl, G. M. "The Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities." In *Proceedings of AFIPS Spring Joint Computer Conference* (April 1967). Atlantic City, NJ: AFIPS Press, 483–485.
2. Sanchez, Matt, and Naveen Balani. "Creating Flexible Service-Oriented Business Solutions with WebSphere Business Services Fabric, Part 1: Overview." *developerWorks* (IBM website), August 2007.
3. N.R. Jennings *et al.* "Autonomous Agents for Business Process Management." *Applied Artificial Intelligence*. Volume 14, no. 2 (2000): 145–189.
4. Chou, David. "Using Events in Highly Distributed Architectures." *The Architecture Journal* (Microsoft website), October 2008.

About the Authors

Nishant Thorat (Nishant.thorat@ca.com) is a Principal Software Engineer at CA in the Virtualization and Service Automation group. Nishant has filed several patents with the United States Patent and Trademark Office (USPTO) in the fields of desktop management, virtualization, and green IT. He is an active proponent of open standards and is a member of DMTF Cloud Incubator. His interests include cloud computing, virtualization, service management, and software architecture. Nishant received a bachelor's degree in Computer Engineering from the University of Pune, India, in 2001.

Arvind Raghavendran (Arvind.Raghavendran@gmail.com) is a Principal Software Engineer with CA and specializes in Microsoft technologies in Hyderabad, India. He has filed patents with the USPTO in the field of desktop IT management. His interests include virtualization and software-design patterns. Arvind received a bachelor's degree in Computer Science from Bangalore University, India, in 1998.

How Virtualized Corporate Networks Raise the Premium on System Reliability

by Phil Riccio

Summary

Most application managers have heard of virtualization, but few of them understand its implications for application availability and performance. Even IT managers who make their living at the top of the IT stack need a good working knowledge of this critical new infrastructure technology.

Introduction

It is hard to argue that the action in IT is anywhere else but at the application level. Applications are the frontline of IT, where technology touches workers, customers, and partners. E-mail, messaging and database servers, and business services such as online transactions and credit-card authorization: These are where companies make their money—or where companies lose their money, if their applications fail for any reason.

Perhaps in contradiction of the previous point, the hottest technology movement of the moment—virtualization—is not an application play, but an infrastructure play, far down the stack from applications. Virtualization is about remaking the data center. So, why should managers who are in charge of, say, Microsoft Office Exchange/Office Outlook or Office SharePoint, care what is happening that far down the stack?

The answer is simple: Because they will ultimately affect application operation and management. Residing on the glamorous end of the IT stack does not justify a benign ignorance of virtualization. Even a passing curiosity is not enough for application managers to do their own jobs properly if their organizations are considering virtualization. IT managers must be aware of the strengths and weaknesses of virtualization, because how they respond to virtualization will determine whether their applications continue to meet service-level agreements by providing the necessary application availability.

Virtualization adds flexibility and efficiency to the IT infrastructure. Its potential to reduce overall footprint in the data center by making better use of server resources benefits the entire IT organization by trimming costs and freeing up money and people to perform revenue-producing work. However, it also reverses a long trend that is marked by the beginning of the client/server computing era and extending until recently. Virtualization reintroduces single points of failure back to the IT infrastructure. Even when IT managers think about application availability in a virtualized environment, they often come to the wrong conclusion, which is that virtualization is its own availability solution. That conclusion reveals a risky misperception of the capabilities of virtualization and can lead to unacceptable levels of application downtime.

The consequences of downtime are high. IDC estimated the cost of an 11-hour IT outage at $1 million. Half of the medium-size businesses that were surveyed in a recent Gartner report revealed that their employees experience at least six hours of unplanned downtime per month, with corresponding losses in productivity. Large companies—those that have 2,500 users or more—report even higher losses: up to 16 hours of unplanned downtime per month.

Virtual Reality Check

Server-based architectures reduced the single points of failure that existed in mainframe- and minicomputer-based architectures. In the data-center architectures that still dominate today, applications either have a dedicated server or share a server with (at most) one or two other applications. Workloads have been physically partitioned to safeguard security, performance, and availability. A single server failure can knock a few applications offline, but does not have broad consequences across the application infrastructure.

The potential loss of performance and availability from a hardware failure is much more serious in a virtualized environment. Virtualization software can contribute to availability higher up the stack, at the application level. A hardware failure, however, will take down all of the applications on the physical server, regardless of how many virtual machines are backing up the primary application servers. The reason is simple: The ability to perform "live migration" is meaningless when the server crashes, because there is nothing to move from or to; everything is down.

Even if the virtual environment includes backups for primary physical servers, failures can create substantial downtime and erode performance. Applications are not available in these high-availability environments when a primary server is failing over to and restarted on a secondary machine, and all in-flight data is lost. Applications that run on secondary machines are also more likely to experience extra latency, because the secondary machines are more likely running their own primary applications, as well as the applications that were inherited from the failed server.

Transient errors are an additional availability and performance consideration in virtual environments. High-availability features that are built into some virtual environments often have the unintended effect of duplicating the error that caused a primary server to fail in the first place. The transient error migrates to the secondary server along with the processing load, where it can crash the secondary server in the same way that it crashed the primary server.

Even now, relatively early in the development of virtualization, companies are consolidating as many as 15 to 20 application servers on a single physical machine that is running multiple virtual machines. In addition to reducing the server footprint of the data center, this consolidation enables IT to provision extra processing capacity

more easily to meet peak requirements. Credit-card companies, for example, can be expected to process the majority of their transactions in the period from November to December, because of holiday/seasonal buying. Instead of building for that peak processing and leaving unused processing power on the table for most of the year, a virtualized infrastructure enables IT to provision a single pool of resources as fluidly as demand ebbs and flow. Virtualized environments can also take advantage of "availability on demand" by moving applications from standard servers to fault-tolerant virtualized servers via live migration. IT managers can do that when applications are needed during certain times of the week, month, or year.

In the long run, the virtualized model is a better server architecture for supporting the application infrastructure than today's one-server–one-application model. However, it also increases the number of applications that are affected by each crash. Even if the individual applications are not necessarily critical, their collective impact from a server failure can be very disruptive and costly to an organization. A virtual architecture that includes fault-tolerant hardware avoids most of these pitfalls. Fault-tolerant servers are architected to maintain uninterrupted processing, even if one or more components fail. That eliminates failover time and, with it, the downtime and performance degradations that occur during failover.

Beyond the application-consolidation factor, there are other reasons for application managers to have a solid working knowledge of virtualization. In most cases, application managers cannot count on simply virtualizing their existing architectures, as though conventional- and virtual-server environments function in the same way. Virtualization is not an automatic fit for some applications, especially for those that rely on real-time processing. An application that relies on "in flight" data—data that has not been written to disk—cannot simply be ported over to a virtual environment and expected to deliver the service level that it did when it ran on a highly reliable conventional-server infrastructure. This is not a failure of virtualized architectures, which were not designed to provide ultra-high reliability. It is a combination of over-reliance on virtualized environments that have been built from off-the-shelf hardware without built-in reliability functionality.

Understanding the Limits

At the architectural level, a well-designed virtual-server environment is inherently more resilient than a conventional-server environment. That is primarily because IT managers can create multiple instances of applications on different virtual machines and across multiple physical machines, to ensure that they always have a backup at hand should the primary fail. This is one of the most valuable properties of virtualization; a similar scheme in a conventionally architected data center would be prohibitively expensive and unwieldy to manage.

The resilience of virtualization can be deceptive, however. When a virtual machine has to be restarted, the information about application state as well as in-flight data is lost. It can also take anywhere from 3 to 30 minutes to restart a virtual machine, which is more downtime than many critical applications can tolerate.

"The more workloads that you stack on a single machine, when something does go wrong, the more pain an organization is going to feel because all those applications have dropped on the floor," said industry analyst Dan Kuznetsky, vice president of research operations at The 451 Group. "Even when people are thinking about availability and reliability, they do not often think about how long it takes for a function to move from a failed environment to one that is working. Next, when they choose an availability regimen—a set of tools and processes—they often do not think about how long it will take for that resource to become available again, once the failure scenario has started."

In-flight data loss is not the only drawback to relying on virtual machines as an application-availability solution. An error that causes downtime on a primary virtual machine can propagate to secondary virtual machines—crashing them as soon as they launch. Today's virtualization-software solutions do not isolate the cause of a failure or enable you to find the root cause, whether in software or hardware, which increases the chances that failures can be replicated between virtual machines. Virtualization software also is not designed to deal with transient (temporary) hardware errors such as device-driver malfunctions, which can cause downtime and data corruption when they are left uncorrected.

The point of all of this is not that virtualized architectures are inherently flawed. They are not. But neither the architectures nor the software that spawns them are designed to deliver the ultra-high reliability that many critical applications require. The hardware infrastructure has been largely overlooked in the equation. Without proper attention, it can cause trouble all the way up the stack.

Building Blocks and Foundations

The second factor that application managers must understand is the importance of the *actual hardware*. Hardware is such a commodity in conventional data-center architectures that application managers stopped thinking of it long ago. Servers were like the furnace: out there somewhere doing its job reliably, without needing a lot of hand holding. When server prices dropped enough and for each application to have its own server, the primacy of the applications was ensured.

Virtualization reverses that dynamic, back to a variation on the mainframe model. With so many processing loads relying on fewer hardware devices, the quality and durability of that device is pivotal to maintaining application uptime. Commodity servers often cannot support peak processing loads from multiple virtual machines, without slowing down or experiencing a failure.

Blade servers are often cited as the cure for this problem, but they are not. Blade servers assemble server resources in a more physically efficient form factor, but at their base they are still commodity servers. They share a common chassis, but they are essentially separate devices that have the same failover issues as stand-alone PC servers. They are not architected to communicate with each other automatically or to provide higher-than-standard reliability. "Standard reliability" would be at about 99.9 percent (at best) or, more commonly,

> *"The more workloads that you stack on a single machine, when something does go wrong, the more pain an organization is going to feel because all those applications have dropped on the floor."*
>
> –DAN KUZNETSKY, VP, RESEARCH OPERATIONS, THE 451 GROUP

at 99.5 or 99 percent. That amounts to more than eight hours of unscheduled downtime per year. Most critical applications require at least 99.99 percent uptime, and many require 99.999 percent, which is approximately five minutes of unscheduled downtime per year.

A virtualized server environment that supports critical applications with 99.999 percent application availability must include more than commodity servers. Even in a virtual environment, commodity servers cannot provide adequate uptime for critical applications.

Application managers need to know that their critical applications are implemented on a virtual architecture that includes fault-tolerant servers for maximum uptime.

Fault-tolerant servers are engineered "from the ground up" to have no single point of failure, and they essentially are two servers in a single form factor—running a single instance of an operating system, and where the application(s) are licensed once. A truly fault-tolerant architecture employs "lockstepping" to guard against unscheduled downtime. Unlike virtual machines or other hardware-based availability solutions—server clusters, hot standbys, and storage-area networks, for example—there is no "failover" time if one of the processors in a fault-tolerant server crashes. A true fault-tolerant architecture eliminates failover; it does not "recover" from failure. The unit simply continues to process with no lag or loss in performance, while it automatically "calls home" to alert the network-operations center.

Fault-tolerant hardware has proven itself over three decades of use in industries such as financial services, in which it provides continuous operation of crucial functions such as stock trading and transaction processing. And what was once large, expensive, and proprietary is now based on industry-standard components—fitting in industry-standard racks, and running industry-standard operating systems at industry-standard prices. Fault-tolerant hardware fits seamlessly into virtual environments as simply another server, albeit with ultra-high reliability built in.

There are software products that attempt to emulate fault-tolerant functionality; so far, however, software has been unable to emulate the performance of hardware-based fault tolerance. Software-based solutions have more resource overhead. They cannot stave off the effects of transient errors or trace the root causes of errors. Trying to achieve full fault tolerance through software also means sacrificing performance; applications that run software fault-tolerance products are restricted to using only a single core of a multicore CPU in a multisocket server.

