The Architecture Journal

input for better outcomes

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

19

TechEd Special Edition

Microsoft

Contents

Founder
Arvindra Sehmi

Director
Lucinda Rowley

Editor-in-Chief
Diego Dagum

Contributors for This Issue
Darryl Chantry
Kendrick Efta
Nick Malik
J.D. Meier
Ranjan Sen, Ph.D.
Mike Walker

Design, Print, and Distribution
United Business Media Limited – Contract Publishing
Chris Harding, Managing Director
Angela Duarte, Publication Manager
Bob Steigleider, Production Manager
Camille Verde, Senior Art Director

Dear Architect,

We have been actively working on enhancing your experience both in terms of delivery channel and high quality content. To do so, we have decided to stop our print edition and become a fully digital resource. With the dynamic and flexible digital format, you can search inside articles, send by email, and access related information and multimedia with a simple mouse-click. And, of course, you can still print the whole magazine or an individual article if you want to.

Let's start from delivery channels. Some readers were already receiving the *Journal* as a quarterly newsletter containing a link to the PDF. Those subscribed to the printed edition are now receiving this newsletter too, sent to the email account used when you originally subscribed.[1] This newsletter is now enriched with a selection of relevant architecture articles, featuring content published by Microsoft as well as other media groups and independent voices. The newsletter frequency becomes monthly (the *Journal* will keep its quarterly cycle). Content won't be limited to articles: You'll also get taped interviews, demos, case studies, gadgets, calls for papers, and a wealth of other resources to help you in your work as an architect, aligning technology to the business.

The editorial coverage is also being re-aligned and enhanced. We'll continue to cover current and forward-thinking topics of interest to the industry (we are working these days on "*Architecture during uncertain times: Giving value back to the business thru smart technology decisions*"), looking at issues from development to infrastructure from the architect's eye. Our coverage of concrete platform solutions will be extended to new depths. You'll find useful information about putting ideas into practice in both the featured articles themselves and the complementary readings through our partnership with *MSDN Magazine* for software developers and *Technet Magazine* for IT professionals.

Dear reader, while discontinuing the print magazine represents a disruption, we are excited about the improvements to relevance, accuracy, and quantity of information we will be able to deliver to your inbox.

[illegible]

Diego Dagum
Editor-in-chief

[1] You don't lose your right to unsubscribe to this newsletter.
Find how at http://msdn.microsoft.com/en-us/architecture/aa699369.aspx.

Mapping Applications to the Cloud

by Darryl Chantry

Summary

As economic pressure builds globally, many organizations are starting to look at Cloud Computing as a potential choice to reduce the total cost of ownership for IT. In searching for ways to use Cloud Computing technologies, enterprises have to ask what applications make good candidates for moving to the Cloud and which do not, such as "Does the nature of the business itself allow for Cloud Computing to even be considered?"

This article provides a broad overview into what Cloud Computing is and discusses an approach to mapping enterprise applications to Cloud Computing platforms to assist in determining whether your applications or your business model are a good fit for the Cloud.

"Will all applications run in the Cloud? Should you attempt to port all of your existing applications to the Cloud? Should all your new applications be developed in the Cloud? What is this Cloud thing, anyway?"

Which Came First: The Cloud or Cloud Computing?

Cloud computing has fired the imaginations of many information-technology (IT) professionals around the world, whether they are small independent software vendors (ISVs), Silicon Valley startups, or large corporations that are looking to cut costs. There seems to be an ever-increasing number of people who look to the Cloud to hit upon the magic bullet that will solve any IT problem.

One interesting aspect of the hype that surrounds cloud computing is the lack of a clear definition as to what cloud computing is and, just as relevant, what it is not. If you were to ask 100 people to define the Cloud and what they believe cloud computing is, you would probably get 150 different answers (some people tend to answer twice, with the first answer contradicting the second). With this in mind, it seems only fitting to begin this article by discussing a general definition for cloud computing.

The Cloud (or the Internet, if you prefer) has been around for some time now, about 25 years; so without a doubt, the Cloud came first, right? Well, one could argue that the first servers on the Internet were really storage devices for data and applications to be shared and run globally; or to put it another way, to provide cloud computing resources in multiple locations globally with almost infinite scalability. Contrast that to today's cloud-computing initiatives that are pretty much there to provide data, applications, and computing power globally with almost infinite scalability, and you quickly see the difference… Or is there really a difference?

The difference is that we are using new technologies to put a new spin on old ideas. Cloud computing is more about evolution than revolution, with technology allowing price points that take these thoughts and make them available to all people — regardless of budget size — via a utility-based, pay-for-what-you-use model.

Utility Computing

Utility computing refers to using computing resources (infrastructure, storage, core services) in the same way you would use electricity or water; that is, as a metered service in which you only pay for what you use. The utility can eliminate the need to purchase, run, and maintain hardware, server, and application platforms, and to develop core services — for example, billing or security services. Consider the following scenario.

A Web-based ISV that wants to makes components available for Facebook or MySpace faces the following dilemma: The components they create could be adopted by thousands, or could struggle to find acceptance in any form. Most ISVs have limited capital, so they need to balance the expenditure between developing their application and providing infrastructure to support their software.

Such balancing acts can lead to poor applications with good platform support, or great applications that are rarely accessible due to poor platform support. Neither scenario is a path to success; this is where utility-based cloud platforms can help. Cloud utility platforms can provide a low-cost alternative that can easily scale to meet the demand for the ISV's application, which allows them to commit practically all of their resources toward building a great application.

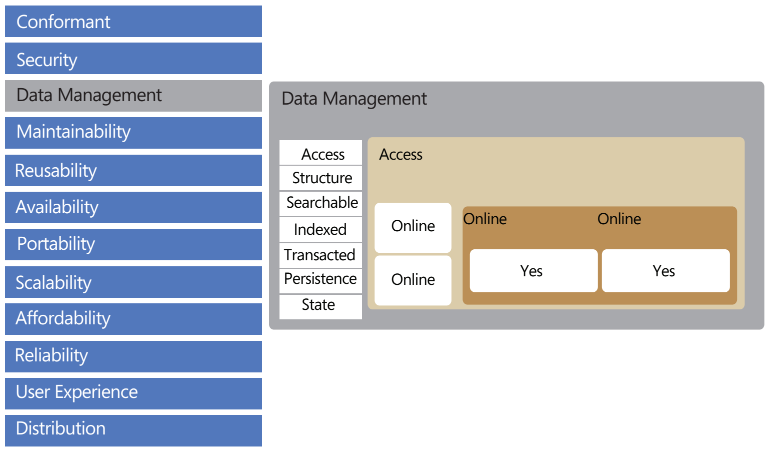
As cloud services are essentially available as a utility offering, should the product fail, the ISV can simply shut down the services and stop all costs associated with the software.

The utility model also allows organizations to offset some of the costs of running private data centers by providing additional infrastructure resources to manage peak loads; this is also known as *cloud bursting*.

Traditionally, to handle peak loading, organizations would often design data centers that had the processing power to manage peak loads; this meant that for the majority of the time the data center was underutilized. By using cloud bursting, an organization can build a data center to the specifications that will allow the entity to run all normal day-to-day workloads within their environment, and then use cloud providers to provide additional resources to manage peak loads.

Utility computing is often associated to some form of virtualized platform that allows for an almost infinite amount of storage and/or computing power to be made available to the platforms users through larger data centers. The evolution of cloud computing is now broadening the definition of utility computing to include service beyond those of pure infrastructure.

Figure 1: Attributes map of an application



"Every application is designed to fulfill some purpose, whether an flight reservation system, CRM application, or something else. To implement the function of the application, certain attributes need to be present, which could mean cloud storage might not be suitable as a data store. Determining the key attributes of any application is a key step in determining whether an application is suitable for the Cloud."

Will All Applications Move to the Cloud?

Will all applications run in the Cloud? Should you attempt to port all of your existing applications to the Cloud? Should all your new applications be developed in the Cloud? What is this Cloud thing, anyway? These are a few of the questions that arise whenever you start thinking about using cloud services.

Some applications will be ideal candidates to be ported to a cloud platform, developed on a cloud platform, or hosted on a cloud infrastructure, while other applications will be poor cloud candidates. In this case, the standard architectural answer "it depends" can be applied to all of the preceding questions. Practically every application potentially could exist either partially or fully in the Cloud; the only caveat to this are the trade-offs in an application's attributes — and, possibly, functionality — that you might be willing to make to move it to the Cloud.

The following pages discuss a few ideas for decomposing an application into its basic attributes, and decomposing the Cloud into its basic attributes, to help make decisions as to whether running your specific application in the Cloud is practical.

Mapping an Application to the Cloud

Every application is designed to fulfill some purpose, whether an order management system, flight reservation system, CRM application, or something else. To implement the function of the application, certain attributes need to be present: For example, with an order management system, transaction and locking support might be critical to the application. This would mean that cloud storage might not be suitable as a data store for such a purpose. Determining the key attributes of any application or subsystem of a larger application is a key step in determining whether or not an application is suitable for the Cloud.

Figure 1 shows a number of key high-level attributes (blue column) that could be relevant to any application. The potential number of attributes for any given application does not need to be recorded; what you are attempting to determine is which attributes are the critical ones for your application. This will likely produce a manageable list of attributes that can then be mapped to the Cloud. Selecting Data Management, for example, presents a list of secondary attributes that provide more details for the high-level attributes. Selecting Access then allows you to specify if you want either online, offline, or both online and offline access to your data source.

Building on the Data Access example, you can start to see how this attribute could affect the choice as to whether or not to use a cloud provider for data storage. Should the application in question need purely online data, cloud storage would be an excellent choice; however, if offline data is all that is required, this could be a key indicator that the application is not suited for the Cloud. And if you decide that the application requires both an online and an offline model, the cost of developing the application to synchronize data between the application and the Cloud would need to be considered.

Choosing to support both an offline and online experience for the end users will add additional cost to the project; however, should another attribute, such as high scalability, be identified, the advantages that the Cloud provides in this area easily could offset the cost of developing an offline experience. (See Appendix A, Sample Application-Mapping Attributes, page 7.)

What Makes Up the Cloud?

After you have decomposed an application and determined its key attributes, you can begin work on a similar exercise for the Cloud — specifically, for cloud service providers. Splitting cloud attributes into broad categories can simplify the mapping process. The categories used in this example are cloud infrastructure, cloud storage, cloud platform, cloud applications, and core cloud services.

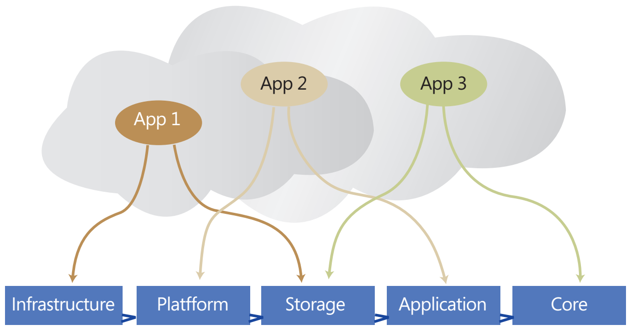
You could map any application attributes to cloud attributes in one or more categories, as depicted in Figure 2.

Cloud Infrastructure

Cloud infrastructure is infrastructure, or more commonly, virtual servers in the Cloud. Infrastructure offerings are the horsepower behind large-scale processes or applications. For large-scale applications, think Facebook or MySpace; for large-scale processing, think a high-performance infrastructure cluster that is running engineering stress-test simulations for aircraft or automobile manufacturing.

The primary vehicle for cloud infrastructure is virtualization; more specifically, running virtual servers in large data centers, thereby removing the need to buy and maintain expensive hardware, and taking advantages

Figure 2: Mapping application attributes to cloud attributes



of economies of scale by sharing Infrastructure resources. Virtualization platforms are typically either full virtualization or para-virtualization environments. (See Appendix B for a more detailed explanation of virtualization, page 8.)

Cloud Storage
Cloud storage refers to any type of data storage that resides in the Cloud, including: services that provide database-like functionality; unstructured data services (file storage of digital media, for example); data synchronization services; or Network Attached Storage (NAS) services. Data services are often consumed in a pay-as-you-go model or, in this case, a pay-per-GB model (including both stored and transferred data).

Cloud storage offers a number of benefits, such as the ability to store and retrieve large amounts of data in any location at any time. Data storage services are fast, inexpensive, and almost infinitely scalable; however, reliability can be an issue, as even the best services do sometimes fail. Transaction support is also an issue with cloud-based storage systems, a significant problem that needs to be addressed for storage services to be widely used in the enterprise.

"A *cloud platform* is really the ability to build, test, deploy, run, and manage applications in the Cloud. Cloud platforms offer alternatives to these actions; for example, the build experience might be online only, offline only, or a combination of the two, while tools for testing applications might be nonexistent on some platforms, yet superb on others."

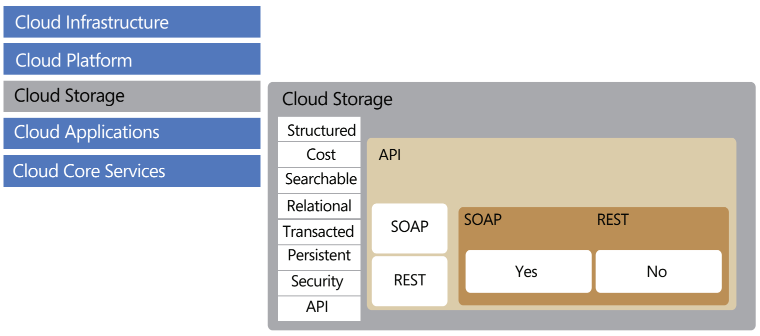
Cloud Platform
A *cloud platform* is really the ability to build, test, deploy, run, and manage applications in the Cloud. Cloud platforms offer alternatives to these actions; for example, the build experience might be online only, offline only, or a combination of the two, while tools for testing applications might be nonexistent on some platforms, yet superb on others.

Cloud platforms as a general rule are low-cost, highly scalable hosting/development environments for Web-based applications and services. It is feasible (although an oversimplification) to consider cloud platforms as an advanced form of Web hosting, with more scalability and availability than the average Web host. There are pros and cons for any technology, and a con in the cloud platform world is portability. As soon as an application is developed to run on a specific platform, moving it to another cloud platform or back to a traditional hosting environment is not really an option.

Cloud Applications
A *cloud application* exists either partially or fully within the Cloud, and uses cloud services to implement core features within the application. The architecture of cloud applications can differ significantly from traditional application models and, as such, implementing cloud applications can require a fundamental shift in application-design thought processes.

Cloud applications can often eliminate the need to install and run the application locally, thereby reducing the expenditure required for software maintenance, deployment, management, and support. This type of application would be considered a Software as a Service (SaaS) application.

Figure 3: Five cloud categories and attributes for cloud storage



An alternative to this would be the Software plus Services (S+S) model. This is the hybrid between traditional application development and a full SaaS implementation. S+S applications typically use rich client applications that are installed on a client's PC as an interface into externally hosted services. S+S often includes the ability to interact with an application in an offline mode, and sync back to a central service when required.

Core Cloud Services
Core cloud services are services that support cloud-based solutions, such as identity management, service-to-service integration, mapping, billing/payment systems, search, messaging, business process management, workflow, and so on. Core cloud services can be consumed directly by an individual, or indirectly through system-to-system integration.

The evolution of core cloud services potentially will mimic that of the telecommunications industry, with many services falling under the categories of Business Support Systems (BSS) or Operational Support Systems (OSS).

BSS services manage the interactions, with customers typically handling tasks such as:

- Taking orders
- Processing bills
- Collecting payments.

OSS services manage the service itself and are responsible for items such as:

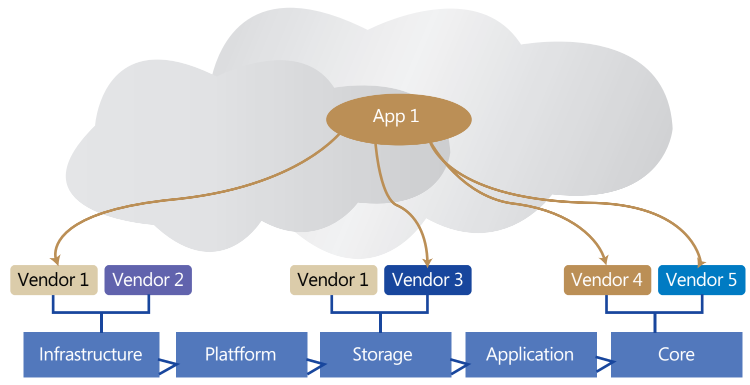
- Service monitoring
- Service provisioning
- Service configuration.