Know Thyself

The inevitable march of virtualization through the data center does not mean that application managers have to become infrastructure experts. However, they do need to understand the availability requirements of their application mix. Trusting the management tools that come with new virtualization environments without weighing the uptime needs of each application is asking for lost revenue and disgruntled users. Virtualization-management tools enable application managers to shift processing loads to new virtual machines. However, if the underlying hardware infrastructure fails, that critical advantage of virtualization is lost. You cannot manage virtual machines on a crashed server.

No virtualization solution on the market today can deliver the 99.999 percent uptime that many critical applications require, and many cannot even guarantee 99.9 percent. In its research report "Beating the Downtime Trend," HP documented a 56 percent increase in unplanned downtime since 2005.

Many applications can endure as much as 30 minutes of downtime without a serious impact on the operations of the company. They are well suited to operate in a virtual environment without integrated fault tolerance. Inventory tracking, for example, can just catch up whenever the system comes back online.

If the application relies on in-flight data, however, the virtual architecture almost certainly requires fault-tolerant hardware. No virtualization solution on the market today can deliver the 99.999 percent uptime that many critical applications require, and many cannot even guarantee 99.9 percent, which allows for hours of unscheduled downtime.

In its research report "Beating the Downtime Trend," HP documented a 56 percent increase in unplanned downtime since 2005. This is occurring at exactly the same time that businesses are increasingly depending on IT systems as direct revenue streams and customer-service channels. With the tide already flowing against them, application managers have to know what they are facing when their companies start to virtualize the data center. While it is true that the money is made at the application layer, it can be lost just as quickly if the bottom layer of the stack—the hardware infrastructure—is shaky.

Conclusion

An IT manager who is educated in virtualization and its nuances is one who fully understands the importance of infrastructure planning, which includes application availability and hardware uptime.

About the Author

Phil Riccio (phil.riccio@stratus.com) is a product manager at Maynard (MA)–based Stratus Technologies and is responsible for the company's overall virtualization and Linux marketing strategies. He has been in the high-tech industry since 1981. Prior to joining Stratus, he was director, business development, at Mainline Information Systems—IBM's largest reseller—where his focus was on new and competitive opportunities for the IBM AIX and *x*86 platforms. Previous to Mainline, Riccio was North American marketing manager for enterprise-server products at Hewlett-Packard.

Virtualization: Without Control, Power Is Nothing

by Alan Maddison

Summary

As a technology, virtualization provides some compelling capabilities. After it has been deployed, however, it will provide some significant management challenges. Implementation of the Microsoft Operations Framework (MOF) can help organizations overcome these challenges.

Introduction

As a technology, virtualization provides a fundamental shift in the way in which an organization manages its infrastructure. While this is true for all organizations, there is often a distinct difference in the significance of this change for a company, depending on how mature a company is in its use of virtualization. For organizations that are beginning to mature in their use of and reliance on virtualization, there is much more significance in this change in management requirements. This is primarily due to the fact that, as the use of virtualization matures within an organization, there is often a desire to apply virtualization to a much broader range of systems—moving from what is commonly referred to as Tier 3 or noncritical systems to Tier 2 and Tier 1 or mission-critical systems.

As the last barriers to virtualization are eliminated for the most critical applications, IT organizations are finally able to fully embrace and deliver on the concept of IT as a Service. Embodied most recently in the concept of the dynamic data center, this aspect of IT as a Service fully utilizes the entire suite of capabilities that are native to virtualization, including dynamic workload management, automated provisioning workflows, and self-service capabilities. However, with the achievement of these capabilities and the inherent fluidity and dynamism that by definition become a daily part of your IT infrastructure, there is a corresponding increase in the operational complexity of your environment.

As this change occurs, the need to have a clearly defined management framework that is capable of providing structure is imperative. Unfortunately, while many IT administrators understand that, once virtualization is part of the IT infrastructure, there is a critical need to ensure that the tools that are used to manage that infrastructure are effective, they overlook the importance of an effective management strategy. Implementation of a comprehensive management framework can not only facilitate operational goals, but it can also make a tremendous difference in ensuring that the organization achieves its long-term strategic goals. Perhaps more importantly, a failure to implement an effective management strategy alongside the right management tools can lead to virtualization becoming another difficult technology to manage—one that does not meet expectations and fails to deliver the expected return on investment (ROI).

This article takes a look at the challenges of virtualization management and understanding how the Microsoft Operations Framework (MOF), if applied correctly, can ensure that you are effective in addressing your day-to-day operational concerns.

Microsoft Operations Framework

Currently, at version 4.0, the goal of the MOF is to provide practical and relevant guidance on the delivery of IT as a service. By combining Microsoft best practices with a service-management framework and then mapping these guidelines to real-world processes, the MOF can be used to help align IT with the needs of a business. Because of the granular nature of the framework, it can be successfully applied either across the entire enterprise or to a specific service. However, regardless of your approach, the detailed steps that the MOF provides make the process of understanding your environment very straightforward, in terms of the MOF components. Moreover, the framework covers the entire life cycle of an IT service—from conception and development to operation, maintenance, and retirement.

The MOF is structured to support the concept that the life cycle of a service is based on three ongoing phases—Plan, Deliver, and Operate—with one underlying layer that operates over the entire life of a service—Manage. Each phase defines a number of goals that are achieved when the phase has been completed. For example, some of the goals of the Plan phase are to ensure that IT services provide value, are predictable and reliable, are cost-effective, and can adapt to changing business needs. To achieve these goals, each phase contains what are known as Service Management Functions (SMF), which define the processes and people that are necessary to complete each phase successfully and move to the next phase.

For example the Plan phase provides a number of SMF, including Business/IT Alignment, Reliability, Policy, and Financial Management. Within each of these functions is a series of deliverables, each of which provides a clearly defined outcome. For example, the Business/IT Alignment SMF defines the deliverable of this function as being an IT service strategy. This delivery, in turn, will: allow IT services to be aligned with business processes and functions, define services that will directly support business needs, provide knowledge about how services will be used, and, perhaps more importantly, ensure that business units are satisfied. Oversight of all of these processes is provided within each phase through the use of Management Reviews.

Each phase includes at least one Management Review, to ensure that objectives are being met. By defining a series of controls, the

MOF provides a mechanism that allows Management to provide guidance and ensure that each phase is completed correctly. Underlying all of these phases is the Management Layer, which serves to ensure that best practices are followed, risk is managed, and the IT service delivers the expected value.

In addition to the value that the framework itself offers, there is a rich set of resources that complement and strengthen the MOF, particularly from a practical day-to-day operations perspective. Two of the most compelling sets of resources are the Solution Accelerators and Job Aids, both of which offer practical aids and guidance to real-world operational challenges.

Using the MOF to Overcome Management Challenges in Virtualization

There are many aspects of virtualization that make it compelling to IT organizations. Some of these reasons include reduced capital costs, improved operational flexibility, and efficient utilization of hardware resources. In fact, for most organizations, the initial impetus for virtualization often comes from this desire to consolidate servers and maximize the efficient use of hardware resources. As many IT administrators know, with a traditional physical infrastructure, there is a strong desire—and, sometimes, a technical need—to isolate applications and services onto their own hardware, to help avoid conflicts and maximize availability. From a business-operations perspective, many organizations see a need to isolate applications and services, because of the asset-ownership model that they use. However, regardless of these reasons, the end result is that it is very common to find that physical servers are tremendously underutilized.

Given the need of IT organizations to deliver services cost-effectively, this underutilization is always a tremendous concern. The attractiveness of virtualization and its ability to share workloads efficiently across the same set of hardware—while maintaining the desired level of isolation—thus becomes compelling. When organizations have accepted this rationale, they will move quickly to start the virtualization and consolidation process. For most organizations, this is a well-understood process that is extensively documented, and these initial deployments are nearly always associated with a formal assessment methodology that seeks to maximize the ROI of virtualization by using frameworks and best practices that govern the entire consolidation process. Unfortunately, this early commitment to following a standardized methodology often falls by the wayside, as IT and business units in general seek to take advantage of some of the capabilities of virtualization.

A good example of the impact of this failure to follow proven processes when it comes to managing a virtual infrastructure is virtual-server sprawl. As virtualization adoption has continued to increase, virtual-server sprawl has become widely recognized as a common symptom of the management challenges that IT departments face in a virtualized environment. There are generally considered to be two forms of virtual-server sprawl; each has a distinctly different cause, although at the heart of each is a failure or inadequacy in the management processes that govern the environment.

The first form of virtual-server sprawl that is denoted by the implementation of an ever-increasing number of virtual servers without proper operational controls can be caused by two factors. Perhaps the more important of these two factors is the impact of virtualization on the nature of asset ownership within an IT organization.

Traditionally, many organizations used an asset-ownership model under which individual business units owned the servers that ran line-of-business (LOB) applications. However, to take advantage of many of the sophisticated features of virtualization—such as dynamic workload management and resource pools, and the underlying centralization of workload management—it becomes inefficient and unwieldy for ownership of server hardware to occur at the business-unit level. By allowing IT to become the *de facto* owner of a centralized pool of hardware assets, organizations can achieve a great deal of flexibility.

The MOF is flexible enough to allow organizations to choose the level of formality with which to approach a project. It is imperative that you acknowledge the risk associated with such a fundamental change to ensure that you take a very formal approach to completing each phase of the MOF: Plan, Deliver, and Operate.

In fact, this concept of asset ownership plays a significant role in the delivery of IT as a Service. A centralized pool of hardware resources has a natural tendency toward a utility or service-provider model in which business units purchase slices of computing power and specific service levels. Perhaps more importantly, this mechanism is the only long-term approach that is both sustainable and capable of maximizing the ROI for a virtualized infrastructure.

The second factor that contributes to virtual-server sprawl is the ease with which virtual servers can be deployed. Capabilities such as cloning and templates dramatically reduce the deployment times, when compared with a traditional physical environment. Unfortunately, this capability, when combined with a lack of well-defined and easily understood management processes, leads to a loss of management control. When it is technically easy to deploy a virtual server and business operations are no longer forced to account accurately for the costs of LOB applications, the end result will be potentially very damaging to the sustainability of an efficient and cost-effective virtual infrastructure—the upshot of which is a dramatic increase in the number of virtual machines, without any insight into the financial costs and impact of provisioning.

The second form of virtual-server sprawl comes in the form of resource sprawl, which occurs when virtual machines are provisioned with a fixed set of virtual resources (processor, memory, and storage), without due regard for the underlying workload requirement of the application. This one-size-fits-all approach often leads to underutilized hardware. In fact, this approach creates the very situation that many organizations hoped to address by implementing virtualization.

Regardless of your particular problem (and, for many organizations, both forms of server sprawl are prevalent in their environments), a disciplined approach to IT service management

can remediate many of the issues. Specifically, in terms of the MOF, these failures can be mitigated by successfully defining the virtual machine–provisioning process as an IT service within the guidelines that the MOF specifies. By implementing all three phases (Plan, Deliver, and Operate), as well as successfully enforcing the Management layer, IT can begin to impose the necessary management processes to take ontrol of this problem.

However, before you start, it is important to understand that the management challenges that are associated with virtualization represent a significant risk in terms of the degree of change that it brings. Because the MOF is flexible enough to allow organizations to choose the level of formality with which to approach a project, it is imperative that you acknowledge the risk that is associated with such a fundamental change to ensure that you take a very formal approach to completing each phase of the MOF. When you have developed a comprehensive framework around virtualization as a technology and how it will be implemented in your environment, you will able to approach many of the day-to-day operational tasks—such as provisioning virtual servers—less formally.

What this means in practical terms is that tasks such as deploying a virtual server can be facilitated quickly and easily—thereby, allowing IT to maintain a high degree of responsiveness to changing business needs. Perhaps more importantly, it also allows the IT organization to achieve the level of control that is needed to be successful in the long term. This tiered approach to defining and controlling the different aspects of virtualization is also critical in ensuring the long-term viability of implementing the MOF. Without the ability to streamline the requirements of the framework as they relate to daily operations, there will always be an incentive to shortcut the very processes that ensure control.