Attributes Map for Cloud Services
By using the five cloud categories, we can now develop a set of attributes for each of the categories. These attributes can be used in two ways:

- Mapping your application's attributes to cloud attributes to validate whether cloud services are suitable for your application, and identifying which types of services to use
- Evaluating cloud service providers as possible candidates for hosting your applications, identifying which types of services are available from your chosen provider(s), and then determining specific implementation attributes of the services offered.

Figure 3 shows the five cloud categories and a list of attributes for the cloud-storage category. Each cloud provider implements its cloud services

Figure 4: Single application that uses multiple cloud services and vendors



in a slightly different way, with companies like Microsoft offering a number of different storage alternatives that developers can choose to use, depending on the required features, for a specific application.

Just like application attributes, cloud attributes must be weighed carefully when determining whether a cloud provider's services are a good fit for your needs, as you will have to factor in implementation cost for each decision you make. (See Appendix A, Sample Cloud-Mapping Attributes, page 8.)

Overlaying the Cloud and Applications

Now that you have a complete understanding of the application and of what cloud services you could use to implement a solution, you can now start to make decisions about what the eventual architecture could be. Should you find that the Cloud is a viable and cost-effective alternative to traditional application architecture, the next step would be to choose the most suitable cloud provider(s) for the application.

It is quite possible that no single vendor would match completely with your requirements; however, you might find that you can obtain all of the services that your application requires from multiple vendors.

Figure 4 depicts an application that uses a number of cloud services, from multiple cloud providers. The preceding example could represent an application that is built in ASP.NET and is running on the Azure platform (cloud platform); however, the application also needs components with full trust, which means that the components can run only in a full virtual environment (cloud infrastructure). Data is stored in a Microsoft cloud (cloud storage), with services such as Workflow and Identity (core cloud services) also provided via Azure. The last requirement for the application could be a billing/payment service (core cloud services), which could be provided by another cloud provider.

Although this scenario is feasible, the costs that are associated with having accounts with multiple providers, using a number of APIs, and then integrating all of the services into an application could be impractical. The likely solution would be to find a single vendor that delivers the majority of the services that are required by your application, and use this as the base platform for a hybrid solution.

One Cloud to Rule Them All

Is there one cloud, or are there multiples clouds? This is a debate that I have heard a number of times already. One side of this argument is public versus private clouds. Can a private implementation of cloud technologies be called a cloud at all, or is it something else? Are all of the public cloud offerings the same? And what about applications or systems that span both private and public clouds in a hybrid model? Where do these fit?

The honest answer is that the argument is irrelevant. Whether you subscribe to one theory or another, the desired outcome is the same: Build the most cost-effective system that you can that works. The previous section looked at ideas to help you make decisions; now, we will take a quick look at potential applications that could exist in the Cloud or as part of a hybrid cloud solution.

Architecting Solutions in the Cloud

This section describes three application scenarios for which solutions could be implemented by using cloud services. The following scenarios are by no means an extensive list of possible solutions that are suitable for cloud services; they are only indications of applications that could be feasible.

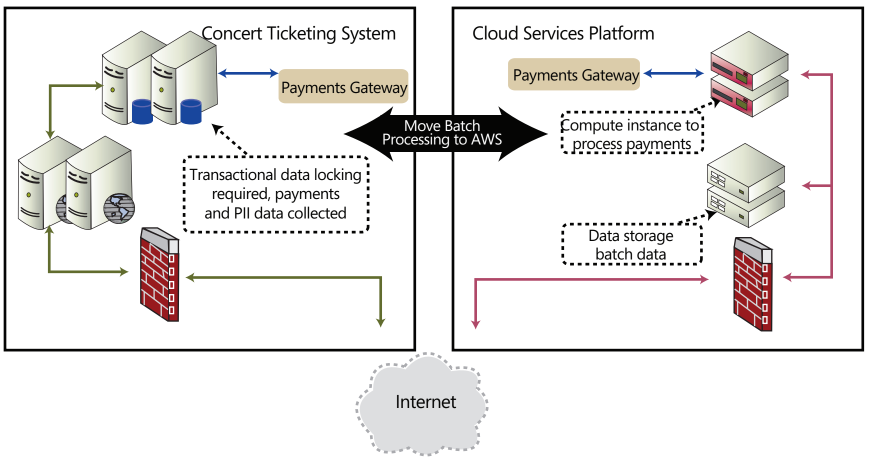
Ticketing System

When discussing the benefits of a cloud infrastructure, there seems to be a consensus that a concert ticketing system would make an ideal candidate for a cloud scenario (Figure 5). On the surface, this type of application looks like a viable candidate; ticketing systems are often subject to high demand over a short period of time (people scrambling to buy tickets for a concert or sporting event that will sell out rapidly). This is often followed by long periods of low to moderate activity.

Ticketing applications are often overloaded during the periods of high demand, when the need for computing resources is extremely high. The ability to run up instances of virtual machines to cover such periods would be beneficial. There are, however, a number of issues that must be taken into account before architecting such a solution:

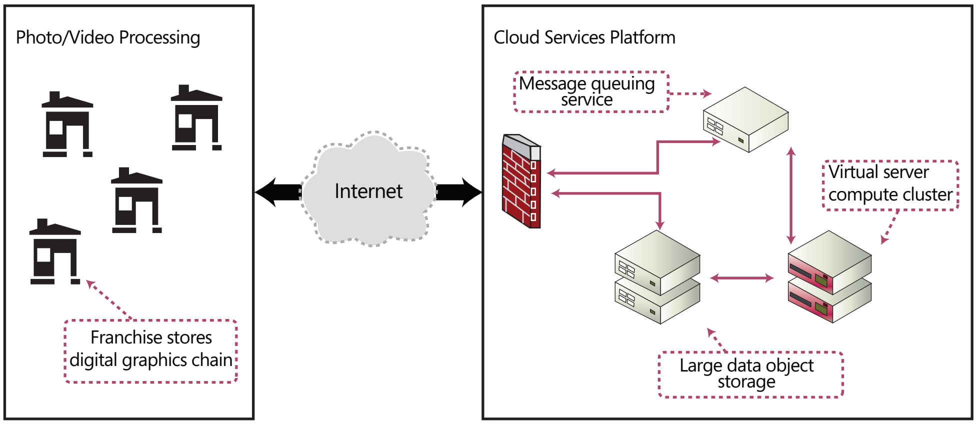
- Ticketing systems are data intensive and highly transactional. Transactions can be required for the payments system as well as to reserve specific seats for a given event.
- Personally identifying information is almost certain to be collected, with many customers having an account with the ticketing company or, potentially, wanting to create an account with the organization.
- Validating credit-card payments can be time consuming, and is a

Figure 5: Using cloud infrastructure for ticketing system


Concert Ticketing System
Payments Gateway
Transactional data locking required, payments and PII data collected
Move Batch Processing to AWS
Cloud Services Platform
Payments Gateway
Compute instance to process payments
Data storage batch data
Internet

potential source of bottlenecks in the ticketing process.
- Some virtual cloud-server platforms cannot save state, which means that once a server image is shut down, all changes or additions to data that is stored within the image are lost.
- Existing ticketing companies already will have significant investment in infrastructure and data management.
- Depending on the cloud service that is chosen, virtual cloud-server images that are not able to save state will need to be recreated for every event, which could well result in a significant amount of work being required to prepare an environment for each new instance.

Figure 6: Using cloud infrastructure for photo/video processing


Photo/Video Processing
Franchise stores digital graphics chain
Internet
Cloud Services Platform
Message queuing service
Virtual server compute cluster
Large data object storage

With all of this in mind, there are a number of ways to use a cloud service to reduce the demand on an existing system during a peak loading time:

- *Duplicate the internal system completely.* This would require the most amount of work; once the application is ready to use, it still would need to be synchronized with the current system for every new instance. Permanently leaving the system on (even in a reduced capacity) could be expensive due to the cost of using a services platform.
- *Split the workload between internal systems and a cloud service in real time.* This would involve splitting the process of selecting seats and purchasing tickets across the two environments; for example, the transaction could begin on the internal ticketing system where customers log into their account, select the event they wish to attend, select the seats, and then are passed off to a cloud service for final processing of payment. This would mean creating a virtual cloud server that simply completes the final stage of processing and, as such, would not need to be synchronized with the main system — effectively being a stateless processing engine. Only minimal data would need to be transferred to the cloud service, and credit-card information could be collected on the external system for single use and then deleted.
- *Split the workload between internal systems using batch processing.* Similarly to the preceding processing method, this method would differ in that all personal information would be collected on the internal system, including Cc: details. This information then could be placed in a process queue and shipped in batches to the cloud service for processing. This would mean that should payment fail, a secondary process for contacting the person who is attempting to purchase the tickets would need to be implemented if it does not already exist.

The preceding solutions are examples of how a ticketing system could split processing between a cloud service and a company's internal systems during periods of heavy use.

Photo/Video Processing
This example shows how you can combine multiple services (infrastructure, storage, and queuing) to provide a solution for data processing (Figure 6). In this scenario there are a chain of photo processing stores that make use of the cloud service to render or reformat digital media files.

The photo chain has a number of stores spread across the U.S. and wishes to centralize large image and video processing to reduce two aspects of the system: the amount of hardware in each store; and the complexity of maintaining and supporting the hardware.

When a customer comes into a store with a video that needs to be converted to a different format, the video file is first uploaded to a storage service, and then a message is placed in a queue service that a file is on the storage platform and needs to be converted to a different format. An application controller that is running computer instances receives the message from the queue, and then either uses an existing instance of a virtual machine, or creates a new instance, to handle the reformatting of the video. As soon as this process is complete, the controller places a message in the queue to notify the store that the project is complete.

The preceding scenario easily can be converted to an online experience, so that customers could upload files for processing without having to go to a physical location.

Web Site Peak Loading
The final example that I will use is that of a Web site that has an extremely high amount of traffic on an irregular basis, which makes it impractical to build out the hosting infrastructure to support such peaks (Figure 7). Such sites could be news sites with breaking stories, game sites announcing a new game, or movie sites showing trailers of the next blockbuster.

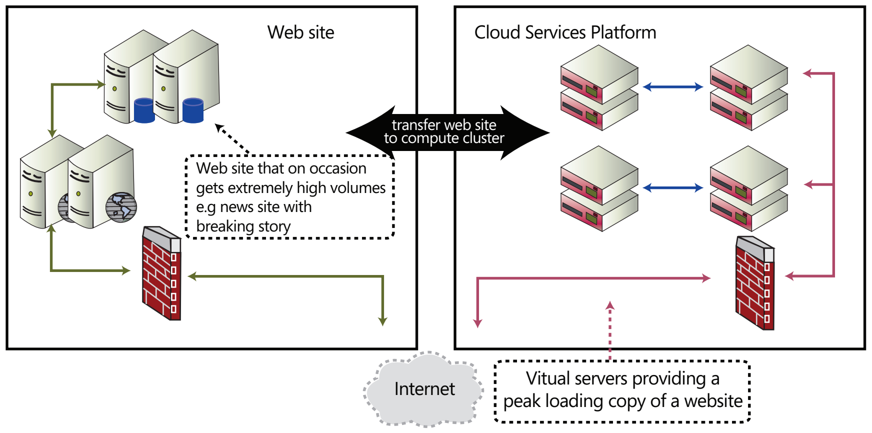
The solution to this scenario involves creating a complete copy of the company's Web site, or the part of the Web site that will experience the heavy traffic, on a cloud-service infrastructure service. The copy of the site would be a static instance running across a number of Web servers that could be configured as either a load-balanced set of servers or as a cluster. You can make any changes you need to on the original Web site, and then synchronize them back to the cloud servers. This would create latency, but would greatly reduce the effort needed to maintain the Web servers and Web sites, and would eliminate the problem of maintaining state between the internally and externally hosted Web sites.

There are many ways to architect solutions for the preceding scenarios, as there are many more scenarios in which you can use a cloud service. The goal of this article is merely to highlight a few of the alternatives and uses for the services that are emerging.

Conclusion
The fascination of the Cloud and cloud technologies is driving many developers, ISVs, start-ups, and enterprises to scrutinize cloud services

and assess their suitability for adoption. The promises of lower cost of ownership — and of almost limitless scalability, in both storage and infrastructure power — are hard to ignore. The promise of the Cloud definitely warrants inspection; however, you must manage the adoption of cloud services carefully and realize that not all applications are suited for the Cloud. Many applications will work in the Cloud; however, hidden costs of hosting some solutions in the Cloud could see projects being delivered with much higher development and running costs than would be true of more traditional and well-defined architectures and technologies.

Figure 7: Using cloud infrastructure for peak load coverage


Web site
Cloud Services Platform
transfer web site to compute cluster
Web site that on occasion gets extremely high volumes e.g news site with breaking story
Internet
Vitual servers providing a peak loading copy of a website

About the Author

Darryl Chantry is a senior architect in the Platform Architecture Team at Microsoft. He has a broad skill set as an architect, having worked as an enterprise architect, solutions architect, and infrastructure architect. He joined Microsoft's New Zealand Developer & Platform Evangelism team in 2002. Darryl was born and raised in Auckland, New Zealand. Rugby was always a passion for him, but he has developed a special interest in anthropology, history, and user experiences. He and his wife reside in Redmond, Wash., with a Boerboel dog and two Burmese cats.

Follow up on this topic

- Introducing the Azure Services Platform (a David Chappell article): http://download.microsoft.com/download/e/4/3/e43bb484-3b52-4fa8-a9f9-ec60a32954bc/Azure_Services_Platform.pdf
- Azure for Business: http://www.microsoft.com/azure/business.mspx
- Azure for Corporate Developers: http://www.microsoft.com/azure/corpdev.mspx
- Azure for Independent Software Vendors (ISVs): http://www.microsoft.com/azure/isv.mspx

Appendix A

Sample Application-Mapping Attributes

User experience

- Usability
- Responsive
- Efficiency
- Performance
- Personalizable
- User interface
- Graphical
- Interactive
- Distributed
- Textural
- None
- Interaction model
- Device
- SaaS
- Online

Security

- Emergency hotfix or breach management
- Security procedures
- Trust relationship with platform
- Applications security model
- Data flow
- Malicious code
- Access controls
- Remote access
- Identity
- Cryptography
- Auditing
- Authentication/Authorization model

Affordability

- Resource cost
- Development
- Available skills
- Software enhancements cost
- Licensing
- Postproduction hardware
- Decommissioning
- Initial hardware

Data

- State
- State full
- Stateless
- Stability
- Application constraints
- Database constraints
- Persistence
- Online/Offline
- Structure
- Unstructured
- Indexed
- Searchable
- Transaction management

Maintainability

- Available skill sets
- Language support (dev)
- Application standards
- Technology implementation
- Application-code complexity and volume
- Configuration management
- Operational management
- Flexible
- Technology

Scalability

- Replication
- Caching
- Pooling
- Software load balancing
- Scale out
- Scale up
- Hardware load balancing

Availability

- Technology/Configuration/ Implementation to support availability
- Uptime requirement

(cont'd on page 8)

Sample Application-Mapping Attributes - cont'd

Conformability
- Auditable
- Regulatory
- Standards

Portability
- Cross-platform
- Within platform

Distributability
- Local
- Geo-distributed

Extensibility
- Meta-model
- Configurable

Reliability
- Configuration management
- Startup and automatic recovery
- System performance
- Recovery procedures and methods
- Load balancing
- Fault tolerance

Interoperability
- Communications and data usage
- Integration impacts
- Architecture compatibility
- Ease integration (APIs)

Reusability
- Distributable and reusable
- Modularity
- Hierarchy
- Code abstraction

Sample Cloud-Mapping Attributes

Cloud infrastructure
- High availability
- Isolation level
- Support application types
- Support legacy applications
- Network support
- Platform (OS) support

Cloud platform
- Development environment
- Test environment
- Deployment model
- Language support

Core services
- Queue
- Identity
- Federated
- Claims
- Custom
- Billing
- Workflow
- Search

Cloud storage
- Structured
- Unstructured
- Highly scalable
- Stability
- Application constraints
- Database constraints
- Transaction management
- Indexed
- Searchable
- Online/Offline
- Persistence

Cloud application
- Messaging (e-mail)
- Customer relationship management (CRM)
- Project management
- Accounting
- Web portal
- Calendar
- Maps

Appendix B

Cloud-Infrastructure Platforms and Virtualization Types

One of the key enabling-technologies for cloud-computing platforms is virtualization, which is the ability to provide an abstraction of computing resources. When we look at cloud-infrastructure platforms as they stand today, they predominately come in two flavors: fully virtualized or para-virtualized environments.

There are many more variations to virtualization than the two that I have just mentioned; so, for this post, I thought that I would discuss some of the virtualization methods that exist and that could well find their way into a cloud-infrastructure offering.