Looking at the issue of virtual-server sprawl in more depth, there are some key areas of the MOF that clearly illustrate the value, flexibility, and strength of this tiered approach. For example, consider the fact that one of the primary drivers of virtual-server sprawl is that cost of ownership is devolved from business units in such a way that business units are often no longer required to perform a financial analysis of any new initiatives—particularly, those that are in a testing or development phase. If an IT organization had used the MOF to define virtualization as a service, it would have completed the Financial Management SMF as part of the Plan phase. Because the goal of the Financial Management SMF is to ensure that there is an accurate accounting of IT expenditures, with costs mapped to specific services, the completion of this activity would have ensured that the capital and operational costs that are associated with the physical hardware that was hosting the virtual servers was a known value.

Moreover, as part of the Deliver phase, the Build SMF would have produced a design that specified an architecture for the virtual infrastructure that, among other things, would have specified the number of physical hosts and identified the computing capacity of those hosts. Combining the financial costs that were developed in the Plan phase with the capacity calculations in the Deliver phase would allow an IT organization to create a dollar cost for each virtual server that is deployed. To leverage this formal and comprehensive analysis of virtualization as a service, the IT organization would then seek to implement a much less formal but nonetheless well-documented virtual server–provisioning process. While a less formal approach to a regularly occurring task is beneficial to the overall ability of IT to be responsive, it is nonetheless an important control that will allow IT to ensure not only that virtual machines are provisioned with the correct capacity, but also that all provisioning requests follow a standardized justification process. In the long run, these requirements will ultimately rein-in server sprawl.

A specific example from the MOF that highlights how this could be achieved can be found in the Deliver phase. Using the Envision SMF and applying it to the provisioning process would allow IT to require a written justification for the proposed virtual machine. The information that is required as part of this justification could include resource requirements (CPU, Memory, and Storage), a specific project name and business-unit name, and the name of a sponsor. While this requirement could be completed by using no more than a single page—perhaps, included as part of a workflow that supported automated provisioning of the virtual machine—it would ensure that every virtual server was justified and that costs could be apportioned to the correct business unit.

Perhaps more importantly, and despite the underlying simplicity of this step, it would provide a critical checkpoint at which IT could exert and maintain control of the number of virtual servers that were being deployed and the resources that were being allocated to those servers. With these controls in place, it is possible to overcome many of the management challenges that are associated with virtualization, including the most common of all: virtual-server sprawl.

Conclusion

An effective management strategy is a critical requirement of any well-run IT organization. The introduction of virtualization in an environment only serves to underline this requirement, as many of the capabilities that make virtualization attractive lend themselves to issues of control. While organizations that are new to virtualization are often unaware of these issues, in many organizations in which virtualization plays an increasingly important role, there has been a growing realization that, as a technology, virtualization brings a unique set of management challenges. Some of these challenges can be addressed through selection of the right management tools, but many of these challenges require adhesion to a management strategy that is based on a comprehensive framework that lends itself to defining the means to achieve the management goals both effectively and efficiently. The MOF provides such a guide and, when it is applied correctly, can help IT organizations impose the controls that are required to maintain a virtual infrastructure successfully.

About the Author

Alan Maddison is currently a Senior Consultant with Strategic Business Systems, Inc., a division of Brocade. Specializing in Microsoft technologies, he has a 15-year track record in the IT business across many different industries. As a Consultant, he has delivered services on behalf of some of the leading IT OEMs to many companies across the country. Alan also enjoys writing about technology, and he is the author of a number of articles. You can send email to him at amaddison@sbsplanet.com.

Models and Application Life-Cycle Management

by Clemens Reijnen and Ir. Robert Deckers

Summary

This article discusses the creation and maintenance of consistent, correct, and communicative software architecture by using Microsoft Visual Studio 2010.

Introduction

Setting up, validating, and maintaining consistent, correct, and communicative software architecture in its context and interaction with its environment is no trivial task.

A new architecture needs to be created. But how do you know if this architecture is good? Covering all areas, dealing with all the requirements and quality attributes is difficult; but forgetting to discuss one or more is easy and will leave space for assumptions.

Because of changing requirements, your architecture needs to be changed. But how do you know if it is still consistent? Unstable documents and models will confuse the development and test teams, who will not know what to build or what the intention of the system is.

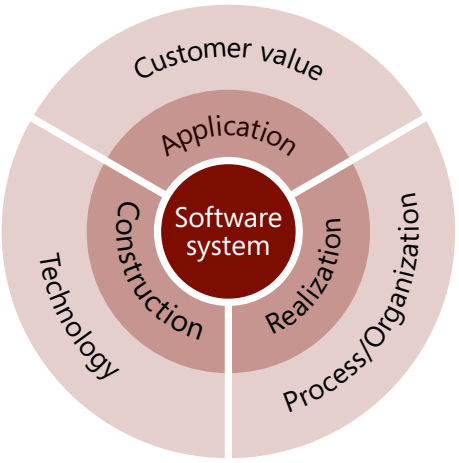
How can you make practical use of the Microsoft Visual Studio 2010 modeling capabilities in the application life cycle and an architectural approach, to cover these challenges and create good software architecture?

Good Software Architecture: An Approach

Software architecture gives direction to the realization of a software system, which must fulfill business goals. This software architecture has to be correct, consistent, and communicative when it has to cover the needs of these business targets. When you focus on these aspects of software architecture, you can determine that the architecture is indeed good when it is:

- **Correct.** The architecture is based on validated statements that cover the system environment and, especially, stakeholder interests. These statements are prioritized. The final architecture is the balance between all of these prioritized statements.
- **Consistent.** Architectural decisions do not clash. The architecture is checked for completeness, consistency, accessibility, and translatability into application life-cycle management (ALM) artifacts, configuration items, and work items such as test cases, sources, and maintenance documents.
- **Communicated.** The architecture is communicated and checked with the stakeholders, who know their relationship with and influence on it. The physical architecture and architectural decisions must be communicable, and stakeholders must find the right spot for them.

Figure 1: Reasoning model



Reasoning Model

To support the creation of a correct architecture, we use a reasoning model (see Figure 1) that is based on DyA|Software.[1] This model helps the software architect with decision-making and—even more importantly—with analysis of the correctness and consistency of the architecture.

During the application life cycle of a system, including the realization process (the software-development life cycle [SDLC]), a system is influenced by changes in requirements and planning, differences among project teams, and more. All of these changes must be fitted within the architecture and vice versa. Given these different changes by and for different people, the widespread diversity of aspects makes architecture creation an interesting task. The reasoning model helps to cover all of these aspects of software architecture.

There are three dimensions about which an architect must think in relation to the system. These are the application domain, the construction domain, and the development domain:

- **Application domain**—Addresses what the system means for the customer—the end user of the system. This domain describes the goals, needs, business value, use cases, functionality, and quality of the system.
- **Construction domain**—Addresses the elements from which the system is built—the components, the interfaces, and so on—as well as the design patterns, necessary infrastructure, and other technologies.

- **Development domain**—Addresses the processes or methodologies that are used for realizing the system. These include organizational structures, road maps, knowledge, roles, people, and outsourcing.

Decisions that are made in these three domains all have their influences in the environment of the system, the borders of the system, or the system itself. Every decision in a domain has to be related to a decision in the other domains, so as to get a correct and consistent architecture.

For example, a statement such as, *"The customer must be capable to edit his/her own information"* (which has its place in the customer-value section) will influence all of the different domains. The application must have a capability for user logon, and there must be a subsystem for customer information that is based on the technology that is used. Construction statements are made that have an influence on team members, who must have experience with these types of application and technology. As for the planning, the statements also have their influences, which may result in a team per subsystem or a component per iteration.

Every domain is connected; statements about one domain must be covered by statements in the others domains to get a correct and consistent architecture. When a statement is not related to another, you have a dead end; when two statements interfere, you have a challenge.

Design and Realization in the Application Life Cycle

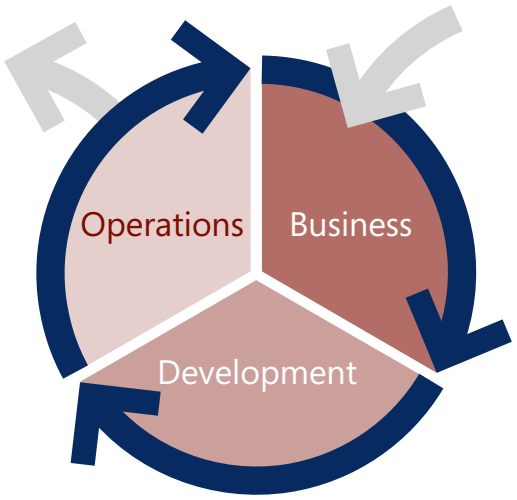
As described in the previous paragraph, good architecture is based on a set of statements—requirements that are connected to and influenced by each other—and is translatable into consistent ALM artifacts (see Figure 2), so that people to whom the architecture is communicated will understand it and can add their knowledge to it. However, none of this can be realized in an ivory tower. To design and realize good software architecture, one that has value throughout the application life cycle, you must work together with—and be connected to—all team members, with their different roles, as well as all influencers outside the system.

During the design, realization, and maintenance of a system and its corresponding software architecture, you must write down (that is, document) the architectural description—the whole set of architecture statements and the prioritization of these statements, for validation and communication. How the architectural description is documented and what kind of documents and tools are used depends on the project methodology that is used. However, to get information about consistency and checks (if there are any contradictions), it is necessary to have a repository that has history capabilities or version-control capabilities that can collect statements.

This ambiguities characteristic counts not only for the software architecture; the combination of architecture with code, architecture with requirements statements, and architecture with planning also has to be seamless. You can say something about the quality of the system and architecture only when you can see the whole picture and the relationship among the different parts.

It is useless to create and use a structure or a terminology for requirements that does not fit the separation of the proposed architecture, so that pieces of the system do not relate to a requirement. The same counts for the project road map and work packages in the project planning; these also must fit the system architecture. A team per subsystem that has specific knowledge is one way of planning.

Figure 2: Application life-cycle management (ALM)



Valuable Models

It is possible to put all of this—correct, consistent, and communicative architecture—in place without the use of any kind of tooling. At a minimum, a whiteboard can be used (and sometimes communicated by making pictures of it). However, as agile and flexible as this way might be of making an architecture work, it does not fulfill all of the needs for good architecture, as previously described. Traceability, validation, and even communication of these models are difficult. This will result either in stricter communication and project-management rules or in chaos. The models have value only at the beginning of the application life cycle and in the application domain (see Figure 1 on page 14). There is no possibility to get traceability and validation among the different statements and no possibility to say something about the quality of the system.