Emulation

In this type of virtualization, the virtual environment emulates a hardware architecture that an unmodified guest operating system (OS) requires. One of the common instances in which you encounter emulated hardware is with mobile devices. Application developers will use an emulated environment to test applications that are designed to run on smart phones or on PDAs, for example. (See Figure 8.)

Pros: Simulates a hardware environment, which is completely different from the underlying hardware. An example of this would be a mobile device such as a smart phone emulated on a desktop PC.

Cons: Poor performance and high resource usage.

Full Virtualization

In full virtualization, an image of a complete unmodified guest OS is made and run within a virtualized environment. The difference between full virtualization and emulation is that all of the virtualized guests run on the same hardware architecture. All of the guests support the same hardware, which allows the guest to execute many instructions directly on the hardware—thereby, providing improved performance. (See Figure 9.)

Figure 8: Emulated-virtualization environment

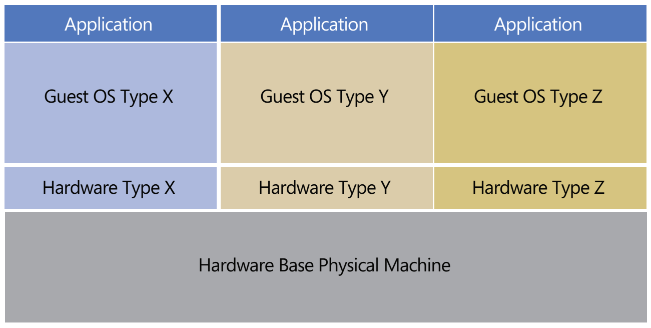
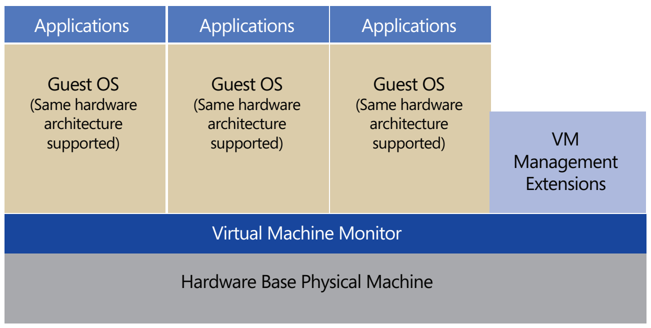


Figure 9: Full-virtualization environment


Applications
Applications
Applications
Guest OS (Same hardware architecture supported)
Guest OS (Same hardware architecture supported)
Guest OS (Same hardware architecture supported)
VM Management Extensions
Virtual Machine Monitor
Hardware Base Physical Machine

Pros: The ability to run multiple OS versions from multiple vendors: Microsoft Windows Server 2003, Windows Server 2008, Linux, and UNIX, for example.

Cons: Virtualized images are complete OS installations and can be extremely large files. Significant performance hits can occur (particularly on commodity hardware), and input/output operation-intensive applications can be adversely effected in such environments.

Para-Virtualization

In para-virtualization, a hypervisor exports a modified copy of the physical hardware. The exported layer has the same architecture as the server hardware. However, specific modifications are made to this layer that allow the guest OS to perform at near-native speeds. To take advantage of these modified calls, the guest OS is required to have small modifications made to it. For example, you might modify the guest OS to use a hypercall that provides the same functionality that you would expect from the physical hardware. However, by using the hypercall, the guest is significantly more efficient when it is run in a virtualized environment. (See Figure 10.)

Pros: Lightweight and fast. Image sizes are significantly smaller, and performance can reach near-native speeds. Allows for the virtualization of architectures that would not normally support full virtualization.

Cons: Requires modifications to the guest OS, which allows the OS to support hypercalls over native functions.

OS-Level Virtualization

In OS virtualization, there is no virtual machine; the virtualization is done completely within a single OS. The guest systems share common features and drivers of the underlying OS, while looking and feeling like completely separate computers. Each guest instance will have its own file system, IP address, and server configuration, and will run completely different applications. (See Figure 11.)

Pros: Fast, lightweight, and efficient, with the ability to support a large number of virtual instances.

Cons: Isolation of instances and security concerns around data are significant issues. All virtual instances must support the same OS.

Figure 10: Para-virtualization environment

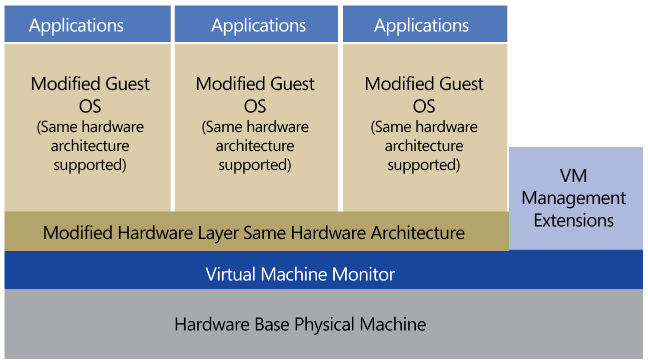

Applications
Applications
Applications
Modified Guest OS (Same hardware architecture supported)
Modified Guest OS (Same hardware architecture supported)
Modified Guest OS (Same hardware architecture supported)
VM Management Extensions
Modified Hardware Layer Same Hardware Architecture
Virtual Machine Monitor
Hardware Base Physical Machine

Figure 11: OS-virtualization environment

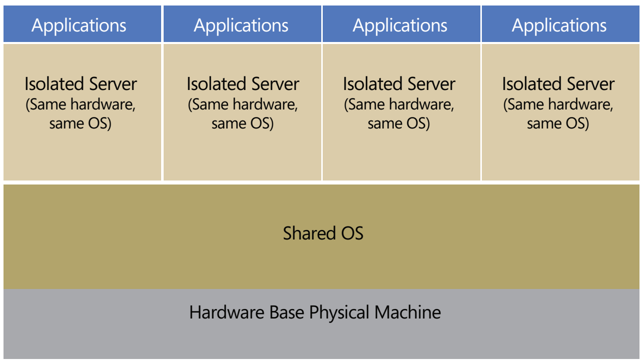
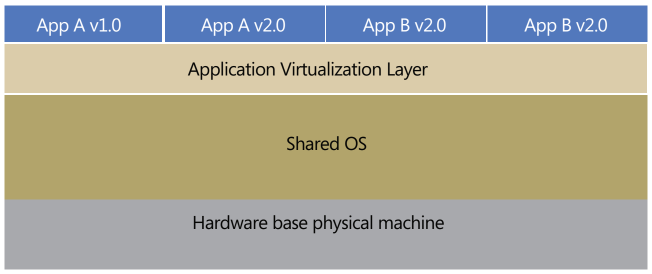

Applications
Applications
Applications
Applications
Isolated Server (Same hardware, same OS)
Isolated Server (Same hardware, same OS)
Isolated Server (Same hardware, same OS)
Isolated Server (Same hardware, same OS)
Shared OS
Hardware Base Physical Machine

Figure 12: Application-virtualization environment


App A v1.0
App A v2.0
App B v2.0
App B v2.0
Application Virtualization Layer
Shared OS
Hardware base physical machine

Application Virtualization

Application virtualization, as with any other type of virtualization, requires a virtualization layer to be present. The virtualization layer intercepts all calls that are made by the virtualized application to the underlying file systems, redirecting calls to a virtual location. The application is completely abstracted from the physical platform and interacts only with the virtualization layer. This allows applications that are incompatible with each other to be run side by side: Microsoft Internet Information Services 4.0, 5.0, and 6.0 all could run side-by-side, for example. This would also improve the portability of applications by allowing them to run seamlessly on an OS for which they were not designed. (See Figure 12.)

Pros: Improves the portability of applications, allowing them to run in different operating environments. Allows incompatible applications to run side by side. Allows accelerated application deployment through on-demand application streaming.

Cons: Overhead of supporting a virtual machine can lead to much slower execution of applications, in both run-time and native environments. Not all software can be virtualized, so is not a complete solution.

Toward an Enterprise Business Motivation Model

by Nick Malik

Summary

A business motivation model is a conceptual information model that demonstrates how the efforts of the business are, or are not, aligned to its goals. Producing and delivering reports that illustrate this alignment supports highly mature decision-making. But before we can collect data to produce these reports, the structure for data must be developed. Existing models are not sufficient to serve these needs. This paper outlines a new structure that can be used to model a wide array of business motivations in context with the structure and activities of the business. Adopting this structure can support an effort toward greater enterprise architecture maturity.

Introduction

Enterprise Architecture is an area of IT that focuses on answering some fairly difficult questions: How do we improve our alignment with the business? How do we increase the agility and flexibility of IT? How do we reduce the cost of managing information while improving the level of service to our business customers? The answers to each of these questions play out in the models, reports, and recommendations of the Enterprise Architect.

Questions like these are tough to answer. The typical enterprise is continuously changing to adapt to market conditions, competitive pressures, regulatory changes, and new opportunities. Talented architects, program managers, and business leaders frequently find themselves negotiating a careful path between the often competing goals of various business leaders across the enterprise. How do you plan for the future when everything is changing?

The answer to this conundrum lies in having a clear view of where the business is, and where it wants to go. If the business and IT leaders are not seeing the same measures or aligned to the same goals, much of the work of Enterprise Architecture loses its effectiveness.

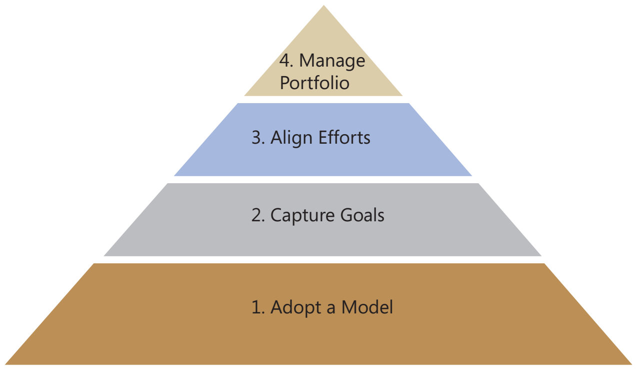
There are four steps to this process, as illustrated in the Figure 1. Each builds on the one before it, so you cannot effectively perform these out of sequence.

1. **Adopt a model:** There are many ways to describe how a business is designed to operate. Various models, from value chain analysis to SIPOC diagrams to business model diagrams have been used to illustrate various aspects of value, relationship, and production. Each has its own strengths. Each focuses on different things.

   Each business needs to carefully select a single model that is as comprehensive as possible, answers the questions that they want to ask, and can be executed by existing resources. Selecting the wrong model has the effect of wasting time to produce bad advice. Selecting many different models has the disadvantage of producing many different answers to pressing questions. As the saying goes, "A man with one watch knows the time. A man with two watches is never sure."

   A single conceptual model can be used to answer many different kinds of questions for different stakeholders. To extend the analogy, one watch can display the current time in two time zones, as well as provide a stopwatch and countdown timer. All can be consistent, yet each answers a different question.
2. **Capture Goals:** Once you have a model that can be used to describe the business, it is time to use it. Describe the business in that model. Capture both existing structures and future looking objectives and goals. Create a base of knowledge and understanding that you can turn to when questions arise. This is the data that you will use to generate consensus when strong-willed leaders make a bid to use emotion and passion to overwhelm reason. Get your data right, because it matters.
3. **Align Efforts:** Most businesses have ongoing "change efforts" and a few more waiting to get started. Each effort has their sponsor and many have ardent supporters. Alignment means making sure that each of these efforts is tied to a specific strategic goal or objective and that there is a reason to believe that accomplishing the effort will further the particular goal. Business Process Improvement techniques like Six Sigma are often used to demonstrate how a particular effort supports a goal through statistics, measurements, and management methods. After completing this stage, an enterprise architect can produce reports, based on verifiable data, that illustrate the relative value of each project to the goals of the business.

Figure 1: Building toward excellence — each step in describing the business builds on the one below it



4. **Manage Portfolio:** Having data is one thing. Using it is another. For an enterprise to make a move away from passion-driven decisions and toward decisions built on data, the data has to be there, and then the business has to make a commitment to use it. That said, having the will to use data, but no data to use, produces an equally ineffective situation.

The first three steps in this process are "all about the data." Step one is to select the data model and then, from there, to fill it with pertinent data to support decisions. Unfortunately, creating an overall model is difficult and time consuming. It is far better to adopt an existing model than to create your own.

Experience teaches us that the simplest of disagreements about seemingly small things (like "what is a service," "how many applications do we have," or "what metrics should we use") can drive wasted effort and produce inaccuracies in understanding. These inaccuracies can be sizeable, significant enough to influence portfolio decisions. At the extreme, valuable projects may be canceled, delayed, or delivered inefficiently, while low priority projects consume resources or block the way.

The Stakeholders of a Motivation Model

Clearly, Enterprise Architects can benefit from a shared understanding of the business. However, many more internal stakeholders, both inside and outside of IT, benefit from a common understanding of the business.

Enterprise Program Management Office (EPMO) – When a change in strategy is considered, the EPMO can quickly identify the projects that may be affected and how a change to the priority of those projects may affect other enterprise dependencies.

Line of business leaders – Through visibility into the goals and strategies of other businesses, the leaders can more effectively negotiate interdependencies to achieve their shared goals. It will be simpler to avoid taking dependencies when nimbleness is required, and easier to take a dependency when efforts need to be leveraged off of one another. Business leaders can also use this information to avoid short-term dependencies while planning for longer-term dependencies.

Business analysts – Analysis and Requirements gathering efforts can benefit through a single shared understanding of how each business unit and change strategy is aligned to enterprise goals. Using this understanding, analysts can collect requirements more efficiently, insure a complete coverage of their efforts, reduce the amount of time that business SMEs need to spend sharing information, and improve the results of their analysis.

Software designers and project managers – Having a common base of understanding of the business can help IT software professionals to produce information models, analysis models, software services, and information portals that more closely align to business goals without requiring the business analysts to document the business in redundant and often conflicting ways. The model can be used to trace the requirements back to their source, not just in terms of the people who provided the requirements, but the rationale that they used to justify them.

IT service managers – Through a shared understanding of the business, IT managed services, as described in ITIL, can be designed in a manner that minimizes overlaps, ensures a cost-effective use of resources, and insures that investments for flexibility are made in the right places. In addition, service managers can align their service level agreements to many parts of the business in a consistent manner and can use the business motivation model to identify and illustrate opportunities to deliver new services and improve existing ones.

Sources of Business Motivation Models

With all of the various benefits of a consistent model of the business, it is no surprise that a number of prior attempts at describing a generic business motivation model have come to light. One strong effort by the Object Management Group produced the OMG Business Motivation Model (the first official version was released in September 2007).

An outgrowth of a multi-year effort by the Business Rules Group, the OMG Business Motivation Model (OMG BMM) allows an analyst to describe the business motivation using a small set of core concepts: *means, ends, influencers*, and *directives*.

The OMG is not the only body that has attempted to solve this problem. The Open Group Architectural Framework (TOGAF) version 9.0 includes, for the first time, a business motivation model that is simpler than the OMG model and is based on the concepts of drivers, goals, objectives, and measures.

These two models are useful in their own narrow context, but the models defined by TOGAF and OMG can only answer a limited set of questions. We needed more than these models could provide. For example, neither model is able to recognize the source of multiple competing goals within an enterprise. Neither model can answer the following question: How do I determine which conflicting strategies, from different business models, must be rationalized in order to simplify shared business processes?

In order to answer questions like this, we need to represent a key concept, one that is not well represented in the existing models: one enterprise can use more than one "business model" (Figure 2). This business makes two types of electronic eavesdropping equipment that it sells through three different channels.

When you examine this enterprise, consider this central question:

How many businesses are represented here?

One must answer this question to understanding the nature of business motivation.

Understanding the Notion of a Business Model

Our answer, to the question above, is based on a comprehensive understanding of business architecture. If a business is a "way to make money," then each "way to make money" is a collection of elements that together form a **business model**. A business model is a specific configuration of these elements. Elements of a business model include things like customers, products, finances, and resources.

A business model goes beyond an abstract "vision" statement. It is a specific set of elements in a specific configuration. Dr. Osterwalder, in his Ph.D. thesis, outlined a set of elements that are part of a business

Figure 2: Value flows for the fictional company IBuySpy — how many business models are represented?

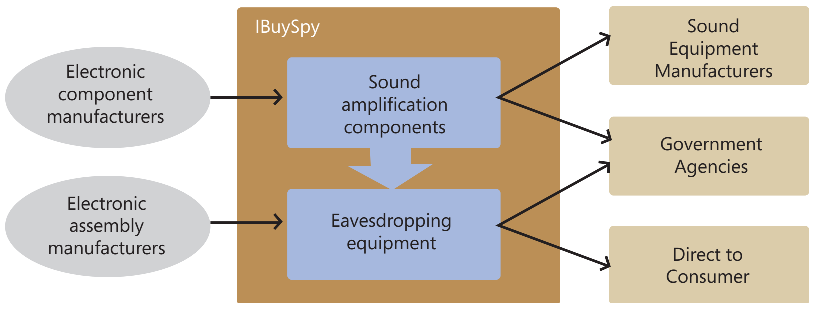
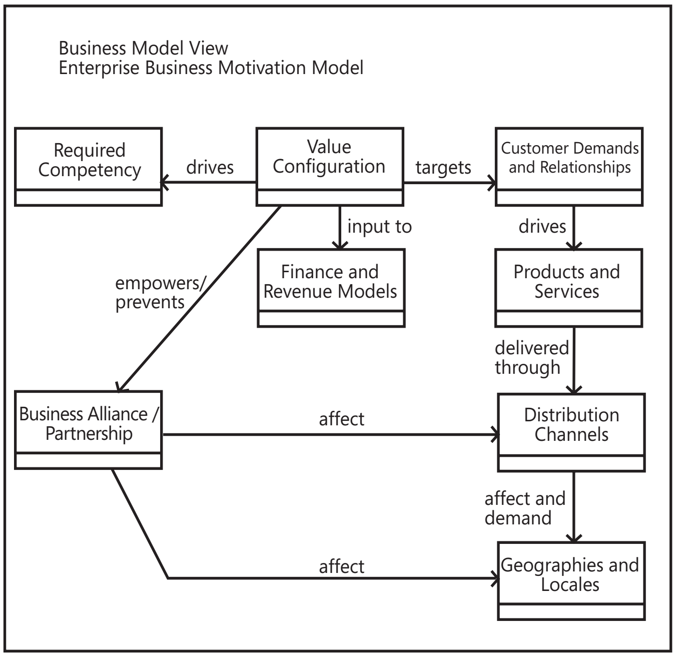

IBuySpy
Electronic component manufacturers
Electronic assembly manufacturers
Sound amplification components
Eavesdropping equipment
Sound Equipment Manufacturers
Government Agencies
Direct to Consumer

Figure 3: The elements of a business model — one of seven core models in the Enterprise Business Motivation Model



model. Our adaptation of his approach appears in Figure 3. For more information on the work of Dr. Osterwalder, visit his blog at http://business-model-design.blogspot.com/

Before we go into detail about the meaning buried in each diagram, let's take a moment to understand how to read the diagrams:

- The arrows from one box to another represent a relationship. The arrowhead tells you how to read the verb on the line. For example, in Figure 3, you may have read that "the value configuration targets the customer demands and relationships." Two verbs on a line can be read as two relationships. Therefore, the value configuration both empowers some partners and prevents others.
- Each type of information appears once and only once. Therefore, if you have a description of a business unit, it belongs in the part of the model that describes business units, and nowhere else. In that way, this model can be used to form a relational database.

In the Enterprise Business Motivation Model, the composition of all of the elements in Figure 3 forms the business model. When a business is just beginning to form, some or all of these elements would be collected together in a document called a "business plan." The business model is a statement of how the business is supposed to operate. It is, in effect, similar to a vision statement, but one that not only motivates behavior but helps to guide development of the corporate structure itself. The business model describes why you need business units, and what responsibilities they have. Business processes are created within the confines of a business unit to fulfill those responsibilities. At the center of this space is the business model.

If we return to the example of IBuySpy, we can ask the question again: how many business models does the enterprise have? IBuySpy sells to three different market segments, and within those segments, IBuySpy sells two totally different types of products. We could describe IBuySpy as having two business models (based on the type of products), three models (based on the market segments) or four business models (based on the value flows). While answering the question is context specific, and therefore beyond the scope of this article, one conclusion should be obvious: an enterprise can have more than one business model. IBuySpy has more than one way to make money.

Note that it is possible to represent some of this detail in the OMG BMM. It is possible, for example, to represent each business model as a vision. However, it is not possible to represent the unique relationship that each business model has with motivating the formation of the business units, business processes, and specific strategies.

Albert Einstein once said "Make everything as simple as possible, but not simpler." The Business Motivation Model from the OMG proved too simple for our needs. It was accurate, but not useful. It was this single realization that inspired the work described in this article.

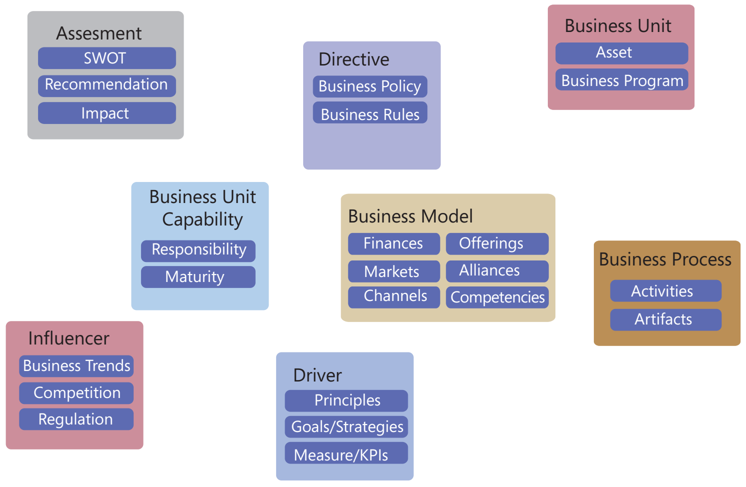
Building a New Business Motivation Model from the Ground Up

Placing the concept of a "business model" into the heart of a "motivation model" is a major change from current thinking. Existing models were not designed to represent an enterprise with all of its business units and competing strategies, business policies, and influences. So, in effect, we are starting over.

However, rather than start from scratch, we started from the same basic elements as those defined by the OMG and TOGAF models, and set out to analyze, compose, and create a single model that would produce an understanding of the business that is rich enough to meet the needs of the stakeholders described above.

From the OMG Business Motivation Model, we draw in the concepts of an Influencer, an Assessment, and a Directive. From the TOGAF model, we draw in the concepts of a Driver, a Business Unit, and a Business Process. From Osterwalder, we draw the concept of the business model. From the Innovation Value Institute, we draw in the concept of a Business Capability. At the starting gate, the "model" is simply a set of entities that may or may not be correct (Figure 4). But it is a place to start.

Figure 4: The raw materials to build a model from — concepts drawn from published and unpublished sources



What Exactly Does It Mean to Motivate an Enterprise?

The next step to developing a new motivation model is to establish the relationships between these entities. In order to clarify the relationships, we start with the nature of motivation. Motivation can be defined as:

motivation (noun) - internal and external factors that stimulate desire and energy in people to be continually interested in and committed to a job, role, or subject, and to exert persistent effort in attaining a goal. *(BusinessDictionary.com)*

This definition clearly states that motivation is about people. You can't truly motivate a business because a business doesn't have desire. You can only motivate the people involved in an enterprise. Equally important is the notion that motivation has different effects. You can motivate people to change a business, or motivate them to maintain a business in a specific manner. There is more than one kind of motivation.

A business motivation model is an attempt to capture the relationship between the factors that stimulate change (motivators) and the places in the business where those changes can be seen. Different motivating factors have different relationships. The relationships are what differentiate one type of motivator from another.

By looking at the relationships between 'motivators' and their effects on the business, we can discern three basic types of motivators.

1. **Influencers** are external to the business. They behave as they wish, and it is up to the business to notice them and respond. Influencers include competitors and analysts as well as abstract things like business trends and competitive opportunities. The relationship between an influencer and the business manifests in the form of a driver. An influencer inspires a driver, and that driver changes the enterprise. The driver translates influence into terms (and activities) that a business can understand.
2. **Drivers** are internal motivators that affect the model, structure or capabilities of a business. Drivers are change agents. They represent any entity or effort that directly drives the business to change. Drivers include strategies as well as mission statements and change projects. Drivers can also be the people who are responsible for bringing those strategies into reality.
3. **Directives** are statements of policy or rules that do not change the business. Instead, they provide the rules by which the business is required to operate. You can change the business by changing the directives that guide it. Directives affect the individual decisions that people make and therefore primarily influence business processes.

"A business motivation model is an attempt to capture the relationship between the factors that stimulate change (motivators) and the places in the business where those changes can be seen."

These three types of motivators are the cornerstones of the Enterprise Business Motivation Model, as illustrated in Figure 5.

We'd like to point out a distinction between this model and those described by the OMG and TOGAF. The model above groups together the concepts of a goal, a strategy, a principle, and a measure as types of drivers. More detail will follow on this point.

To understand the distinction between an influencer, a driver and a directive, consider this simple analogy.

When Frank returned from work one particularly hot day last summer, he noticed that some of the neighborhood children had set up a lemonade stand on the corner. Four children, all under the age of 12, were busy selling refreshments for 50 cents each and doing quite well for themselves.

When Frank mentioned the lemonade stand to his own family, he expected his children to shun the idea. After all, they were not particularly industrious children. Yet his 12-year-old daughter Megan and his 10-year-old son Daniel both showed great interest.

So the next day, with some encouragement, Megan and Daniel marched down to the curb and set up shop on a folding card table, selling their lemonade with cookies they bought by the dozen from Ozun's Bakery three blocks away. Megan's friend Alice joined them and worked through most of the afternoon. At 75 cents each, their lemonade was more expensive than the competition, but the cookies made a huge difference.

Frank had carefully coached his children on how to politely take an order and make the correct change. They all agreed not to "eat the inventory" or give away any free samples. At the end of the day, they had made enough money to pay for supplies with a little profit left over. Megan saved her money, and Daniel bought a new toy. But more importantly, in Frank's eyes, they had learned to take some initiative and were rewarded for it.

In this story, the **influencers** were the warm weather, as well as the neighborhood kids. Frank himself was an influencer. They inspired the business. But warm weather didn't put together the lemonade stand. That was Megan and Daniel's job. They created a **business model** that involved all aspects of the business, from suppliers to customers, including the strategy of selling their lemonade with cookies.

Megan and Daniel were **drivers**, responsible for making the changes needed. But those same kids, along with Megan's friend Alice, were also the **business unit**. The folding table, inventory of lemons and cookies, and even the hand-lettered signs, are resources of the business unit.

Megan and Daniel learned some of the skills they would need (**capabilities**) like how to make change, how to treat customers well, and how to replenish the lemonade, from their father and then taught Alice. In addition, they had decided on some key **directives** ("don't eat the inventory" and "no free samples") that helped to insure that there would be a profit at the end of the day. They kept those directives in mind as they performed their **business processes** throughout the day.

Figure 5: Enterprise Business Motivation Model

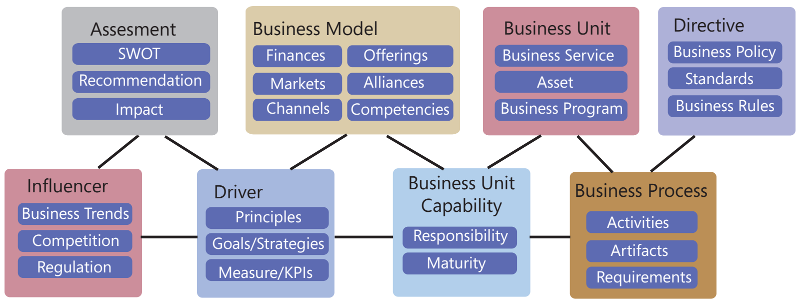
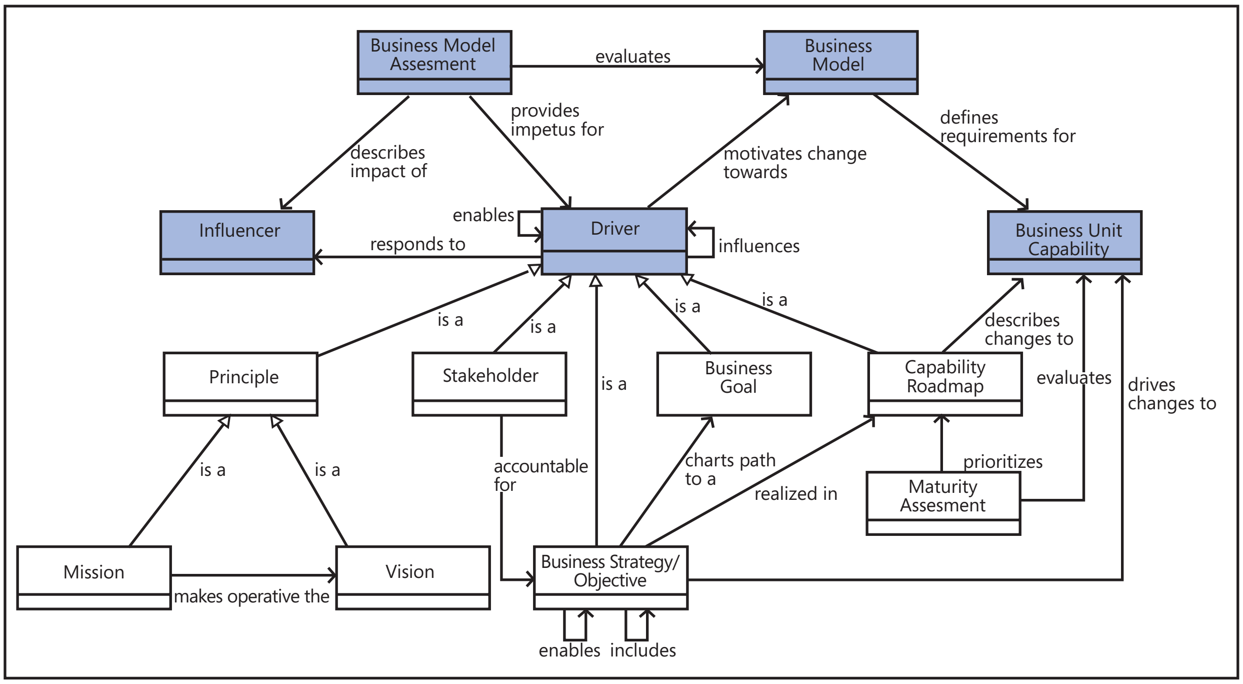


Figure 6: Types of business drivers and how they relate to one another



Types of Business Drivers

As you might imagine, each of the base types described so far have specific subtypes that draw out the distinctions within the model and provide for clear traceability. Figure 6 highlights the various types of drivers within the enterprise business motivation model.

As you can see from this model, the mission and vision of the enterprise are not contained within the description of the business model. While they may influence the business model, the mission and vision are statements of principle. They are used to drive action, and are therefore drivers.

The model presented here is different from the existing motivation model developed by the OMG. In the OMG model, a mission statement is a means to an end described by the vision statement. Strategy and Tactic are subtypes of "course of action." An extract from the OMG Business Motivation model, highlighting these elements, is shown in Figure 7.

The reason for the difference is simple: the Enterprise Business Motivation Model considers the complex (yet common) scenario where some stakeholders may want to change the business while other stakeholders may not want to change it, or may have conflicting ideas about how to change it. If we are going to effectively understand the dynamics of change, we need to be able to describe both the business as it stands today, and rationale for changing it.

Both of these concerns, the "right now" and the "not yet," exist at the same time, and both must exist independently in the model in order to understand and trace the impact of change across the organization.

Business Units and Their Capabilities

Whether your business is a lemonade stand or a multi-national corporation, a business model can be constructed to describe how the business can make money. That business model describes the way the enterprise must behave in order to make money.

The business model demands that the business must have some resources, and that those resources must be applied in a particular configuration in order to produce a valuable result. These resources live within the 'business unit' part of the business motivation model.