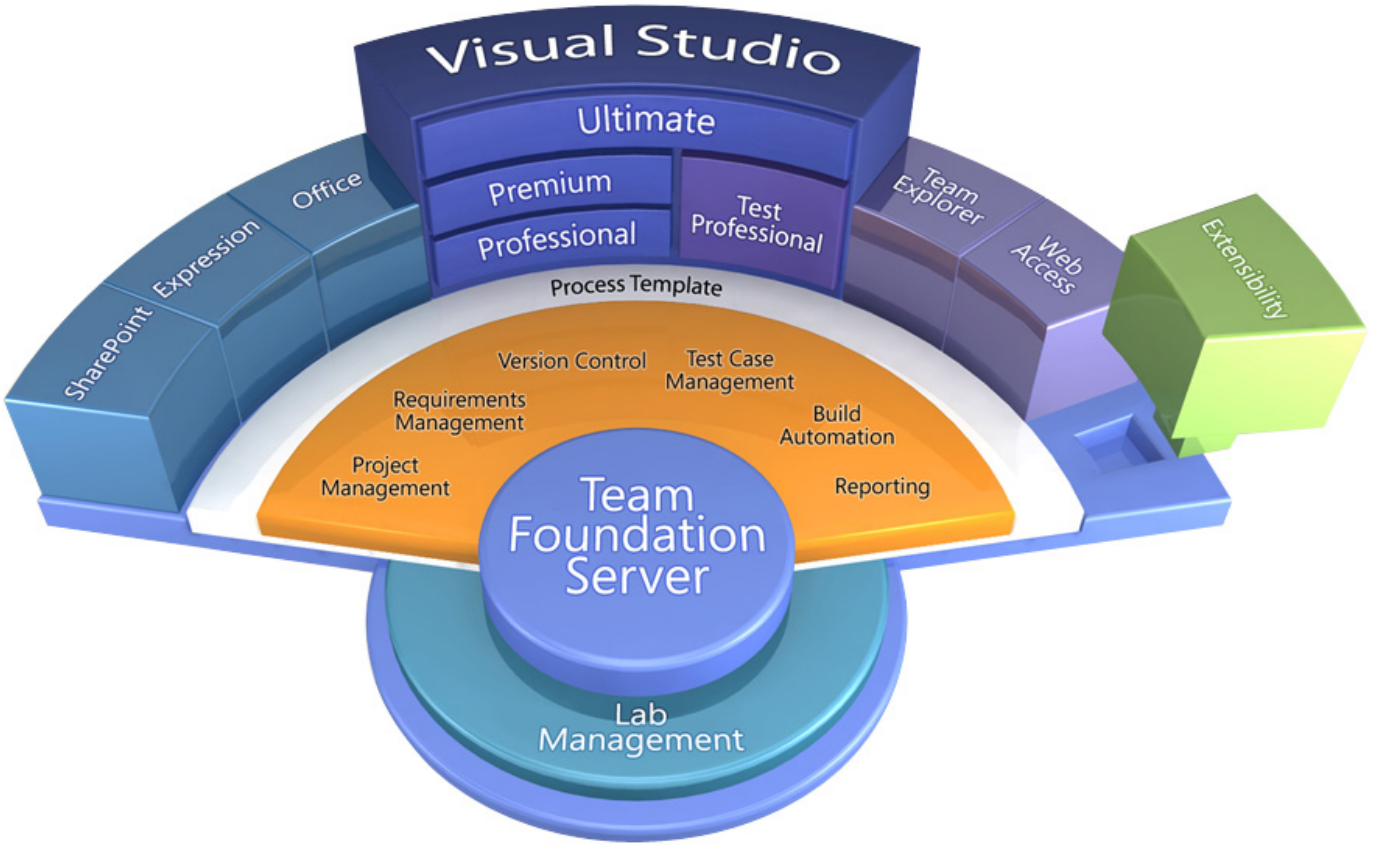
A set of disconnected tools (every role in the application life cycle has its own toolset) has the same missing capabilities. While the maintainability of models is much easier throughout the life cycle of the project, it is difficult to get the validation in place. In validation, as described in an earlier paragraph:

> *"Every domain is connected; statements about one domain must be covered by statements in the other domains.... When a statement is not related to another, you have a dead end; when two statements interfere, you have a challenge."*

This validation must be performed manually, which is difficult. It is difficult to get a statement about the correctness and consistency of the architecture and corresponding models.

From the use of a whiteboard to the use of more sophisticated tools, models will get more and more value in the application life cycle when they can be validated for correctness and consistency. Quality is maintained throughout the application life cycle; and, when a tool supports this value with sophisticated features to connect and communicate with other roles in the application life cycle (see Figure 2), this value will be easier to maintain. Models that are created by using a whiteboard have value only in the application domain. Models that are created and used by using disconnected modeling tools have a wider value; they can be used and reused during the life cycle, although validation is difficult. The use of a connected tool, in

Figure 3: Visual Studio 2010 ALM tools



which every role in the application life cycle can look at and validate its statements, will create even more valuable models.

The Visual Studio 2010 ALM toolset (see Figure 3) is one such connected and integrated ALM environment. For a long time, there has been support for processes and work items (the realization domain, as previously described) by using Microsoft Team Foundation Server (TFS). The configuration-item support for source code is one of the first features that everybody uses; it supports the construction domain, and the connection between change sets and work items covers a validation mechanism that is needed to check the consistency between the construction domain and the realization domain. With the support of models—UML, Direct Graph diagrams, and Layer diagrams in Visual Studio 2010—capabilities to support the application domain are also added. Not only do the models add this support for creation of good architecture, but the connection of the configuration items and work items to these models makes them more valuable. This connection—the integrated work environment—makes the models so valuable. While this is the first version of UML support in Visual Studio (and, surely, other tools have more sophisticated and extended UML-modeling capabilities), the seamless connection to the other roles in the application life cycle makes it a true game-changer.

The paragraphs that follow discuss a work pattern for how to use Visual Studio ALM features to check statements in the different domains, as described in the previous paragraph (see Figure 1 on page 14), and how you can validate and check your architecture in a seamless, collaborative, and easier way.

Models in the Life Cycle

Visual Studio modeling tools cover more than UML support. Besides several UML diagrams (Activity, Use Case, Sequence, Class, and Component), there is a *layer diagram* that checks the implementation and an *architecture explorer* that analyzes the system. The architecture explorer lets you visualize any kind of reference, dependency in your solution, or other assemblies. All of these models live together with the code as configuration items in a central repository that is accessible by every team member. This kind of capability will support the creation of a correct and consistent architecture, in which every decision in one domain of the reasoning model has a related decision in another domain. This support comes from the fact that Visual Studio 2010 has not only UML tool features, but also capabilities to link modeling artifacts to other ALM artifacts, such as work items and other configuration items.

This capability—linking and referencing configuration items and work items—and the capability for every role in the ALM to have access to the information make the models valuable in the full life cycle of the application. This definitely supports the creation and usage of a good architecture.

User Stories, Requirements, Work Items, and Models

In the application domain (see Figure 1 on page 14), statements are made about customer value. The domain addresses what the system means for the customer—the end user of the system. This domain describes the goals, needs, business value, use cases, functionality, and quality of the system. These "customer value" statements can be named *user stories* in an agile project or *requirements* in a more

Figure 4: Linking work items to model elements

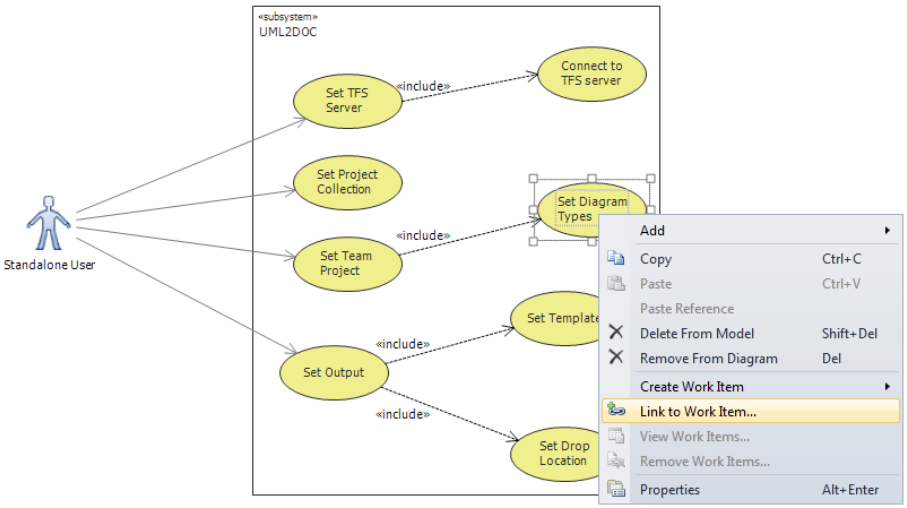


Figure 5: User-story work-item type

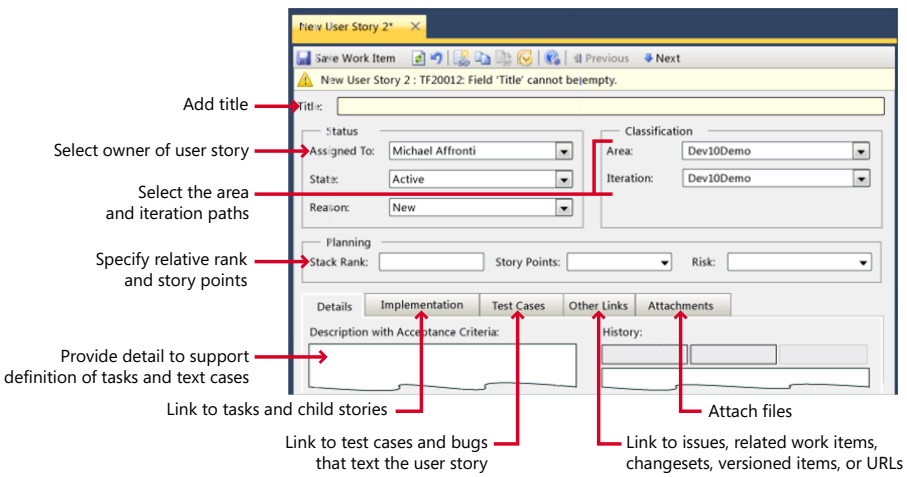
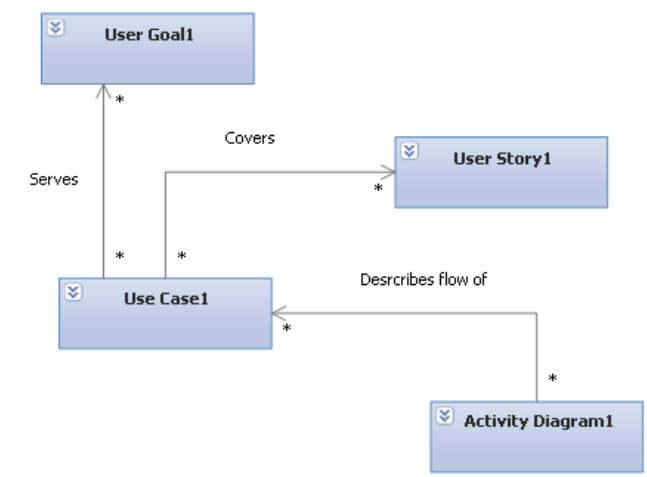


Figure 6: Metamodel: Application domain, user goal (connected with use-case diagram), and user-story work item



formal project. In both types of project methodology, these necessary pieces of customer value are captured in work items within TFS (see Figure 5).

To get everything in place, to visualize all of the aspects of the customer-value domain, create in an iterative process the high-level scope—the use cases.[2] With Visual Studio 2010 in place, you can also capture this information about actors and processes in a use-case diagram. Linking these diagrams to the work items will give both the creator and the reader a better overview, if all important spots are targeted (see Figure 4). User-story work items that do not have a connection to a use case (and vice versa) give a good indication of inconsistency.

Visualized in a metamodel, this connection—among user goals, use-case diagrams, and user-story work items in the application domain of the reasoning model—will resemble what appears in Figure 6. (Please refer to Sneak peek at the first "Feature Pack" for VS2010 Visualization and Modeling Tools.)

Use cases are descriptions of system functions that cover user goals and give a good starting point and overview of what the system must fulfill. They are discussed with the customer and supported by activity diagrams,[3] which can also be linked to work items and from use cases. This linking of model elements, diagram shapes, and lines with work-item types such as user stories and requirements is a key feature for keeping the architecture consistent. For example, a change in a use-case diagram points to a corresponding requirement that has a breakdown in tasks or subrequirements (this will help to keep the different corresponding artifacts consistent). Another example can be a project manager who looks at use cases that do not have a corresponding activity diagram or user story. The manager knows that there is something wrong.

We can start to break down the customer needs into tasks and supportive software components. Create the corresponding statements in the other two domains.

Components, Layers, and Scenarios

In the previous paragraph, the application domain and user needs are captured and modeled by using work items and UML diagrams. In the construction domain, statements must be set according to technology, system elements, patterns, components, and interfaces. All of them must be consistent with the application domain; surely, some iteration between them will take place.

Adding the construction domain to the metamodel will result in what appears in Figure 7 on page 18. A scenario is supported by a function that is implemented by a component. This results in a component diagram.