A business unit is not only the hierarchical list of people employed by the enterprise, but also the assets, products, services, liabilities, and any other "item of value" that tends to appear on a balance sheet or product catalog. The business units are the organizational 'parts' of

Figure 7: The existing OMG BMM represents various drivers but not the business itself

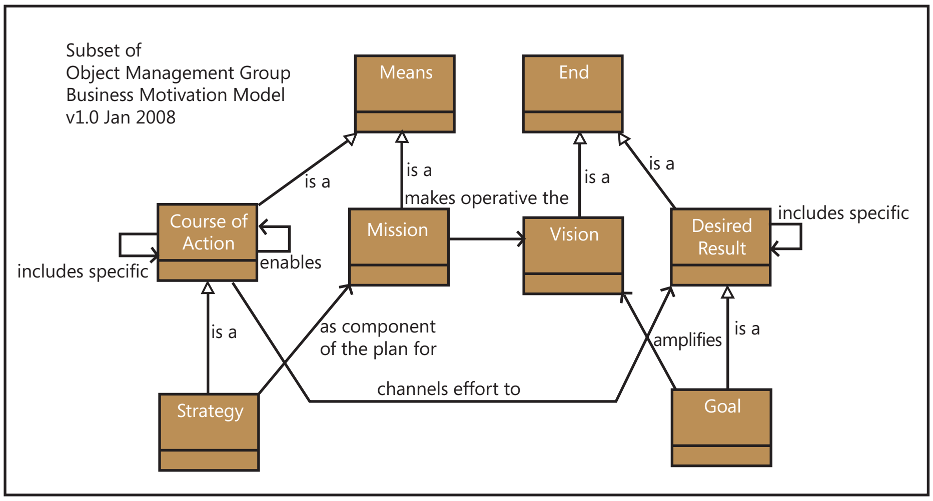
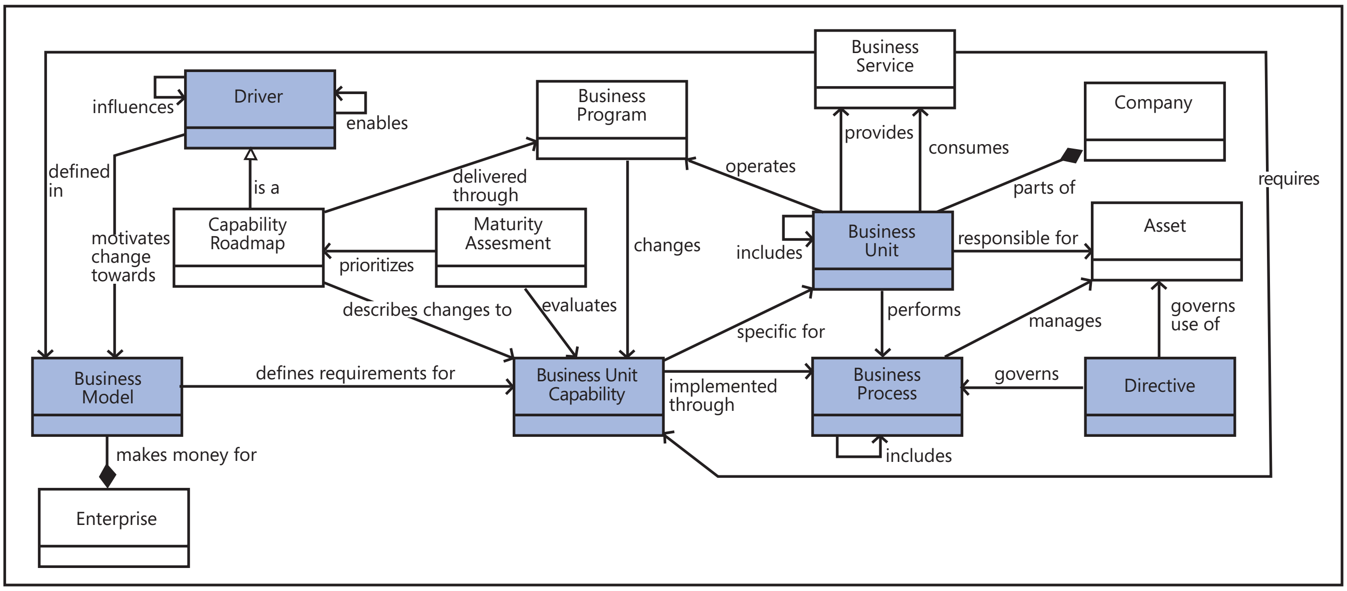


Figure 8: Elements of a business unit



an enterprise that actually do the work. Figure 8 highlights the key concepts within the business unit view.

Many of the concepts in this model are identical to those described by the OMG Business Motivation Model, including the relationships surrounding **Assets**. However, this model adds the concepts of a business capability, capability roadmap, maturity assessment, and business service. (Note that the OMG model uses the term "Organizational Unit" which should be read as a synonym for "Business Unit.")

Where the business model describes the required competencies of a business, the "**business unit capability**" is the description of a specific ability, to be performed by a specific business unit. For example, if our business model says that we need to be competent at "selling widgets," there must be a business unit ("sales") that has the ability to perform this task ("sell widgets") using a set of business processes ("generate lead," "offer product," "close sale"). We included this concept in the Enterprise Business Motivation Model because it is the anchor for one of the two types of assessments: the capability maturity assessment.

A **maturity assessment** evaluates how well a business unit performs a required capability. A highly mature capability is efficient, effective, and repeatedly produces a high quality result. A finding of "immature" illustrates the areas of the business that could be improved. It will not, however, illustrate the order in which those improvements will be made. That is where the "**capability roadmap**" comes in. A capability roadmap answers the question "which capability do we need to improve, and when, in order to improve our enterprise." Such a roadmap is, itself, a driver of change.

The concepts of **business program**, **company**, and **asset** are external to the EBMM. They are illustrated here as reference points for extending and connecting this model with others.

A **business service**, in this model, is a packaging of business capabilities so that an offering can be made to a customer or partner (including an internal customer). The business model defines what business services must exist (if any). The business service calls upon specific business units to provide the effort needed for that service.

Governing Business Processes

Business Processes are an integral part of the business motivation model because one of the three types of motivation, **directives**, apply primarily to business processes. A business process is a series of activities, usually performed in order, that create value for the customers of the process. Directives include business rules and business policies, and they are useful for guiding and governing the behavior of these business processes. (See Figure 9.)

Understanding and modeling the business rules is an important activity, and the work of the Business Rules Group to create a methodology for describing business rules is a key step toward maturity in this space. Readers are encouraged to take a look at the SBVR standard, published by the OMG, in order to dig deeper into this critical area.

Figure 9: Elements related to business process and directive

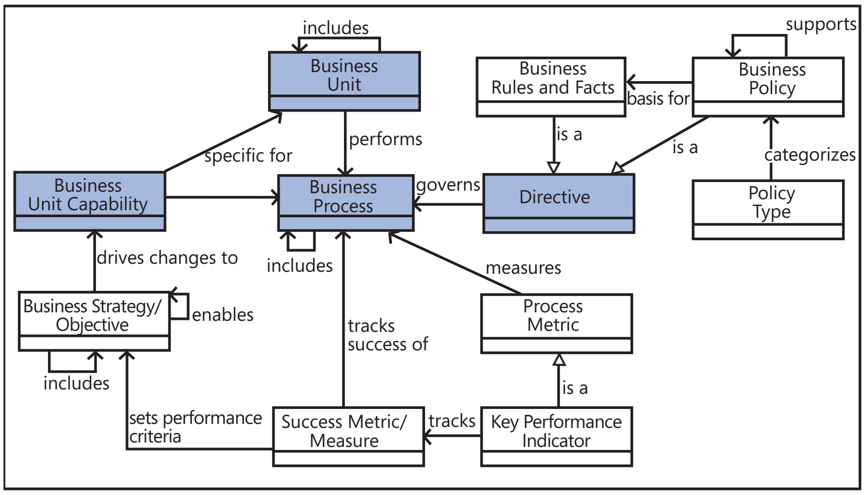
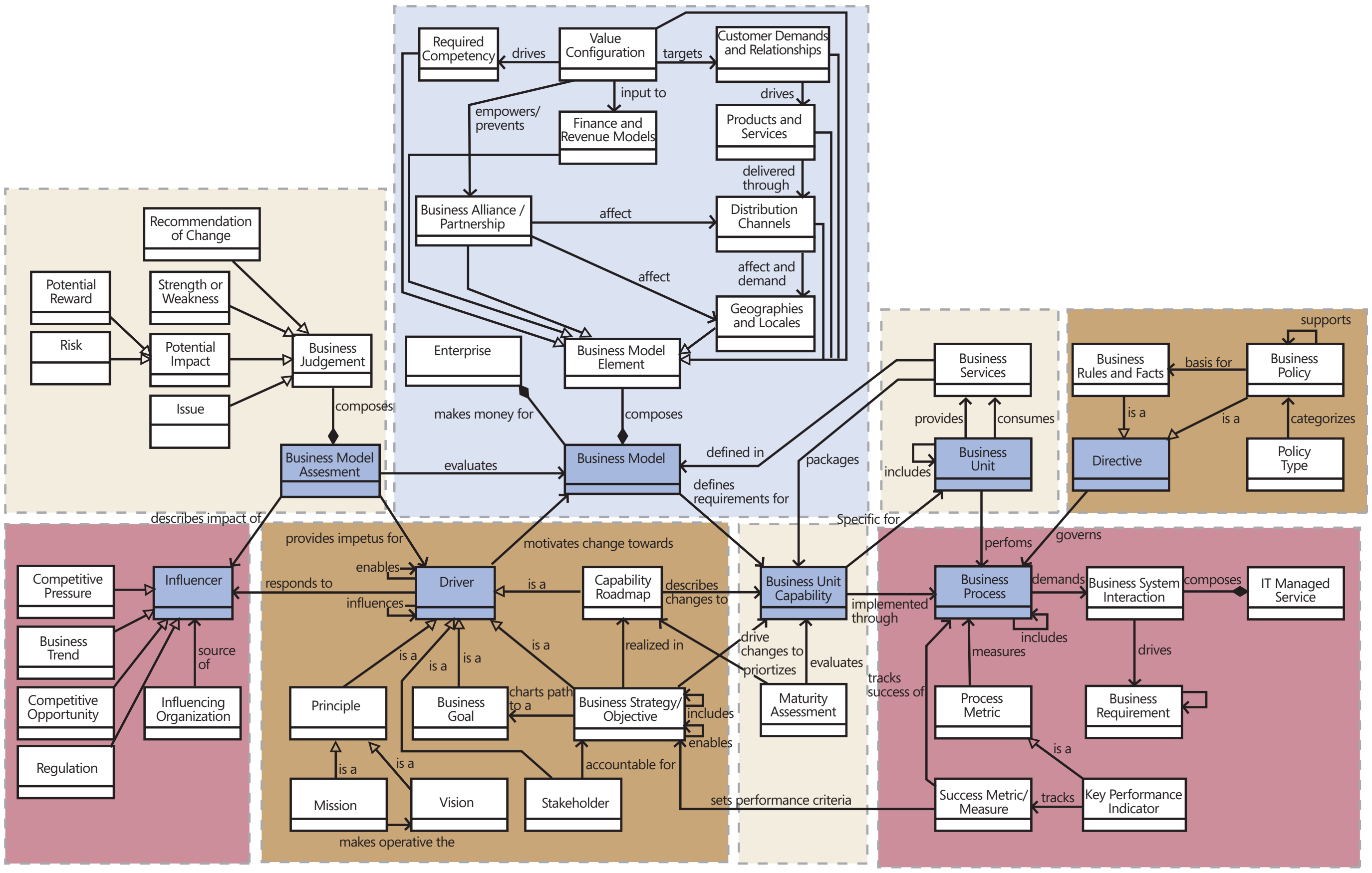


Figure 10: Detailed structure of the Enterprise Business Motivation Model



A **success measure** may not be a direct measurement of a business process. Rather, a success measure is the measurable understanding of "success" that may be cited in the business strategy statements themselves. For our Lemonade stand, "reducing the amount of waiting time" may have the effect of increasing sales because fewer people will leave the line and those that stay will be more satisfied. In this case, our Key Performance Indicator (reduce wait time) is selected because it tracks the success measures (increased revenue and increased customer satisfaction).

Once again, it is the relationship of the metrics to the rest of the model that highlight their nature. A metric is not a driver, but it can be used by a driver to motivate change in the business.

The Overall Structure of the Enterprise BMM

The overall structure of the Enterprise Business Motivation Model is illustrated in Figure 10. This diagram illustrates each of the core entities for the Enterprise Business Motivation Model. Other elements can be connected to this model, including requirements for IT software, to indicate traceability from the business drivers down to the changes needed in the IT infrastructure.

Conclusion and Next Steps

Clearly, the Enterprise Business Motivation Model is substantially different from the models that exist in other frameworks. To the greatest extent possible, we used the elements and relationships that were defined in other models as guides in order to ease the process of adopting a new motivation model into an enterprise architecture program.

Adopting a rich and mature model for capturing the structure and motivations of a business is a key part of growing the maturity of Enterprise Architecture. Many of the most important questions of enterprise architecture require a solid understanding of how the business is structured and aligned to deliver value, and a single comprehensive model of business motivation allows an Enterprise Architect to answer those questions.

We invite comment on this model and hope to see it merge into the EA frameworks that are appearing in various standards bodies around the world. If you have suggestions, questions, or comments, please address them to Nick.Malik@Microsoft.com or visit MotivationModel.com to join the discussion.

About the Author

Nick Malik has been on the leading edge of software development as a developer, architect, and business leader for 28 years, including stints at Racal, American Express, IBM, and Acadio. In his current role as an Enterprise Architect in Microsoft's internal IT group, he considers himself fortunate to be surrounded by some of the most brilliant minds in Microsoft. When he is not busy being a geek, he can be caught watching the latest movies, exercising with his wife Marina, or horsing around with his kids Max, Andy, and Katrina.

Follow up on this topic

- Enterprise Architecture: http://msdn.microsoft.com/architecture/ea
- Enterprise Business Motivation Model: http://www.motivationmodel.com
- Innovation Value Institute: http://ivi.nuim.ie/
- Inside Architecture blog: http://blogs.msdn.com/nickmalik
- Object Management Group: http://www.omg.org

Developing Parallel Programs

by Ranjan Sen, Ph.D.

Summary

Parallel programming is an extension of sequential programming; today, it is becoming the mainstream paradigm in day-to-day information processing. Its aim is to build the fastest programs on parallel computers. The methodologies for developing a parallel program can be put into integrated frameworks. Development focuses on algorithm, languages, and how the program is deployed on the parallel computer.

Introduction

Parallel programming utilizes concurrency to achieve high-performance computing. Historically confined to supercomputing parlance, parallel programming today is becoming the mainstream paradigm in regular day-to-day information processing. This is energized by the widespread availability of multi-core multiprocessors and cost-effective server clusters. The software industry in general is integrating rich desktop and server software-development tools with new-generation parallel-processing tools. Examples include use of Microsoft Visual Studio and the .NET extension for parallel computing, Microsoft Windows HPC Server, decentralized distributed service-oriented programming, grid computing, and so on. Many of these are rich in ideas that are based on decades of research; side-effect–free functional programming, giving protection against race; data-flow paradigm for non–von Neumann architecture; and many more.

Parallel programs are built by combining sequential programs. The goal is to allow independent sequential programs to run in parallel and produce partial results that then are merged into the final solution via different combination patterns. We want to get correct, bug-free parallel programs that can deliver performance and possibly other benefits, such as reliability, availability, and fault tolerance that is integrated with an existing software ecosystem.

Parallel programming is fast becoming an essential developer skill. Multifarious variations in parallel-processing technology, from clients to server clusters, provide diverse developer toolsets and runtime environments. Knowing the basic concepts helps in a better comprehension of the complexity, and it is never more crucial to the developer than now.

Correctness and Performance

Developers must continue creating correct and efficient applications. Both correctness and performance confirm that a program produces the result that it is supposed to deliver within an expected time frame. In establishing this, the conventional model that is used for sequential computers is von Neumann's "stored-program" model. In the "stored-program" model, there is a single thread of execution; instructions are executed by one processor at a time.

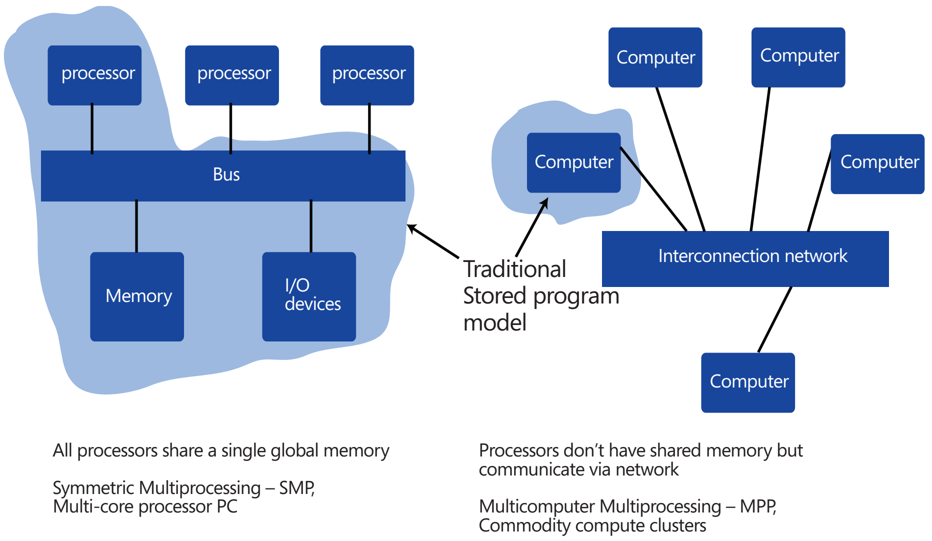
In parallel computers, there is more than one processor, each of which executes an execution thread simultaneously. Parallel-computer models that are used for correctness and performance analysis are simple extensions of stored-program models. The two models that are used are the shared-memory model and the distributed-memory model (Figure 1). In the first model, a common memory is shared by all processors; in the latter model, it is not.