A layer diagram is created up front to cover layer-diagram patterns that are used in the solution. The layer diagram in Visual Studio 2010

Figure 7: Metamodel application domain and construction domain, covered by Visual Studio 2010

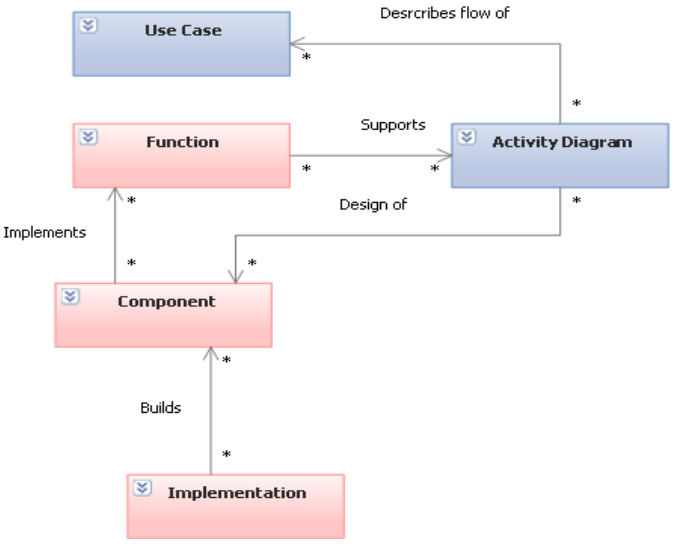


Figure 8: Visual Studio 2010 layer diagram

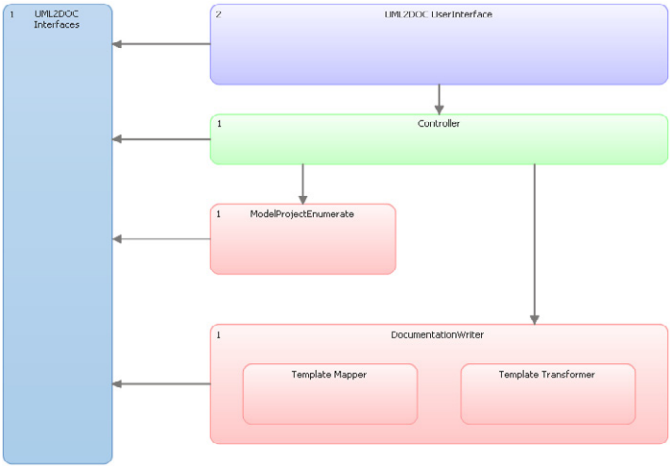


Figure 9: Component diagram

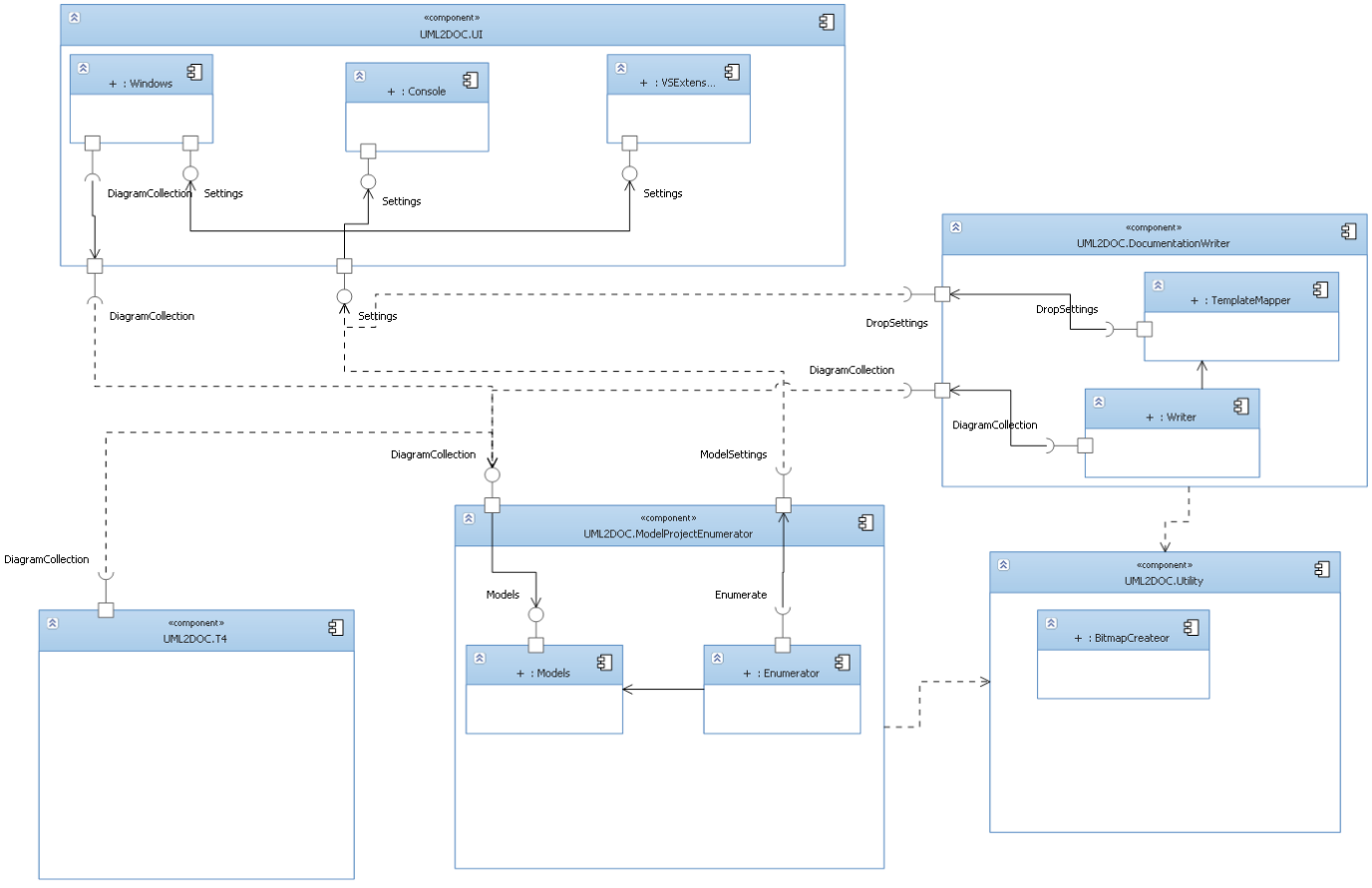
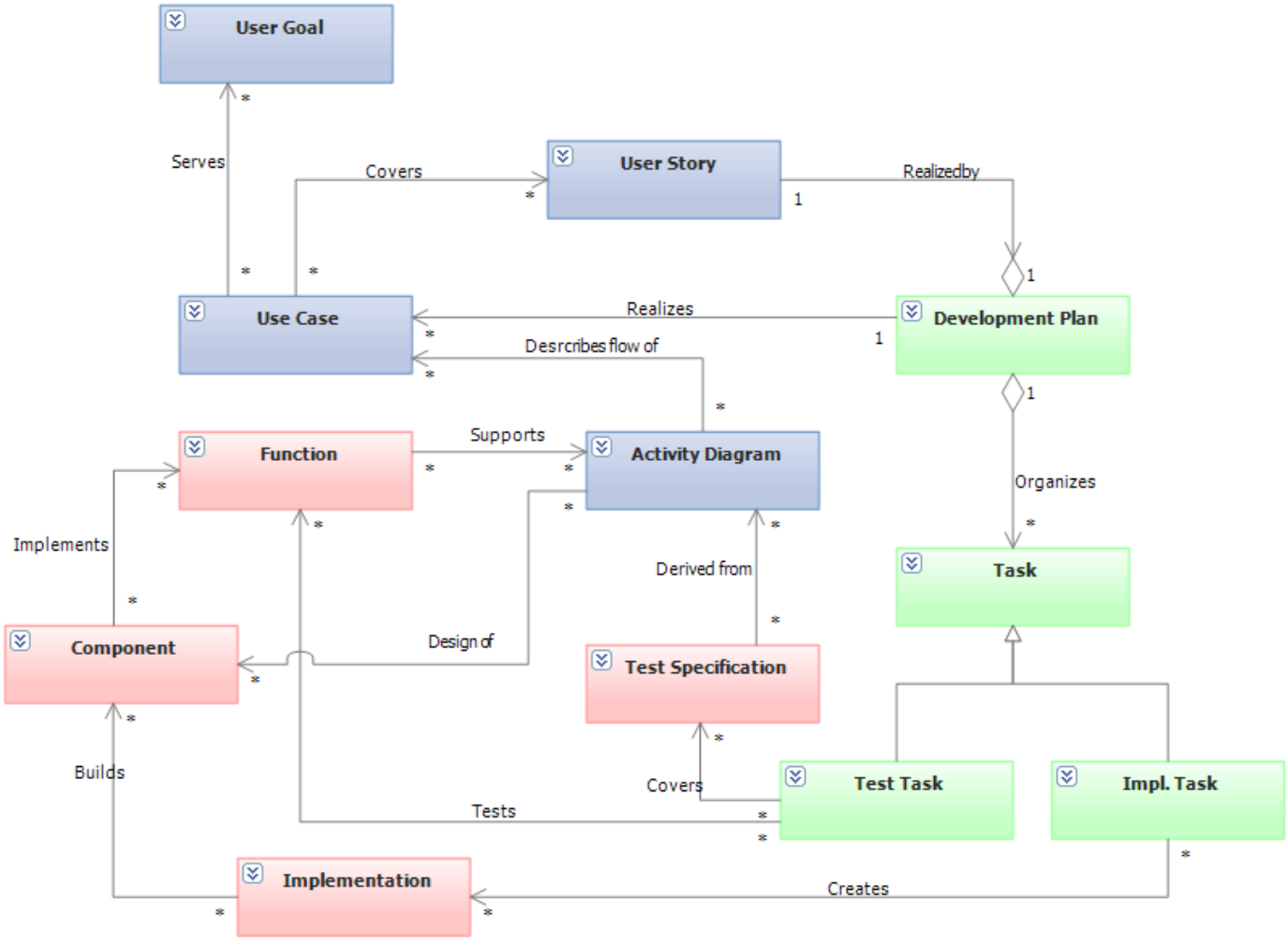


Figure 10: Metamodel: DyA reasoning model, covered by Visual Studio 2010



has a great feature: It can validate the implementation. Using this, the layered architecture can be checked for consistency between design and implementation, which definitely will help to maintain a good architecture (see Figure 8 on page 18).

Together with the layer diagram, a component diagram is drawn. While the layer diagram is not UML-compliant, the component diagram is. A component diagram virtually holds the same kind of information as a layer diagram, although it is much more detailed with regard to the communication ports and internal parts (see Figure 9 on page 18).

To have good architecture, we must have statements in the construction domain that cover statements in the application domain. For example, when a statement such as, *"As a user, I want to be able to set the output template, so that I can support different formats"* does not have a component and interface that support this functionality, you know that the architecture is incomplete or incorrect. There are two ways to great a component diagram.

You can start drawing sequence diagrams that are based on the most important application flows of the system, based on the use cases and corresponding activity diagrams. With the messages and components that are used in the component diagram, we can start drawing the components, parts, and interfaces.

The other way is actually the opposite direction, based on the layer diagram you can start drawing the main components and interfaces. From there you can start replay the main scenarios by using sequence diagrams. Checking if the are some inconsistencies in the component, parts and interface design. The Visual Studio 2010 component diagram has a feature which creates lifelines for components on sequence diagrams.

The interaction between the artifacts that are used in the application domain, use cases, activity diagrams, and linked work items (user stories or requirements) and the artifacts that are used in the technology to create that high-level architecture is a true iterative process. To get the statements in one domain to fit the statements in the other domain, we must go back and forward between them. As described in the metamodel (see Figure 7 on page 18), the activity diagram in the application domain is a realization of the use case—the user needs. These needs are supported by functions that are implemented by components. These components, in turn, can be validated by replaying the scenarios in the activity diagrams by using sequence diagrams—thus, creating a setting from which you can check the architecture for consistency and correctness—supported by Visual Studio 2010 capabilities for linking and validating (layer diagram) the different artifacts with each other.