The goal of achieving a high-performance application is achieved by having several sequential programs run simultaneously, overlapped in time, with the common goal of solving the same problem. This leads to two important concepts: decomposition and pattern.

Decomposition and Pattern

Decomposition is the art of splitting (or decomposing) a problem into independent parts to be solved concurrently. Each of these parts might obtain (partial) results that can be combined to obtain the final result; we need a combining scheme (*or pattern*) for these parts. We can establish

Figure 1: Parallel computers — shared-memory and distributed-memory models


processor
processor
processor
Bus
Memory
I/O devices
Traditional Stored program model
Computer
Computer
Computer
Computer
Interconnection network
Computer
All processors share a single global memory
Symmetric Multiprocessing – SMP, Multi-core processor PC
Processors don't have shared memory but communicate via network
Multicomputer Multiprocessing – MPP, Commodity compute clusters

correctness and analyze for performance for each of the parts, as well as the pattern that is used, to argue about correctness and performance of the overall parallel computation.

As an example, consider the problem of finding maximum of 16 data. We can divide the data into four parts of 4 data and find the maximum for each of these parts concurrently on four processors. Then, we can find the maximum of the maximums. The sequential parts that are used are the method of finding maximum of 4 data. The pattern that is used is finding four intermediate possible maximums in parallel, and then finding the actual maximum. Figure 2 illustrates this scheme.

The idea of using decomposition and pattern is not new (see a standard text, such as [Quinn, 2004] in References). One can think of decomposition as finding one or more pieces of sequential algorithm (sequential program) that can be run concurrently on more than one processor. Such a sequential piece is often referred to as *computational grain* or simply the *grain* of a parallel computation. Similarly, a pattern corresponds to a high-level algorithm of coordination or a *composition* scheme. Several patterns are known to be useful (see [Mattson, 2005] in References).

Analysis of Parallel Programs

Parallel-computer models can be used to analyze parallel algorithms or the corresponding programs for correctness and performance. A parallel algorithm is correct if both the sequential program and the pattern used are correct. We can follow methods that are similar to those used for sequential programs/algorithms to establish correctness. We can use the same approach for *debugging/diagnosing* a faulty parallel program in this way.

In determining correctness, we examine the memory states of data that the program is supposed to transform. In parallel programs, *dependencies are linear within the sequential pieces of programs* that run in parallel. However, the *pattern may have nonlinear dependencies*. For example, the pattern that is used in the preceding algorithm to find the maximum of 16 integers is a correct scheme, because the programs Max1, Max2, and the composition scheme that is given by runMax all are correct. Figure 2 shows a graph of the dependencies of the sequentially executing programs that are given by Max1 and Max2, as expressed in runMax.bat. More often, the nodes of these graphs can represent either data or tasks (computation), or both. In the former case, it is called a *data-flow graph*; in the latter, it is called a *task graph*.

"One can think of decomposition as finding one or more pieces of sequential algorithm that can be run concurrently on more than one processor. Such a sequential piece often referred to as computational grain or simply the grain of a parallel computation. Similarly, a pattern corresponds to a high-level algorithm of coordination or a composition scheme. "

Similarly, an algorithmic approach can give us an estimate of performance. For example, we can reason that in the first stage (of parallel computation), four execution instances of Max1 on four processors can take place in time T (to find maximum of 4 integers) concurrently. In the second stage, we may have one execution instance of Max2 on one processor. Then, the overall time of the algorithm that is used is 2T with four processes (a *process* is an execution instance of a program). This estimate gives a good point of reference as to what to expect.

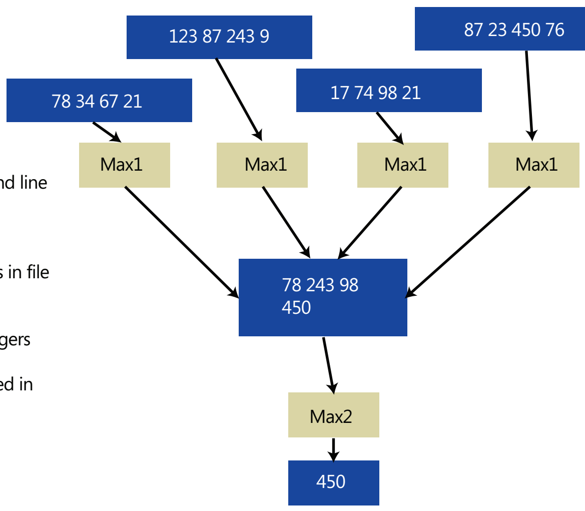
Speedup: A Measure of Performance

The ratio of time that is taken by a sequential program to time that is taken by a parallel program is called speedup. In general, you can find different parallel algorithms to solve a problem. It is important to know which achieves the best performance.

Amdahl's law gives 1/[S + (1-S)/n] as an estimate of maximum speedup, where S is the fraction of inherently sequential code in an application, and n is the number of processors. By way of illustration, in the preceding maximum-finding program, the fraction S is given by the program Max1.c. In the example of finding maximum of the 16 integer, the fraction S is 0.2 (four instances of Max1 and one instance of Max2 run sequentially would be 100 percent) and, by Amdahl's law, speedup can be at most 2.5 with four processors.

The notion of scaled speedup is given by Gustafson-Barsis's law. According to it, scaled speedup is bounded by n + (1–n)*s, where n is the number of processors, and s is the ratio of the time that is spent in the serial part of the program versus the total execution time. In our preceding example, s = 1/($\log_4 16$) = 0.5. So that, for n = 4, this is 2.5; for n = 16 (s = 0.3), it is 11.5; for n = 64 (s = 0.25), it is 49; and so on.

Figure 2: Scheme showing problem of finding maximum of 16 data



Max1.c
Finds maximun of 4 command line integers
Saves it in file l.txt
Max2.c
Finds maximum of 4 integers in file l.txt
runMax.bat
Runs Max1 4 times on 4 integers each, to find max in each
runs Max2 on 4 integers saved in l.txt by the earlier runs

Parallel-Computing Platforms

In the early days of parallel processing, architectures were expensive and specialized. Recently, multi-core processors have become the de facto processor technology.[1] The multi-core phenomenon caused a large-scale impact on game software in early 2000, when Sony used multiple processors for its PlayStation PS2.[2] At the same time, high-performance server-cluster programs are superseding the supercomputers in performance.[3] There is also the trend of special hardware, such as gate arrays (for example, FPGA); Graphics Processor Units (GPU) or cell processors are bringing out new ways to assemble parallel architecture. Today, diverse scenarios of distributed systems are using parallel processing for improved resource utilizations, throughput, reliability, and availability.

In the large-scale parallel-computing platform technology, operating systems are updated for multi-core processors, and new and extension in optimizing compilers and development systems are being crafted out. In the distributed-systems arena, we are seeing rapid integration of mainstream enterprise-grade technology, as well as a growth in loosely coupled systems. Some of the related software and switching technology are mentioned later.

Myrinet is an ANSI[4] standard that is used widely in computer clusters.[5] Features include an interface card that uses firmware to process protocols and off-loads host processors, OS bypass for low-latency communication, and so on. Ten-gigabit Ethernet is an IEEE standard and is the fastest version of the Ethernet standard. This is 10 times as fast as Gigabit Ethernet, which is the technology for transmitting Ethernet frames at the rate of one gigabit per second. Network switched fabrics, such as InfiniBand,[6] are commonly used in parallel-computer architectures

Computer Clusters

Clusters of computers and workstations are a very popular hardware/software commodity as a cost-effective parallel-processing platform (see [Sterling, 2002] in References). However, administering and managing such clusters can be quite complex. Clusters of Windows Server (here, called Windows HPC Server) address these problems in addition to the high-performance platform goals. Windows HPC Server provides necessary cluster services and tools, including Microsoft MPI, job scheduler, and cluster-management service to make powerful cluster solutions in diverse scenarios. The high-performance ranking is in the top 10 of the top-500 list.[7] New-generation network services are added with MSMPI for support of very high-speed communication between physical computes in a cluster.

The job scheduler can run jobs that are defined in service-oriented architecture (SOA), in addition to traditional job definitions, as a composition of tasks that execute programs around the cluster nodes. Also, it accepts jobs via the HPCBP service interface — thus allowing interoperability from any platform that adheres to the grid-interface protocol.

A Dryad[8] is an infrastructure for using the resources in a cluster or data center that allows a programmer to express a program in terms of sequential programs and connecting them via one-way channels. Dryad can express common computing frameworks, such as map-reduce[9] or the relational algebra; it handles job creation and management, resource management, job monitoring, visualization, fault tolerance, re-execution, scheduling, and accounting. (See Figure 3.)

"The job scheduler can run jobs that are defined in service-oriented architecture (SOA), in addition to traditional job definitions, as a composition of tasks that execute programs around the cluster nodes. Also, it accepts jobs via the HPCBP service interface — thus allowing interoperability from any platform that adheres to the grid-interface protocol."

SSIS SQL Server 2005 Integration Service has been built on top of Dryad. It executes many instances of Microsoft SQL Server, each on a Dryad vertex, and uses fault tolerance and scheduling services. This is being used currently as part of the AdCenter[10] log-processing pipelines.

The goal of DryadLINQ,[11] a related project, is to make distributed computing on a large computer cluster simple enough for ordinary programmers. DryadLINQ translates LINQ programs into distributed Dryad computations and distributes them to different nodes of a cluster. The features include declarative programming; automatic parallelization (both multi-core on a workstation and cluster-wide); integration with Visual Studio (Intellisense, code refactoring, integrated debugging, build, source-code management); automatic serialization; job graph optimizations, via both static term rewriting and dynamic query-plan optimizations; and conciseness.

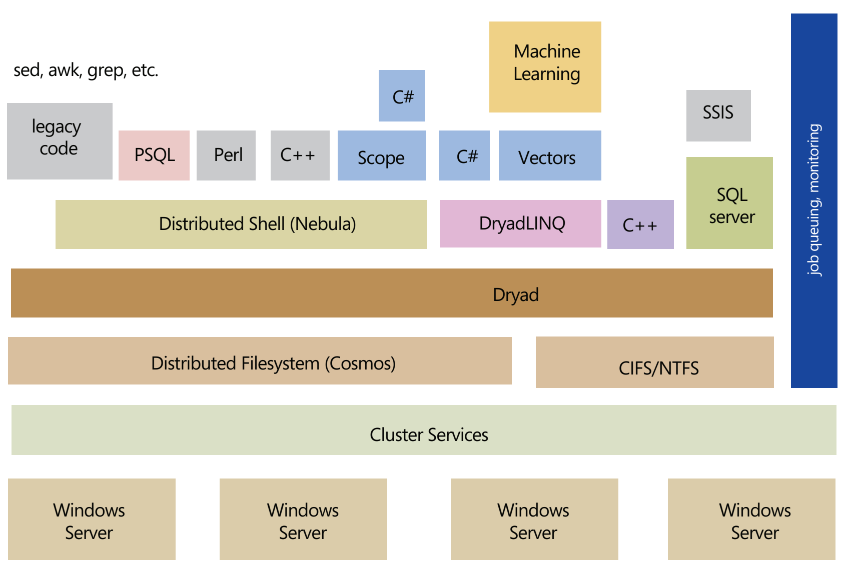
Decentralized Software Services (DSS)

The DSS runtime is built on top of Concurrency and Coordination Runtime (CCR),[12] which is a highly concurrent, message-oriented programming model. CCR has powerful orchestration primitives, enabling coordination of messages without the use of manual threading, locks, semaphores, and so on. CCR addresses the need of service-oriented applications by providing a programming model that facilitates managing asynchronous operations, dealing with concurrency, exploiting parallel hardware, and handling partial failure.

Run-time files for CCR and DSS are available on the Microsoft .NET Framework and .NET Compact Framework. The DSS protocol is being distributed via the Microsoft Open Specification Promise.[13] The availability of the protocol will make communication between a variety of hardware and software easier. Binary serialization gives faster throughput. VPL development tools support regular as well as mobile development. Also, there is a DSS Service–generation tool: Visual Simulation Tool.

Among other large-scale clusters and new-generation integration technologies are the cloud computing architecture from Microsoft Windows Azure,[14] Amazon[15] and the Eco-Science Analysis project.[16] The ecological data is huge; databases, data cubes, and Web services have been used in the context of data handling. In science with electronic means (eScience), tools such as Excel, MatLab, ArcGIS, and SPlus are used. The challenge is how to connect the data in the cloud to the analysis tool on the desktop without requiring full data download.

Figure 3: Dryad architecture



Developing Parallel Programs

To understand better the design, we use a model at a higher level than the shared-memory model or the distributed-memory model. This is the task/channel model (see [Quinn, 2004] in References). A task is a program, its local memory, and a collection of I/O ports. This is represented by a process in an operating system (threads are contained in processes). The local memory contains the program instruction and data. A task can send local data values to other tasks via output ports and receive data values from them via input ports. A *channel* is a message queue that connects the output of one task to the input port of another. Data values appear at the input port in the same order in which they are placed in the output port at the other end of the channel.

Figure 4 gives a conceptual view of the task/channel model. Tasks are represented as circular nodes and channels are represented by directed lines. A direct line between task i and task j indicates a dependency of task j on task i. Independent tasks can run in parallel. Consequently, if the tasks are executed in parallel, task j will have to wait for task i to send data. This is called *data dependency*, and the graph is a data-flow graph. However, if the channels represent completion signals, this depicts *control dependency*; in that case, the graph is a control-flow graph (also called a task graph).

Task parallelism is achieved when independent tasks execute concurrently. Note that tasks that correspond to nodes that have identical labels in Figure 4 run in parallel, and we achieve task parallelism. *Data parallelism* is when a task or tasks operate(s) on disjointed sets of data.

Consider a four-step process for parallel-program design: partition, communication, agglomeration, and mapping (see [Foster, 1995] and [Dongarra, 2003] in References). *Partitioning* is the process of dividing the computation and the data into pieces or primitive tasks. Increasing the number of primitive tasks reduces the inherently sequential fraction in the parallel program that is designed. This helps in raising the parallelism that is possible, according to the theoretical bounds that are given by Amdahl's law and Gustafson-Barsis's law. Communication considers the plan for inter-process communication necessary for the parallel program.

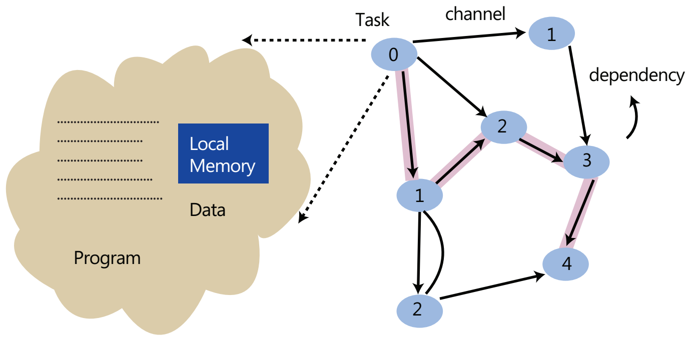
Agglomeration is the process of grouping tasks into larger tasks in order to improve performance or logical abstraction. Mapping is the process of assigning tasks to processors. The goal is to balance computation and communication loads in order to maximize processor utilization and minimize inter-processor communications.

The Parallel-Programming Ecosystem

Parallel programming aims to build the fastest programs on parallel computers. These programs must be correct as well as amenable to modern software-engineering practices for efficient life-cycle management. The main factors to achieve this are the following:

1. Algorithm that is used
2. Implementation language and interfaces
3. Programming environment and tools
4. Target parallel-computing platform

Figure 4: Task/channel model — conceptual view



There is considerable literature on designing parallel algorithms (see [Akl, 1989], [Leighton, 1992], and [Miller, 2005] in References). Essentially, the basic approach is finding sequential pieces that can run in parallel and combining efficiently the results that they obtain.

Tools for developing parallel programs are based on four different approaches. The first is to extend a compiler. The second is to extend a sequential programming language and allow core parallel-programming schemes to be captured from known environments. The third is to add a parallel-programming layer; this is a layer on a sequential core that controls creation and synchronization of processes and partitioning data. The fourth is to create a new parallel-programming language, such as Fortran 90, High Performance Fortran,[17] or C. [18] We will discuss the two more popular approaches: OpenMP, which is an extension of C++, and Message-Passing Interface (MPI). [19]

OpenMP

OpenMP is based on the shared-memory model. The standard view of parallelism in a shared-memory program is fork/join parallelism. When the program begins execution, only a single thread (master thread) is active. The master thread executes the sequential portions of the algorithm. At points where parallel operations are necessary, the master thread forks (creates or awakens) additional threads. Then, the master thread and these new threads work concurrently through the parallel section. At the end of the parallel code, the created threads die or are suspended, and the flow of control returns to the single master thread.

A sequential program is a special case of a shared-memory parallel program—one that has no fork/join. The shared-memory model supports incremental parallelization, which makes it possible to transform a sequential program into a parallel program one block of code at a time. This is a quick way to develop a parallel version of an existing program. However, the underlying algorithm might not be the best parallel algorithm.

OpenMP makes it easy to indicate when the iterations of a `for` loop can be executed in parallel. See the second commented-out line in the following code snippet:

```
#pragma omp parallel private( t, x,y,local_count)
   {
         local_count = 0;
         Random^ rand = gcnew Random();
         t = omp_get_num_threads();
   #pragma omp parallel for
         for (int i = tid; i < samples; i += t) {
               x = rand->Next(0,10000)*.0001;
               y = rand->Next(0,10000)*.0001;
               if (x*x+y*y <= 1.0) local_count++;
         }
   #pragma omp critical
         count += local_count;

   }
```

The `#pragma omp` parallel for directives in OpenMP are denotations to the C++ compiler to process the portion in the curly brackets for parallel execution. Also, note how it is possible to define parameters that are private to each thread (to reduce contention for shared memory), and the use of a critical segment using `pragmas`.

In the preceding example, private variables are declared via a clause of the parallel `pragma` declaration. This allows avoiding contention when all threads access these variables (in the parentheses). Note that we have used a critical segment to allow the threads to add their results back to the value to the shared variable `count`.

Message-Passing Interface (MPI)

MPI is a standard programming library that is available from FORTRAN, C, or C++. It enables creation of a distributed-memory programming environment that can be established across different physical computers. There are different flavors of MPI: Microsoft MPI (which is based on MPI-2[20]), HP MPI, Intel MPI, Open MPI, LAM/MPI, MPICH, FT-MPI, and others.

SPMD for a distributed-memory parallel-computer model is the underlying approach of the programming. The same program is run on all participating computers (processors, cores, and nodes). An MPI runtime makes services available through application programming interfaces (APIs) for necessary support of parallel computation. Processors are identified by rank in a communication world, and it is possible to have one-to-one as well as collective communication between them.

In Figure 5, three physical computers are shown to host multiple processes that have distinct ranks.

The entire collection forms a communication world, so that any processes that are in it can access each other via message-based communication.

A simple MPI program is shown in the following code snippet:

```
 #include "mpi.h"
   #include <stdio.h>

   int main(int argc,char* argv[]) {

   int  numtasks, rank, rc;

   /** initialize MPI environment **/
            rc = MPI_Init(&argc,&argv);
            if (rc != MPI_SUCCESS) {
     printf ("Error starting MPI program.
Terminating.\n");
     MPI_Abort(MPI_COMM_WORLD, rc);
            }
   /** get the number of processes and their ranks
**/
   MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);

printf ("Number of tasks= %d My rank= %d and
Hostname=%s\n",  numtasks,rank,getenv("COMPUTERNAM
E"));

   MPI_Finalize();
   }
```

MPI functions and constants are defined in the mpi.h file and the data types; operations and constants are similar to the standard C and FORTRAN equivalents. For complete list of MPI functions, see [Gropp, 1999] in References.

New-Generation Tools

Java and .NET programming languages have programming extensions to support parallel programming in managed runtimes (see [Lea, 1999] in

"SPMD for a distributed-memory parallel-computer model is the underlying approach of the programming. The same program is run on all participating computers (processors, cores, and nodes). An MPI runtime makes services available through application programming interfaces (APIs) for necessary support of parallel computation. "

References). Parallel FX Library (PFX) runs on .NET Framework 3.5 and the to be released new .NET Framework 4.0.[21] The .NET Framework provides a runtime that is called the CLR and which runs the code in a managed environment, with automatic garbage collection, just-in-time execution, added code-access security, and so on. In this way, parallel processing is integrated with the hosts of modern .NET-based technologies.[22]

The underlying technique in PFX is to use anonymous functions—building expressions by using them and then executing such expressions in parallel. It is convenient to represent anonymous functions as anonymous delegates[23] or as lambda expressions.[24] Also, it is possible to create expression trees by using nesting of expressions; and, with the help of lambda expressions, we can use functions in such expressions.

Imperative task parallelism is achieved via the task parallel library: System.Threading.dll. Task Parallel Library (TPL) is built on a scheduler that uses cooperative scheduling and work stealing to achieve fast, efficient scheduling and maximum processor utilization.

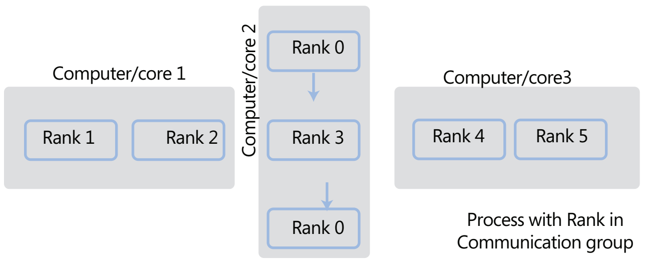
TPL provides the `System.Threading.Parallel`, `System.Threading.Tasks.Task`, and `System.Threading.Tasks.Future<T>` types, respectively. The first type is used for parallelizing loops and regions. The static methods that are available with the `Parallel` type are `For`, `ForEach`, and `Invoke`. For example:

```
Task Parallel:
for (int i=0; i < n; i++) results[i] = compute(i);
Parallel.For(0, n, I => results[i] = compute(i));

Data Parallel:
(IEnumerable<T> objects. Use of foreach and ForEach
keywords):
foreach(testClass t in data) compute (t);
Parallel.ForEach(data, delegate(testClass t)
{compute(c);});
```

Note that the `For` and `ForEach` methods take a lambda expression for definition of the function to apply in parallel. The `Invoke` static method can be used to run statements in a block of statement in parallel. The `Task` class can be used to create and operate on a task; it is similar to

Figure 5: Three physical computers hosting multiple processes with distinct ranks



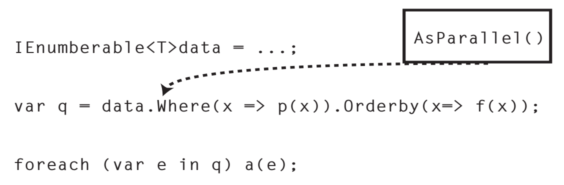
what `ThreadPool` provides. A delegate is queued for execution. The `Task` is simpler to use and offers more functionality. Methods for wait, status check of tasks are present.

Illustrations are given in C#, but TPL is available also in Visual Basic 2008 and F#, which is a functional programming language (see [MacLennan, 1990] in References).

The `Future<T>` class derives from `Task`. This has a value associated with it that is the result of the asynchronous execution of the `System.Func<T>` type instance that is provided as parameter. The value can be accessed from the `Future` instance and can be used to wait until it is available. `Future` provides a mechanism to define a data-driven or data-flow computing architecture.[25]

High-level constructs — such as thread-safe collections, more sophisticated locking primitives, data structures for work exchange, types to control how variables are productive, and the repertoire of powerful synchronization primitives — include `CountEvent`, `LazyInit<T>`, `ManualResetEventSlim`, `SemaphoreSlim`, `SpinLock`, `SpinWait`, `WriteOnce<T>`, and the `Collections.BlockingCollection<T>`, `Collections.CouncurrentQueue<T>`, and `Collections.ConcurrentStack<T>`.

Parallel LINQ (PLINQ) is a component of PFX. The data parallel nature ensures that programs can scale efficiently as data increases. PLINQ offers an incremental way of taking advantage of parallelism for existing solutions to existing problems. To use PLINQ, you will have to wrap the data source in an `IParallelEnumerable<T>` with a call to the `System.Linq.ParallelEnumerable.AsParallel` extension method (`IParallelEnumerable` is an extension of `IEnumerable<T>`).



```
IEnumberable<T>data = ...;                AsParallel()

var q = data.Where(x => p(x)).Orderby(x=> f(x));

foreach (var e in q) a(e);
```

The `var q` defines the query, and `foreach` actually executes it over the data source `q`. This declarative query helps the PLINQ to delay determination of optimal resource uses, such as the number of processors to run the query until it is actually executed in the `foreach` with action `a`. It will arrange for parts of the query to run on the available processors through the hidden use of multiple threads.

MPI.NET

MPI.NET[26] is an efficient interface for using the native MPI library from C#. It simplifies interface and extends MPI by taking advantage of features of C# and the managed-unmanaged interoperability mechanism. Several innovative measures have been taken to reduce abstraction penalties in performance. For example, generic versions of point-to-point `Send` allow the use of any user-defined types for transmission. In general, this is extended to all types of communication operations. For more information, see [MPI.NET, 2008] in References

Programming Environment and Tools

In multi-core systems, operating systems are revamped to include various paradigms to ensure better resource utilization. In clusters, many of these supports are integrated development and deployment services and tools, including service-oriented job scheduling.[27]

"Parallel programming is an extension of sequential programming. A parallel algorithm is given by algorithms of the constituting sequential program and a pattern to combine them. The programming model for analyzing sequential programming is extended to the shared-memory model and the distributed-memory model."

Tools provide debugging support at both source levels, such as in Visual Studio. Visual Studio also provides a parallel debugger extension.[28] Trace logs can help diagnose these problems.[29] Portland Group has a debugger for Windows cluster.[30] Other providers include TotalView.[31]

The most common process is to profile the behavior via tracing tools, followed by analysis and tuning. MPI was developed with tracing in mind. MPE, which is trace library, is available with MPI distribution [32]; also, it is shipped with Windows HPC Cluster.[33] The trace may be viewed by using viewing tools, such as Jumpshot.[34] Other tools include Intel Trace Analyzer and Collector (Vampir), [35] MPICL + ParaGraph,[36] and Epilog and KOJAK.[37]

Conclusion

Parallel programming is an extension of sequential programming. A parallel algorithm is given by algorithms of the constituting sequential program and a pattern to combine them. The programming model for analyzing sequential programming is extended to the shared-memory model and the distributed-memory model. Various processor and cluster architectures that support parallel computing are variations of these two models.

Correctness of parallel programs can be established via correctness of the sequential programs and the pattern of combination of these pieces. Performance of parallel programs depends on algorithm, implementation details, and target-computer architecture. Parallel computers range from multi-core processors to clusters, computational grids, and cloud computers.

All of the methodologies for developing parallel programs can be put into an integrated framework. Development focuses on algorithm, languages, and how the program is deployed on the parallel computer.

Endnotes

[1] Moore, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.75.4155

[2] Copeland, Michael. "Faster Chips, Slower Computers?" *Fortune Magazine*, August 14, 2008, http://money.cnn.com/2008/08/13/technology/microchips_copeland.fortune/index.htm?postversion=2008081405

[3] TOP500 Supercomputing Sites, http://www.top500.org/

[4] American National Standards Institute.

[5] Myrinet Overview, http://www.myri.com/myrinet/overview/

[6] InfiniBand, http://en.wikipedia.org/wiki/InfiniBand

[7] TOP500.org uses LINPACK Benchmarks to rank performance of parallel computers, including clusters worldwide, http://www.top500.org/

[8] Dryad Overview, http://research.microsoft.com/research/sv/dryad/

[9] Google MapReduce, http://labs.google.com/papers/mapreduce.html

[10] Microsoft adCenter, http://advertising.microsoft.com/search-advertising

[11] DryadLINQ Overview, http://research.microsoft.com/research/sv/DryadLINQ/

[12] Microsoft Robotics Studio (MSRS), CCR, and DSS, http://msdn.microsoft.com/en-us/library/bb648760.aspx

[13] Microsoft Open Specification Promise, http://www.microsoft.com/interop/osp/default.mspx

[14] Microsoft Windows Azure, http://msdn.microsoft.com/en-us/azure/default.asp

[15] Varia, Jinesh. "Cloud Architectures." Amazon Web Services, July 2008.
[16] Van Ingen, Catharine, and Jay Gupchup. "Enabling Eco-Science Analysis with MatLab and DataCubes in the Cloud." Microsoft Research Publications, May 2008, http://research.microsoft.com/research/pubs/view.aspx?type=Technical Report&id=1488
[17] High Performance Fortran is based on Fortran 90 and was developed in 1993 at Rice University.
[18] C* is a parallel version of C that was developed in 1992 for The Connection Machine, http://www.cs.sunysb.edu/~csilva/papers/rpe/node5.html
[19] Microsoft Visual C++ has an OpenMP extension; Microsoft Windows HPC Server 2008 and Microsoft Windows Compute Cluster Server 2003 (CCS) SDK has an MPI (MPICH2). Visual Studio Integrated Development Environment can be integrated with both OpenMP with C++ and Microsoft MPI.
[20] MPI-2 is the standard for an MPI, http://www.mpi-forum.org/docs/docs.html
[21] Microsoft .NET Framework, http://msdn.microsoft.com/en-us/netframework/default.aspx
[22] MSDN Parallel-Computing Forum, http://forums.microsoft.com/MSDN/default.aspx?ForumGroupID=551&SiteID=1, and Parallel-Computing Developer Center, http://msdn.microsoft.com/en-us/concurrency/default.aspx.
[23] A delegate is a pointer to functions that preserves type and signature information.
[24] Anonymous methods were introduced in C# 2.0 that allow declaring inline methods with a delegate function. In C# 3.0, the lambda expression is available for the same purpose, but more elegantly.
[25] Read about Jack Dennis and the papers on data-flow architecture at http://en.wikipedia.org/wiki/Jack_Dennis.
[26] MPI.NET at PPoPP Home Page, http://research.ihost.com/ppopp08/
[27] New in Windows HPC Server 2008, http://www.windowshpc.net/Pages/Default.aspx.
[28] Read about debugging MPI on Windows cluster using Visual Studio at Windows HPC home page, http://www.microsoft.com/hpc/
[29] MPI trace logs in WHPC.
[30] Read about PGDBG and PGPROF at The Portland Group Web site, http://www.pgroup.com/resources/mpitools.htm
[31] TotalView Technologies, http://www.totalviewtech.com/index.htm
[32] MPI Parallel Environment (MPE), http://www-unix.mcs.anl.gov/perfvis/download/index.htm), from Argonne National Lab.
[33] Windows HPC Cluster is the product Windows CCS 2003 or Windows HPC Server 2008 edition.
[34] Jumpshot, http://www-unix.mcs.anl.gov/perfvis/download/index.htm
[35] Intel Trace Analyzer and Collector, http://www.intel.com/cd/software/products/asmo-na/eng/306321.htm
[36] MPICL (Instrumentation Library for MPI), http://www.csm.ornl.gov/picl/, and ParaGraph (Performance Visualization Tool for MPI), http://www.csar.uiuc.edu/software/paragraph/
[37] Epilog, http://icl.cs.utk.edu/projectsfiles/kojak/software/kojak/win_epilog.zip, and KOJAK, http://icl.cs.utk.edu/kojak/index.html.php

References

[Akl, 1989] Akl, Selim G. *The Design and Analysis of Parallel Algorithms*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[Dongarra, 2003] Dongarra, Jack J., et al. *Sourcebook of Parallel Computing*. San Francisco: Morgan Kaufman Publishers, 2003.

[Foster, 1995] Foster, Ian. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Reading, MA: Addison-Wesley, 1995.

[Gropp, 1999] Gropp, William, et al. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA: MIT Press, 1999.

[Lea, 1999] Lea, Doug. *Concurrent Programming in Java: Design Principles and Patterns*. Second edition. Reading, MA: Addison-Wesley, 1999.

[Leighton, 1992] Leighton, Frank Thomson. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Mateo, CA: M. Kaufmann Publishers, 1992.

[MacLennan, 1990] MacLennan, Bruce J. *Functional Programming: Practice and Theory*. Reading, MA: Addison-Wesley, 1990.

[Mattson, 2005] Mattson, Timothy G., et al. *Patterns for Parallel Programming*. Boston: Addison-Wesley, 2005.

[Miller, 2005] Miller, Russ, et al. *Algorithms Sequential and Parallel: A Unified Approach*. Second edition. Hingham, MA: Charles River Media, 2005.

[MPI.NET, 2008] Gregor, Douglas, and Andrew Lumsdaine. "Design and Implementation of a High-Performance MPI for C# and the Common Language Infrastructure." MPI.NET Publications. Proceedings of 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Salt Lake City, February 2008.

[Quinn, 2004] Quinn, Michael J. *Parallel Programming in C with MPI and OpenMP*. Dubuque, IA: McGraw-Hill, 2004.

[Sterling, 2002] Sterling, Thomas L. *Beowulf Cluster Computing with Windows*. Cambridge, MA: MIT Press, 2002.