Methodology, Tasks, and Road Maps

In the application domain, statements are made about customer value. The application domain addresses what the system means for the customer—the end user of the system. This domain describes the goals, needs, business value, use cases, functionality, and quality of the system—written down in work items such as user stories or requirements and drawn up in use-case diagrams and activity diagrams, all of which are connected by work-item links. The artifacts in the construction domain—such as components, layers, and sequence diagrams, and the resulting implementation code and configuration files that fulfill these needs—are validated by replaying scenarios from the activity diagram, which is also connected. So, we can check the architecture for consistency and correctness between the application domain and construction domain and, certainly, within each domain itself.

As mentioned in the first paragraph, a good architecture also covers the development domain. It must address the processes—the methodologies that are used for realizing the system—including organizational structures, road maps, knowledge, roles, people, and outsourcing.

Adding the development domain to the metamodel will result in what appears in Figure 10 on page 19 (starting at the bottom of the model). A component is built by an implementation that is a result of an implementation task, which is part of a development plan. This development plan holds the methodology and tasks that must be executed to realize the use cases. In Visual Studio 2010, the work item–type user story (or requirement), which is used to hold the user needs, can hold several other (hierarchical) work items (see Figure 5 on page 17)—the implementation and test-case tabs. Using this connection, we can organize the development plan based on the tasks that must be executed to realize the functions that are needed for that use case. And this makes the circle complete.

One other addition to the metamodel is the test task, which validates the function that supports the scenarios in the activity diagram. This test task is part of the test specification that is derived from the activity diagram—using some kind of test methodology, such as a process cycle test. These test tasks are again part of the work items that serve the user goals, user stories, or requirements.

With this, every domain in the DyA|Architecture reasoning model is covered, and validations are checked against each other. When, for example, there is a completed implementation task that is not part of a user story, we know that we have dead code. The same goes for test cases. When a test case does not belong to a scenario that is written down in an activity diagram, we know that there is not a corresponding statement in the application domain. With this, we have a "kind of" methodology-agnostic mechanism for validating an architecture with regard to consistency and correctness.

Conclusion

It is possible to maintain and organize the separate responsibilities—project planning in a planning tool, architectural models on the whiteboard or drawing program, requirements written down in documents—but the important connection is not there. Validation of statements is difficult and carries a risk of creating an inaccurate, unreliable architecture. As an application life-cycle tool, Visual Studio 2010 covers solutions for many of the challenges that exist, as described in the preceding article and metamodel. Not everything is supported by automation, such as automation of the validation between the implementation and the layer diagram; the links between models and work items must be maintained manually. But having all of the roles working with just one single repository of configuration items and work items is a great start.

We do expect more validations of statements over the different domains, which will make the Visual Studio 2010 models that are used by projects even more valuable in the future. But this is a great way to start creating a good architecture in the application life cycle.

References

1. DyA: DYnamic Architecture, an architectural approach by Sogeti Netherlands (http://eng.dya.info/Home/).
2. See http://en.wikipedia.org/wiki/Use_case.
3. See http://en.wikipedia.org/wiki/Activity_diagram.

About the Authors

Clemens Reijnen is a software architect at Sogeti and was awarded Microsoft Most Valuable Professional (MVP) by Microsoft. With over 20 years of experience in IT, he has a broad background in software development and architecture. Clemens runs a technical blog on www.ClemensReijnen.nl and is coauthor of *Collaboration in the Cloud: How Cross-Boundary Collaboration Is Transforming Business* (Groningen: Microsoft and Sogeti, 2009). You can reach Clemens at Clemens.Reijnen@Sogeti.nl.

Ir. Robert Deckers is a senior software architect and architecture consultant at Sogeti. He is founder of the DyA|Software approach and is writing a book (to be released in summer 2010) about this approach. You can reach Robert at Robert.Deckers@Sogeti.nl.

Getting the Most Out of Virtualization

by Greg Hutch

Summary

Virtualization is a powerful architectural technique and comprises a set of technologies that can drive a range of benefits.

Introduction

Most organizations fail to realize the full potential of virtualization.

While various virtualization technologies have been around for a long time, we now have the pieces that are required to drive complete virtualization strategies that are optimized for different business scenarios.

The origin of virtualization—and the most common reason for an organization to pursue virtualization—is efficiency and the potential cost savings. The following virtualization strategy was focused on reducing costs and increasing efficiency.

Virtualization for Efficiency

The organization needed to drive down all costs, including IT, as much as possible. At the same time, it was under pressure from its customers to provide more online services. The organization needed to get as much value from its assets as possible.

We started with monitoring and analysis of the existing server workloads. We quickly found that the existing servers were old single-purpose servers that had very low CPU-utilization rates; they had been sized to have enough capacity for their peak utilization, which was rarely needed. Report servers were sized for a peak that occurred only at month's end. Backup servers were idle during business hours. File servers never used their full processor capacity and were almost idle outside of business hours. Security servers were busy only during business-day mornings. The disaster-recovery servers were sized for their emergency workloads, but never ran at that level outside of testing. It was very clear that virtualization could help us consolidate these workloads in fewer processors. (The situation with memory was comparable.)

It would have been easy to cut the number of servers in half, but we wanted to optimize virtualization. In the best case, we were able to virtualize and consolidate 30 of the older single-purpose servers in a single new server. This was an unusually aggressive consolidation approach, but it proved supportable.

Before virtualization, each server supported an application or service that had a unique set of service expectations. A small number of applications had specific high-availability requirements, and special servers had been allocated to provide that service. High-availability clusters generally demanded specific configurations, special versions of software, and an active/passive mode that drove (at best) 50-percent productivity from the clusters. Moving to a virtualized approach meant that we could reduce the overhead and still provide fault-tolerant platforms. To maximize consolidation, we standardized the service levels. This allowed us to consolidate workloads that had identical or compatible service levels.

Another cost-reduction opportunity revolved around networking. The large number of physical servers led to a large number of network adapters, ports, and media types. It was clear that virtualization could allow us to reduce the number of network connections, ports, and media types.

Another cost-saving opportunity was the ability to buy capacity "just in time." In the past, servers were ordered with extra resources, because changes were challenging and time-consuming.

When we had driven the reduction in servers and networking, there was far less demand for data-center resources. We needed far less physical space, electrical service, uninterruptible-power-supply (UPS) capacity, and air-conditioning capacity. The result was that we could cancel the planned data-center upgrades. We did need to change the configuration of the data center, to avoid the creation of hot spots as we consolidated. While the project was not driven by a "green agenda," the efficiencies did contribute to a significant data-center "greening."

We also discovered that virtualization led to larger-than-expected licensing savings. Increased CPU efficiency and a reduction in the number of servers allowed us to reduce significantly the license requirements for the organization.

In summarizing this experience, we learned that virtualization for cost savings appears to be applicable to almost all organizations and leads to larger-than-expected savings. Furthermore, when we benchmarked with other organizations, we learned that most organizations fail to drive out all of the potential cost savings simply because they do not plan sufficiently.

Virtualization for Increasing Agility

Virtualization is also a very powerful approach toward increasing the IT agility of an organization. This example of a virtualization strategy comes from an organization that was frustrated by long "time-to-solution" times. The strategy for agility was based on the same fundamental technologies, but it required even more planning to optimize. In the first example, we used virtualization to put the largest possible workload on the least expensive resources. In this example, we use virtualization so that we can quickly deploy and change the allocation of resources to workloads.

Before virtualization, each project required several weeks to acquire new hardware that was ordered to the specific needs of the new service, application, or project. The organization frequently encountered capacity problems, and significant downtime was required when capacity had to be added.

Improved architectural standards were required to ensure that new applications would be deployed quickly. We created two standard virtual servers. This allowed us to cover the broadest range of commercial applications and support the management tools that we needed to achieve the agility and efficiency targets. Virtualization allowed us to create pools of CPU and memory capacity that could be allocated quickly to a new application or a temporary capacity issue. This allowed the organization to purchase generic capacity and allocate to specific needs, as required.

Virtualization also supported "on-demand" capacity. We could move quickly to allocate resources to temporary demands. We needed to plan carefully for the current and expected network demands, as network changes in a virtualized environment are challenging. As of this writing, there are recent products that promise to bring network virtualization to the same level as servers.

The virtualization tools allowed administrators to move running workloads easily between hardware platforms. This allowed administrators to move a workload, upgrade a platform, and move the workload back, with no user impact. The ability to add resources, allocate them, and maintain them without user downtime dramatically changed the scheduling of project activities. This capability allowed us to perform application- and server-setup tasks without having to schedule around downtime windows or service levels.

One of the first opportunities to increase agility was in the development and test environments. We standardized testing and promotion processes. Before virtualization, each application project tended to design its own approach for testing and promotion. The approach was "optimized" to the needs of the specific project—minimizing resource cost and critical path. This tended to introduce long setup times for the test environment and unique promotion processes. After virtualization, we had a standard approach to allocate and share environments. We also had standard processes to promote applications. The time that is required to set and rest these environments was dramatically reduced.

Increasing the agility of CPU and memory deployment would have meant little without improving the agility of storage. It was critical to have as little local storage as possible. It was also critical to consolidate storage-area-network (SAN) islands. Local storage and separate SANs caused delays and complexity during deployment of new applications. Creation of large pools of storage that had the required performance and reliability characteristics was essential for agility. We created redundant high-performance paths between the servers and storage, so that we could quickly assign storage to server resources.

In summarizing this experience, the biggest benefit was a 96-percent reduction in the time that was required to deploy a new application. A dramatic example of this new agility occurred shortly after virtualization was complete. A project required a very significant increase in capacity. Sufficient generic capacity was ordered. While the equipment was being shipped, the organization's priorities and projects changed. When the new generic gear arrived, it was easily allocated to the new projects.

Planning

As with any architectural endeavor, full value is realized only when the business benefits are aligned and you start with good information about the current state.

The origin of virtualization is efficiency and potential cost savings. Most organizations fail to drive out all potential cost savings; they simply don't plan sufficiently. Emerging network-virtualization products promise more efficiency and manageable server virtualization.

Plan Your Benefits

We needed to be clear on the benefits that we were trying to achieve. For example, a project that is focused on cost reduction will be more aggressive in server consolidation than one that is focused on agility. In our experience, failure to clarify virtualization objectives leads to the project stopping before the full benefits are realized. Those projects tend to virtualize only the "easy" workloads and servers.

Understand the Current Environment

To plan for optimal virtualization, we needed solid information about the current (and planned) workloads. We also needed good information about the network, to avoid designing a solution that would create network congestion.

Create the New Server Standards

It was well worth the time and effort to establish the new server standards at the outset of the project, so as to support the required workloads and drive virtualization benefits.

Plan the New Service Levels

At the outset of both examples, we rationalized the service-level structure. By standardizing a small number of service-level offerings, we greatly simplified the analysis and created a simplified process of allocating resources for applications. The applications that had the most demanding service levels could be deployed in about the same time as any other application. The reduced cost of disaster recovery effectively reduced the service-level insurance premium—shifting the cost/benefit analysis—and made it practical for almost all applications.

Start by Standardizing Development and Test Environments

Development and test environments are readily virtualized, but getting maximum value usually depends on standardizing the processes.

Plan Data-Center Savings

Significant savings were realized from reviewing planned data-center investments. We also needed to consider the new data-center demands of the virtualized environment to optimize cost reductions. In both cases, there were significant cost avoidance opportunities. We still had to avoid creating hot spots or congestion points.

Consider Security
Security must be carefully considered. Virtualization technologies have significantly improved, with regard to security; however, we found that the extra time that we spent in ensuring a secure virtualization strategy was well-invested.

Plan for Operations
Virtualization will drive significant operational changes. New tools and processes are required for managing a virtualized environment. We planned for the new tools and processes that were required to manage the new environment and realize the planned benefits. We also think it is important to plan, so that operations will have the tools and processes that are required to measure the benefits of the new environment.

Operations

Most of the benefits of virtualization are achieved in improved operations. The corollary is that all of the benefits that have been targeted can be offset by weak operations. Organizations that had weak processes quickly needed to drive improvements to manage new environments. Organizations that had strong processes yielded new benefits. This applied to organizations that pursued either strategy. Fortunately, many vendors are now providing tools that are aware of virtualization.

Configuration and change management became more demanding. Before virtualization, the number of new servers and configuration changes was restricted by the pace of physical changes. The ease of creating new virtual servers could have led to virtual-server sprawl. We needed better change-management processes.

Incident management was more demanding in the virtualized environment. Isolation of a performance problem and root-cause analyses were more complex. Solid incident management also relied on the improved configuration information.

Capacity management was less of an effort after virtualization. The consolidation of storage simplified management. Although this created fewer potential points for capacity issues, it also meant that more devices and services depended on the consolidated storage. Capacity required more diligence and monitoring. Understanding the overall capacity demands on CPU, memory, storage, and network was more complex after virtualization.

It is our experience that one of the most challenging changes was around network management. It was very important to monitor network activity after virtualization. Aggressive consolidation can create network congestion that is more challenging to diagnose in a virtualized environment. These changes dramatically changed network-traffic patterns. Ongoing monitoring, tuning, and management are absolutely required to maintain the health of the solution. The emerging technologies around network virtualization promise improvements, but they will be even more demanding on monitoring and management.

What's Next?

We are very interested in the emerging technologies around virtualization of the network adapters. This could significantly influence the server marketplace and virtualization implementations. In the preceding examples, the network is still one of the most challenging components to plan, implement, and manage in a virtualized environment.

We are also seeing more virtualization to secure environments. Virtual-desktop technology readily lends itself to deploying secure standard environments. New, more efficient protocols could change potential deployment scenarios.

Cloud computing is essentially a virtualization technique that we expect to become a regular part of virtualization planning. We expect combinations of cloud computing and on-premises virtualization to become a common pattern.

Conclusion

Our experience, based on a number of organizations and virtualization scenarios, has led to the following conclusions:

- Most organizations have unexploited virtualization opportunities. Virtualization is applicable to a number of business settings and can drive a wide range of benefits.
- With more planning, most organizations would derive more benefits from virtualization. They do not perform the planning that is required to maximize the benefits that are realized.
- Most organizations do not provide the tools and processes for maximum operational benefits. We would not consider a strategy to be successful unless it demonstrated value over time. We have found that ongoing investment in the strategy depends on being able to measure the benefits that are identified in the planning phase.
- Finally, we believe that emerging technologies will support even larger benefits from virtualization. New applications and infrastructure products are increasingly virtualization-certified. Emerging network-virtualization products promise more efficiency and manageable server virtualization. Desktop virtualization promises significant efficiency improvements. Security concerns are being addressed on an ongoing basis. Our next project will combine new cloud-based virtualization and the most current on-premises virtualization technologies.

As an architectural technique, virtualization is already very powerful. All signs indicate that new approaches and technologies will provide even more powerful virtualization cases.

About the Author

Greg Hutch's experience with virtualization spans a number of organizations. Currently, he is Director, Enterprise Architecture for Viterra (an international agri-business), where his responsibilities include enterprise architecture and strategic IT planning. Previously, Greg was Manager, Technical Development, at SaskPower; and Director, Technology Solutions, at Information Services Corporation; and he has worked in a range of consulting roles. Greg can be reached at greg_hutch@hotmail.com.

From Virtualization to Dynamic IT

by David Ziembicki

Summary

Virtualization is a critical infrastructure-architecture layer that is required for achieving higher IT-maturity levels, but several others layers—such as automation, management, and orchestration—are equally important.

Introduction

Dynamic IT is an advanced state of IT maturity that has a high degree of automation, integrated-service management, and efficient use of resources. Several IT-maturity models, including Gartner's Infrastructure Maturity Model and Microsoft's Infrastructure Optimization Model, define maturity levels and the attributes that are required to achieve each level.

Both the Gartner and Microsoft models agree that virtualization is a critical architecture component for higher IT-maturity levels. However, many other components are required. An infrastructure that is 100 percent virtualized might still have no process automation; it might not provide management and monitoring of applications that are running inside virtual machines (VMs) or IT services that are provided by a collection of VMs. In addition to virtualization, several other infrastructure-architecture layers are required to reach the highest levels of IT maturity.

A rich *automation layer* is required. The automation layer must be enabled across all hardware components—including server, storage, and networking devices—as well as all software layers, such as operating systems, services, and applications. The Windows Management Framework—which comprises Windows Management Instrumentation (WMI), Web Services-Management (WS-Management), and Windows PowerShell—is an example of a rich automation layer that was initially scoped to Microsoft products, but that is now being leveraged by a wide variety of hardware and software partners.

A *management layer* that leverages the automation layer and functions across physical, virtual, and application resources is another required layer for higher IT maturity. The management system must be able to deploy capacity, monitor health state, and automatically respond to issues or faults at any layer of the architecture.

Finally, an *orchestration layer* that manages all of the automation and management components must be implemented as the interface between the IT organization and the infrastructure. The orchestration layer provides the bridge between IT business logic, such as "deploy a new web-server VM when capacity reaches 85 percent," and the dozens of steps in an automated workflow that are required to actually implement such a change.

The integration of virtualization, automation, management, and orchestration layers provides the foundation for achieving the highest levels of IT maturity.

Architectural Considerations

Achieving higher IT maturity requires a well-designed infrastructure architecture that comprises the previously outlined layers, each of which might be provided by one or more products/solutions. Figure 1 illustrates this conceptual architecture.

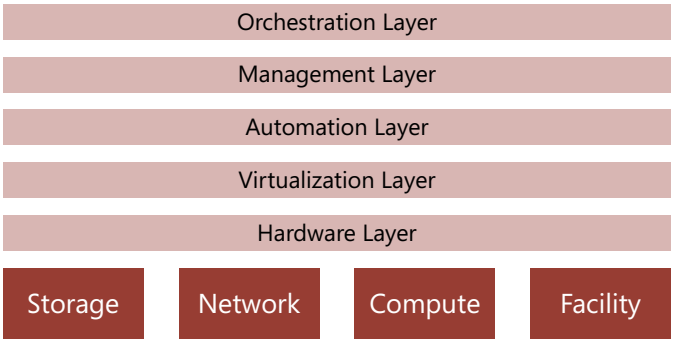
One of the most important choices to be made is which architecture layer or layers will provide resiliency and high availability for your IT services. These attributes can be provided by the infrastructure layers, at the application platform layer, or some combination of them.

It is well known that the architecture pattern that is used by most large cloud services is a scale-out model on commodity hardware. High availability and service resiliency are provided by the application platform and associated development model (stateless-server roles, automatic data replication, and so on). This model makes sense, given the nature of the workload—in particular, the small amount of standard workloads that the hardware must support in very large quantities, such as a standard data server and a standard web server.

For the more complex and heterogeneous workloads that are found in most infrastructures, this model might not be possible, particularly in the short term. Virtualization and consolidation of legacy workloads that do not include load balancing and other high-availability techniques at the application layer might require the virtualization or hardware layers to provide this capability.

It is critical to evaluate the costs of each of these approaches. Providing high availability in the software and application layer can enable a significant reduction in hardware costs by enabling commodity hardware to be utilized and reducing the need for expensive, fault-tolerant hardware. One goal of an enterprise architecture strategy should be to move your application inventory over time (via redesign, retirement, and so on) to the least costly infrastructure model. This is one of the key business drivers of cloud computing.

Figure 1: Infrastructure-architecture layers



To illustrate how this conceptual architecture enables a high degree of IT maturity and automation, we will consider a virtualized, three-tier application that consists of a clustered pair of database servers, two application servers, and three web servers. For each layer of the conceptual infrastructure architecture, the requirements and capabilities that it must provide to support the workload will be illustrated.

Hardware Layer

The hardware-architecture choices that are available to data-center architects are constantly evolving. Choices range from commodity rack-mounted servers to tightly integrated, highly redundant blade systems to container models. The same spectrum exists for storage and networking equipment.

Hardware and software integration is an attribute of higher IT maturity. Server, storage, and networking hardware should be selected based on its ability to be managed, monitored, and controlled by the architecture layers that are above it. An example would be a server that is able to report issues—such as high system temperatures or failed power supplies—to higher-level management systems that can take proactive measures to prevent service disruptions. Ideally, there would not be separate management systems for each hardware type; each component should integrate into a higher-level management system.

Careful consideration of the virtualization and application platforms should be part of the process of hardware selection. If the application platform will provide high availability and resiliency, commodity hardware and limited redundancy can be acceptable and more cost-effective. If the application platform cannot provide these capabilities, more robust hardware and high-availability features in the virtualization layer will be required.

For our reference three-tier application, the hardware layer provides the physical infrastructure on which the application's virtual servers run. The hardware layer must provide detailed reporting on the condition and performance of each component. If a disk or host bus adapter (HBA) starts to exhibit a number of errors that is higher than average, this must be reported to the management system, so that proactive measures can be taken prior to an actual fault.

Virtualization Layer

The virtualization layer is one of the primary enablers of greater IT maturity. The decoupling of hardware, operating systems, data, applications, and user state opens a wide range of options for better management and distribution of workloads across the physical infrastructure. The ability of the virtualization layer to migrate running VMs from one server to another with zero downtime, the ability to manage memory usage dynamically, and many other features that are provided by hypervisor-based virtualization technologies provide a rich set of capabilities. These capabilities can be utilized by the automation, management, and orchestration layers to maintain desired states (such as load distribution) or to proactively address decaying hardware or other issues that would otherwise cause faults or service disruptions. The virtualization layer can include storage, network, OS, and application virtualization.

As with the hardware layer, the virtualization layer must be able to be managed by the automation, management, and orchestration layers. The abstraction of software from hardware that virtualization provides moves the majority of management and automation into the software space, instead of requiring people to perform manual operations on physical hardware.

In our reference three-tier application, the database, application, and web servers are all VMs. The virtualization layer provides a portion of the high-availability solution for the application by enabling the ability to migrate the VMs to different physical servers for both planned and unplanned downtimes. The virtualization layer must expose performance and management interfaces to the automation layer, so that the management and orchestration layers can leverage these to automate processes such as patching the physical servers, without affecting application availability. In more advanced scenarios, the virtualization layer can span facilities to provide a site-resilient architecture.

Automation Layer

The ability to automate all expected operations over the lifetime of a hardware or software component is critical. Without this capability being embedded in a deep way across all layers of the infrastructure, dynamic processes will grind to a halt as soon as user intervention or other manual processing is required.

Windows PowerShell and several other foundational technologies, including WMI and WS-Management, provide a robust automation layer across nearly all of Microsoft's products, as well as a variety of non-Microsoft hardware and software. This evolution provides a single automation framework and scripting language to be used across the entire infrastructure.

The automation layer is made up of the foundational automation technology plus a series of single-purpose commands and scripts that perform operations such as starting or stopping a VM, rebooting a server, or applying a software update. These atomic units of automation are combined and executed by higher-level management systems. The modularity of this layered approach dramatically simplifies development, debugging, and maintenance.

Management Layer

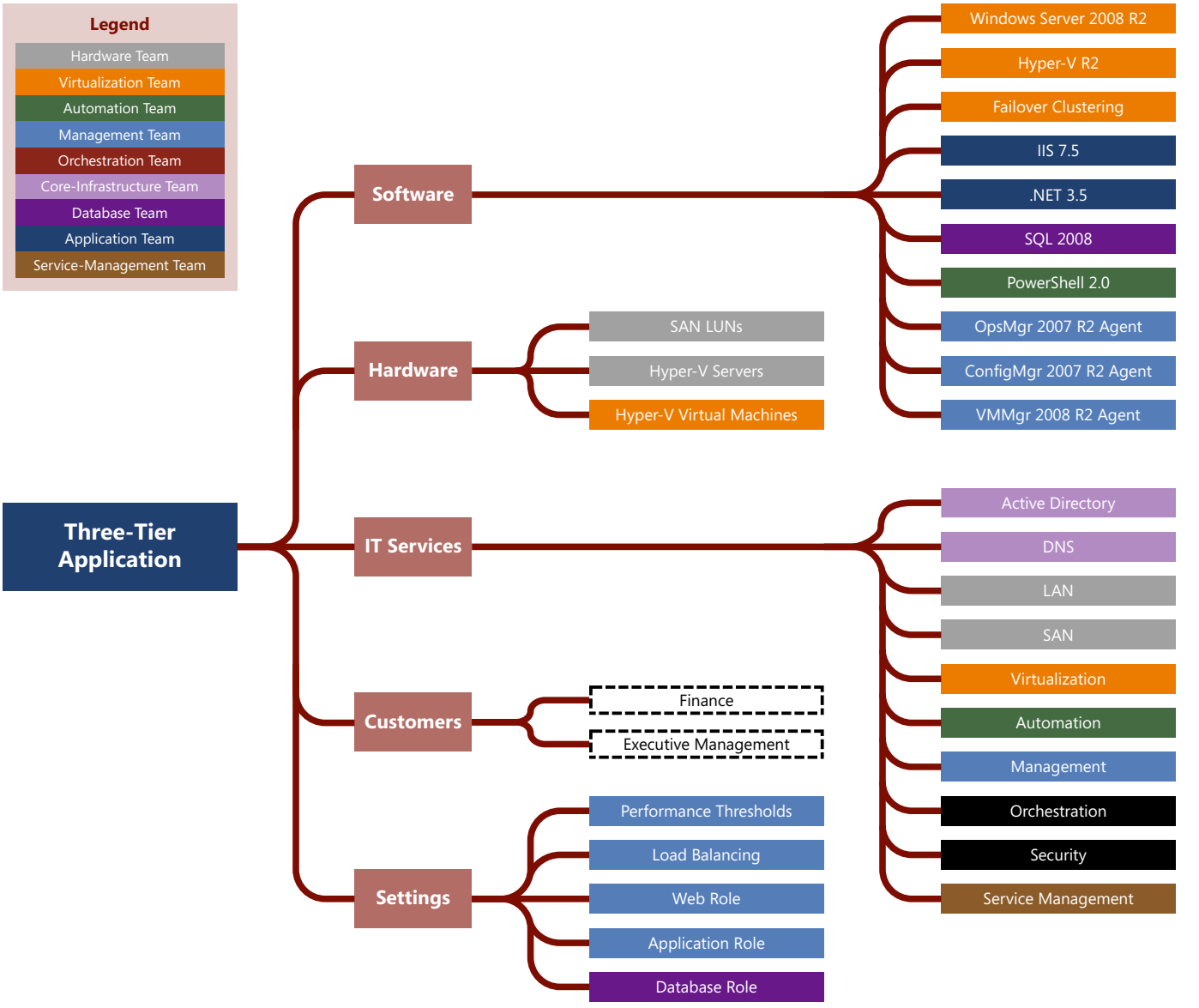
The management layer consists of the tools and systems that are utilized to deploy and operate the infrastructure. In most cases, this consists of a variety of different toolsets for managing hardware, software, and applications. Ideally, all components of the management system would leverage the automation layer and not introduce their own protocols, scripting languages, or other technologies (which would increase complexity and require additional staff expertise).

The management layer is utilized to perform activities such as provisioning the storage-area network (SAN), deploying an operating system, or monitoring an application. A key attribute is its abilities to manage and monitor every single component of the infrastructure remotely and to capture the dependencies among all of the infrastructure components.

One method for capturing this data is a *service map* that contains all of the components that define a service such as the three-tier application that we have been discussing. The Microsoft Operations Framework (MOF) defines a simple but powerful structure for service mapping that focuses on the software, hardware, dependent services, customers, and settings that define an IT service and the various teams that are responsible for each component. In this case, the application consists of the database, application, and web-server VMs, the physical servers on which they run, dependent services such as Active Directory and DNS, the network devices that connect the servers, the LAN and WAN connections, and more. Failure of any of these components causes a service interruption. The management layer must be able to understand these dependencies and react to faults in any of them. Figure 2 on page 26 represents a service map for the three-tier application.

The management layer must be able to model the entire application or service and all of its dependencies, and manage and monitor them. In System Center Operations Manager, this is referred to as a *distributed application*. This mapping of elements and dependencies enables event correlation, failure root-cause analysis, and business-impact analysis—all of which are critical toward identifying issues before they cause service interruption; helping to restore service rapidly, if there is an interruption; and assisting in resource prioritization when multiple issues are encountered.

Figure 2: Service-map diagram



Orchestration Layer

The orchestration layer leverages the management and automation layers. In much the same way that an enterprise resource planning (ERP) system manages a business process such as order fulfillment and handles exceptions such as inventory shortages, the orchestration layer provides an engine for IT-process automation and workflow. The orchestration layer is the critical interface between the IT organization and its infrastructure. It is the layer at which intent is transformed into workflow and automation.

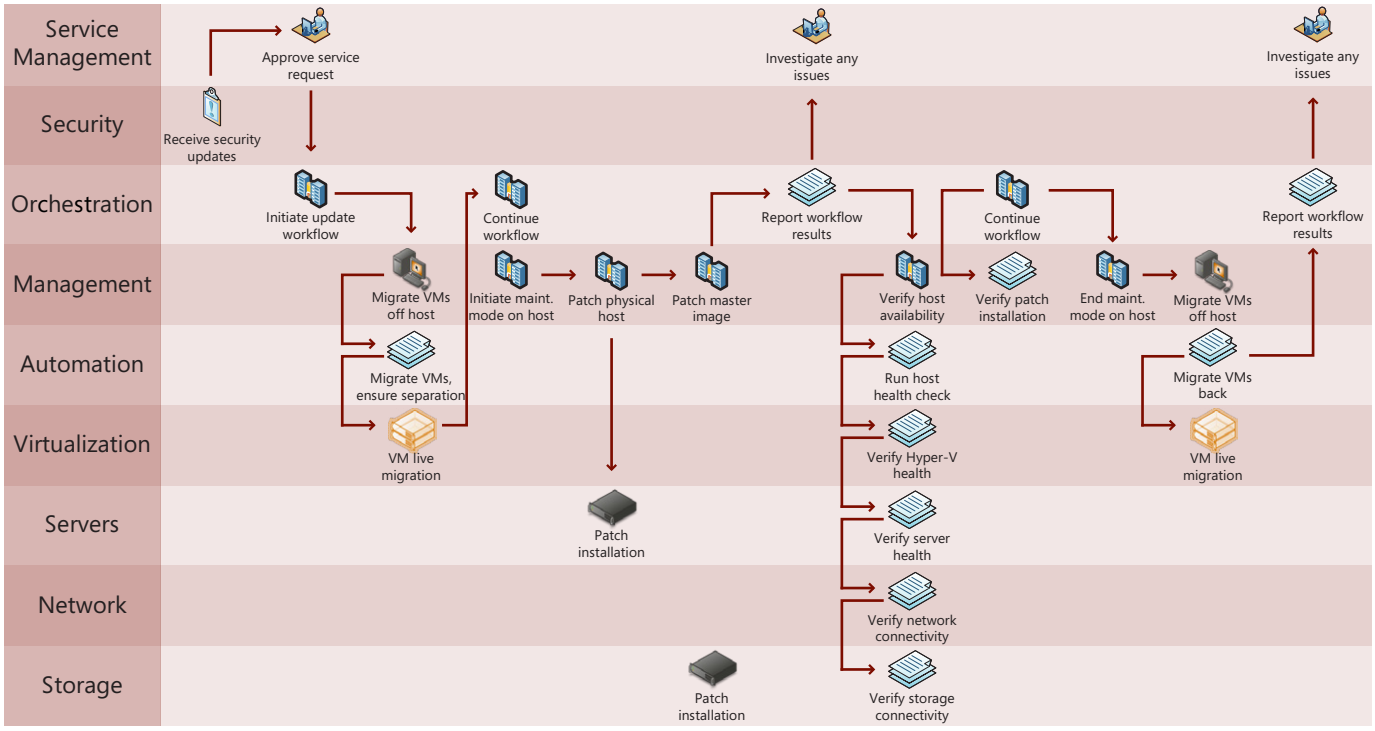
Ideally, the orchestration layer provides a graphical interface in which complex workflows that consist of events and activities across multiple management-system components can be combined, so as to form an end-to-end IT business process such as automated patch management or automatic power management. The orchestration layer must provide the ability to design, test, implement, and monitor these IT workflows.

Consider how many activities must take place in order to update a physical server in a cluster that is running a number of VMs. An applicable patch must be identified; a maintenance window for applying the patch must be defined; the VMs that are hosted by the server must be migrated to a different host; the patch must be applied, and then tested; and, finally, VMs must be moved back to the host. What happens if any one of these operations fails? An orchestration layer is required to enable complex IT processes and workflows, such as those that were previously described.

To be able to orchestrate and automate an IT process, the inputs, actions, and outputs of the process must be well understood. Also required is the ability to monitor the running workflow and configure recovery actions, should any step in the process fail. The notifications that are required during success or failure must also be well understood. An effective method for documenting an IT process is to use an IT-process map. Figure 3 on page 27 illustrates an example that is based on the patch-deployment scenario that was previously described.

In the three tier–application scenario, a process such as the one that was previously defined is required. During maintenance of the physical machines that host the VMs, the orchestration layer must

Figure 3: IT process–map diagram



account for the high-availability requirements of the application. The system must ensure that the clustered database-server VMs are not placed on a single physical host, as this would negate the benefit of the cluster by creating a single point of failure. The orchestration layer is where IT business-process rules and workflows such as this are defined. These should be treated as source code and placed under strict change control. This layer provides an excellent platform for continuous improvement by encoding the lessons that are learned into IT-process automation.

Achieving Dynamic IT

One of the key drivers of the layered approach to infrastructure architecture that is presented here is to enable complex workflow and automation to be developed over time by creating a collection of simple automation tasks, assembling them into procedures that are managed by the management layer, and then creating workflows and process automation that are controlled by the orchestration layer. Atop-down approach to this type of automation is not recommended, as it often results in large, monolithic, and very complicated scripts that are specific to individual environments. These become very costly to maintain over time. A better approach is to assemble a library of automation tasks, script modules, and workflows that can be combined in different ways and reused. Start by recording how often each IT task is performed in a month and automate the top 10 percent. Over time, this automation library will grow into a significant asset.

Achieving dynamic IT requires deep understanding and documentation of your existing IT processes. Each process should be evaluated prior to attempting to automate it. Service maps and IT-process maps are essential tools for gathering and documenting the information that is required to automate, manage, and orchestrate IT processes. All redundant or nonessential steps should be removed from the process prior to automating it. This both streamlines the process itself and reduces the amount of automation effort that is required.

Conclusion

Virtualization is an important tool for increasing IT maturity. To augment the benefits of virtualization, capabilities such as automation, management, and orchestration are required. The infrastructure architecture must be combined with a trained staff and streamlined IT processes. It is the combination of efficient processes, a well-trained staff, and a robust infrastructure architecture that enables dynamic IT.

Resources

Microsoft Operations Framework Team. "Service Mapping: Microsoft Operations Framework (MOF), Version 1.0." Online document. March 2010. Microsoft Corporation, Redmond, WA. (http://go.microsoft.com/fwlink/?LinkId=186459)

Microsoft TechNet Library. "Infrastructure Optimization." Online article series. 2010. Microsoft Corporation, Redmond, WA. (http://technet.microsoft.com/en-us/library/bb944804.aspx)

About the Author

David Ziembicki (davidzi@microsoft.com) is a Solution Architect in Microsoft's Public Sector Services CTO organization, focusing on virtualization, dynamic data centers, and cloud computing. A Microsoft Certified Architect | Infrastructure, David has been with Microsoft for four years leading infrastructure projects at multiple government agencies. David is a lead architect for Microsoft's virtualization service offerings. He has been a speaker at multiple Microsoft events and served as an instructor for several virtualization-related training classes.

You can visit David's blog at http://blogs.technet.com/davidzi.

Platform
Architecture Team

Microsoft

24

subscribe at
www.architecturejournal.net