About the Author

Ranjan Sen is Senior Solutions Architect at Microsoft. He joined Microsoft in 2001, having worked in high performance computing, parallel processing and architecture since 1981. Ranjan earned his Ph.D. in Computer Science at Calcutta University in 1978, and has served on the faculty of several universities since 1979, including Indian Institute of Technology, Rutgers University, and Hampton University. Ranjan specializes in graph theoretic modeling for algorithm to architecture mapping and has published extensively on the subject.

Follow up on this topic

- *MSDN Magazine* October 2008 on Parallel Computing: http://msdn.microsoft.com/en-us/magazine/cc992993.aspx
- High Performance Computing in the Real World: http://technet.microsoft.com/en-us/magazine/2009.04.hpc.aspx

.NET Parallel Extensions: http://msdn.microsoft.com/en-us/concurrency/default.aspx
- Windows HPC Server 2008: http://www.microsoft.com/hpc/en/us/default.aspx

Enterprise Social Computing

by Kendrick Efta

Summary

Given the key business needs and application trends common within enterprises today, Microsoft Office SharePoint Server 2007 allows enterprises to start small and reproduce the effects of consumer-focused social computing technologies within the firewall. In addition to supporting smaller-scale "weak" social computing, MOSS allows users to scale-up to "strong" social computing scenarios that connect larger numbers of users who are widely distributed, and that generate collective intelligence within organizations. As organizations begin to see success stories and case studies take shape, they can begin to plan social computing investments that involve customers, partners, and external communities. Organizations can also seek to leverage the relationship between business decision makers and IT to adopt or develop richer sets of tools on the SharePoint platform that help to enable social computing both within and outside of the firewall.

Introduction

The Web 2.0 phase of the Internet characterizes a fundamental transition from an ecosystem of static, generated Web content, to an ecosystem of applications and services that have become vibrant, thriving communities driven by user participation and promotion. These new services and applications provide rich, collaborative, social experiences to users, helping to foster collective intelligence — the "wisdom of the crowds" — and evolving the way users solve problems, shape opinions and perceptions, and interact with communities. These user experiences and the effects of these social computing capabilities have become the hallmark of Web 2.0 technologies.

Although this shift has had the most dramatic impact on consumer Web applications and services, many businesses and enterprises are still grappling with how to best reproduce the effects to the Web 2.0 consumer and social Web within the four walls of their organizations.

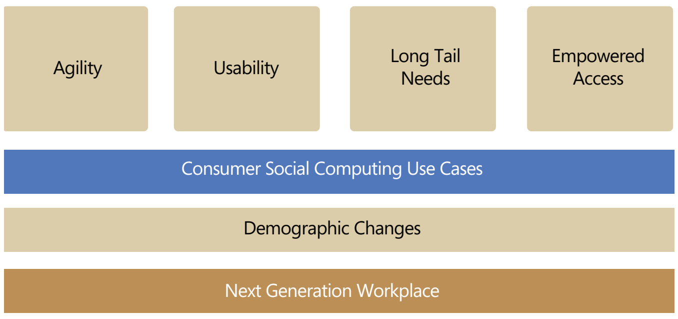
Enterprises have a distinct set of needs and challenges that must be considered and addressed in order for any deployment of social applications and services to be successful. Additionally, the concept of "weak" versus "strong" social software environments — especially as they relate to social structures and norms within an enterprise — can help enterprises plan for the evolution of their investments in social computing, ensuring business value as the enterprise grows and evolves. Finally, investments in platforms like Microsoft Office SharePoint Server 2007 allow enterprises to "start small," trying out aspects of the platform to determine what best meets their needs, and then "scaling up" the proven services and applications to meet the strategic needs of the enterprise.

Enterprise Business Needs

Many enterprises exhibit a common pattern of business needs worth considering when evaluating enterprise platforms and social software. These needs manifest themselves as "Application Megatrends"[1] (Figure 1):

- *Agility:* applications that can be composed in hours or days (as opposed to weeks or months) to meet immediate business needs. The business needs for these applications are often identified and managed at a tactical level, and are often referred to as "provisional applications" — they may be discarded or retired once the business need no longer exists. Conversely, these provisional applications may serve as proofs of concept to help demonstrate the return on investment for a larger, more comprehensive solution.
- *Usability:* functionality and information delivered in interfaces with which the users are already familiar. For instance, if users spend a larger percentage of their work day using Microsoft Outlook 2007, users should be able to access key functionality and information from within the Outlook interface. Additionally, there is an increasing need for ubiquitous channels through which social computing information can be relayed. These may take the form of mobile interfaces, Desktop Internet Applications, or Rich Internet Applications (like Silverlight or Flash).
- *"Long Tail" business needs:* many small companies, or small teams within large organizations, cannot afford to build custom applications and still meet highly individuated business needs. IT organizations within these enterprises are able to address large-scale, enterprise-wide needs, but typically do not have time or budget to implement smaller projects with "niche" requirements.

Figure 1: Business Needs Influenced by Demographic and Workplace Changes



- *Empowered access:* software capabilities to enable all users, not just management or executives, to make better decisions. Empowered access may represent itself as the democratization of information within an enterprise, or simply taking advantage of a platform's functionality to enable smart decisions and actions regardless of the role of the user.

Additionally, there are "horizontal" social changes that may affect how enterprises perceive and respond to the business needs above.

- *Worker demographics:* The 81 million children (Tapscott, 2008, p. 16) born from 1977 through 1997 (known as "Generation Y" or "Millennials") have begun to enter the workforce in significant numbers. These workers have been raised with technology, and many expect to adopt and find innovative uses for technology and social computing in their work just as they do in their personal lives. Conversely, the 77 million "Baby Boom" workers born between 1946 and 1964 (Tapscott, 2008, p. 16) are starting to near retirement age and exit the workforce. Although there is a percep tion that these workers do not widely embrace technology, there is agreement that they have tremendous knowledge and experience and that social computing technology may be an excellent way to record and share their intellectual capital prior to their retirement. [2]
- *Next-generation workplaces:* Enterprises are increasingly embracing arrangements that break with more traditional workplace infrastructure and expectations. These include telecommuting, collaboration with remote colleagues across organizational hierarchies, virtual team-oriented structures, changes in how relationships with customers and partners are managed, and an increasing presence of project freelancers and consultants participating in the "Gig Economy" as economic conditions remain unstable.
- *Consumer-based social computing use cases:* Many workers are active and effective participants in social computing, an have developed the expectations that consumer-based tools can and should be used within their companies. Recently, there have been widely-noted success stories[3] of social-computing deployments within enterprises based on consumer models. These success stories have fueled the conversation about how social computing can be adopted within the four walls of an enterprise, reproducing the positive effects of the consumer-based technology and helping to fulfill company business needs.

The Enterprise Challenge

Although many workers are ready to adopt the consumer social-computing technology to enhance productivity and enable collective intelligence, a key challenge resides with the IT departments who are responsible for the deployment and maintenance of these tools. IT departments need "Enterprise Ready" tools that are secure, controlled, compliant, and manageable. Their concerns about these areas of governance are very real. There are numerous examples of malware infecting systems,[4] of intellectual property being leaked, and of lost productivity. Additionally, many of the consumer-oriented social-computing services do not currently have plans to accommodate enterprise-specific uses of their services.[5] Thus, there are several key considerations that enterprises should take into account when trying to reproduce the effects of the consumer Web within the four walls:

- Successful adoption and deployment of social computing solutions is largely focused on people — their relationships, how they work together, how they communicate, and the business processes they use. The technology itself is typically a much smaller part of the solution. Understanding the needs of the people in the enterprise will greatly increase the likelihood that the technology meets those needs, and that the users understand and are bought into the benefits of the technology.
- Investigate enterprise platforms that support both "weak" as well as "strong" social computing scenarios. A recent Gartner report[6] draws the distinction between "weak" and "strong" social computing. Although every collaboration technology is social in some way, "weak" social software can supplement the preexisting connections and social interaction between individuals (e-mail, document collaboration, instant messaging). "Strong" social software encourages interaction between larger numbers of individuals with looser social connections (Facebook, Digg, LinkedIn, SlideShare, Twitter). This distinction allows organizations to identify how their business needs fit on the continuum between weak and strong social software, framing a discussion with the users in their organizations about these needs, and enabling them to plan their investment in social-computing technology.
- Don't "boil the ocean." Start now, leveraging the platform in place (i.e. "weak" social software), and begin to prove how social computing (i.e. "strong" social software) can help to meet business needs.
- Plan longer-term investments in social computing to engage customers, partners, and other consumer-oriented communities and services.
- Evolve approaches and capabilities for governance and compliance as needed–governance needs will change over time.

Solutions Framework

Microsoft Office SharePoint Server (MOSS) 2007 provides a platform and solutions framework for social-computing business needs outlined above. Although a consumer-focused solutions framework will necessarily include applications, services, and users that site outside of the firewall, for our purposes, we'll consider only the framework for what will be supported within the four walls of an enterprise. Here's a quick tour (Figure 2):

Figure 2: Social Computing Solutions Framework

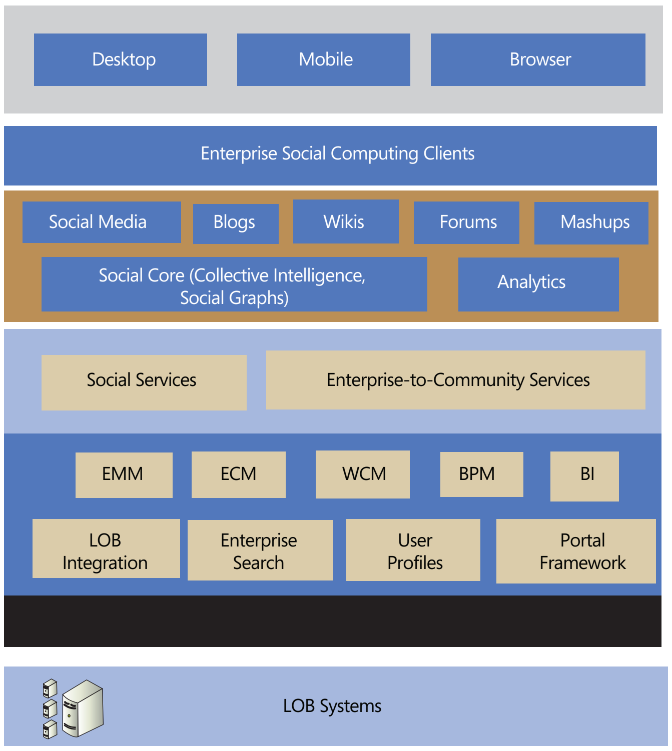


Figure 3: Collaborative Records Management: Home Page

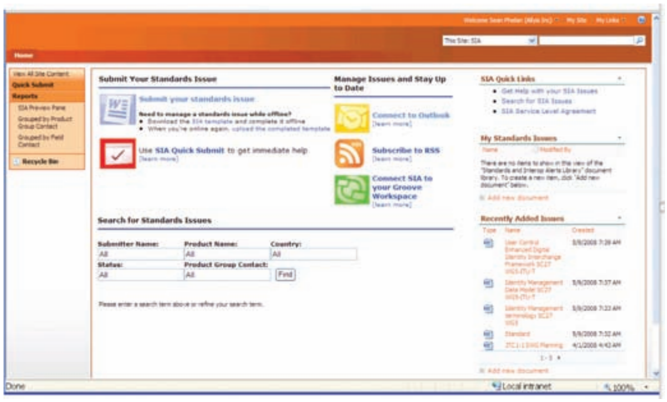
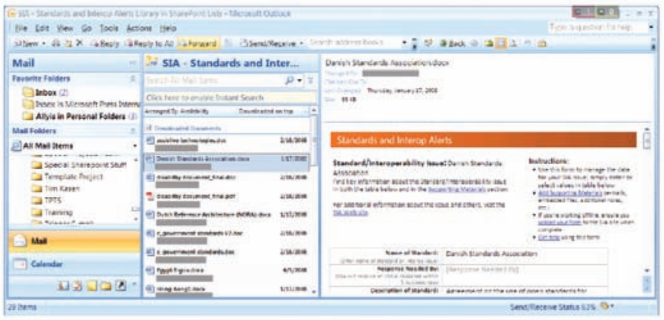

Submit Your Standards Issue
Manage Issues and Stay Up to Date
SIA Quick Links
Use SIA Quick Submit to get immediate help
Connect to Outlook
Subscribe to RSS
Connect SIA to your Groove Workspace
My Standards Issues
Recently Added Issues
Search for Standards Issues
Submitter Name:
Product Name:
Country:
Status:
Product Group Contact:
Find

Figure 4: Collaborative Records Management: Connecting Document Library to Outlook 2007



Line of Business (LOB) Systems
It may be necessary to provide data from an enterprise's line of business systems to an enterprise platform like MOSS 2007. These data may include Customer Relationship Management information, fiscal and accounting data, sales information, and so on. Key business processes and workflows architected on the SharePoint platform many depend upon these LOB data to successfully execute.

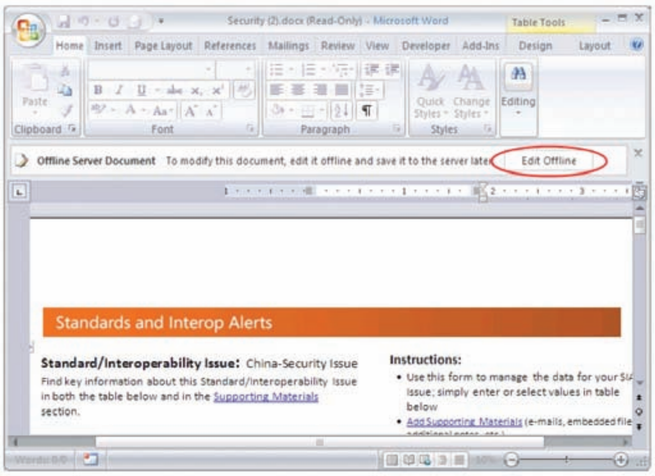
Enterprise Productivity Services
These services include many of the features common to both weak and strong social software, and demonstrate the key benefits of selecting a strong enterprise platform that can enable organizations to materialize both short-term and longer-term investments in the software. MOSS 2007 services include:

- *Enterprise Metadata Management (EMM):* central management and maintenance of corporate metadata to be leveraged in various features of the platform.
- *Enterprise Content Management (ECM):* management of content and assets; integration of EMM to describe and catalog content; architecture of compliance and retention policies; integration of content assets and metadata into productivity applications like the Microsoft Office 2007 suite.
- *Web Content Management (WCM):* management of Web-based content; management of reusable and localized content; content staging and replication; document conversion into Web-based content; creation and maintenance of key UI and branding assets.
- *Business Process Management (BPM):* management of business processes via workflow automation via SharePoint Designer 2007 workflows and custom-developed Windows Workflow Foundation (WWF) solutions.
- *Business Intelligence (BI):* Excel Services, Excel Web Access, Key Performance Indicators (KPIs).
- *LOB Integration:* Integration with LOB systems via Business Data Catalog (BDC), display of business intelligence data from Excel Services.
- *Enterprise search:* search capabilities for content, LOB data, profiles, and search content stored in off-platform file shares and databases.
- *User profiles:* management of profile data, social graph and relationships, personally-managed assets associated with the profile, unified communications, presence, and Active Directory integration.
- *Portal framework:* core services designed to orchestrate and provide user interfaces for the above features; additionally, the framework provides features like e-mail Alerts, RSS feeds, and connectivity to Microsoft Office 2007 products like Outlook 2007 and Excel 2007. Additionally, it provides various authentication, authorization, and permissions models.

Enterprise Social Computing Features
MOSS 2007 provides many out-of-the-box social computing features that build upon core Productivity Services, enabling enterprises to begin using these features with relatively small investments in planning:

- *Blogs:* out-of-the-box features allow users to create their own blogs; post entries via both Web interfaces and tools like Microsoft Word 2007 and Windows Live Writer; and manage categories and metadata. Additionally, users may comment on posts. Enhancements to SharePoint's blog functionality include the Enhanced Blog Edition of the Community Kit for SharePoint.
- *Wikis:* Wiki functionality that allows users to create rich stores of unstructured knowledge by quickly composing wiki pages, creating stub pages to indicate where additional content is needed, and to edit and version content over time. Enhancements to SharePoint's Wiki functionality include the Enhanced Wiki Edition of the Community Kit for SharePoint.
- *Forums and discussion boards:* features to allow users to post discussion topics and replies online; integration with Microsoft Exchange allows users to continue to use the e-mail discussion groups they may still be using while also saving copies of the discussion threads online so that they're available for indexing and can appear in enterprise Search results.
- *Social core:* MySite features allow users to create and maintain profiles as well as a social graph of colleagues and organizational hierarchies within

Figure 5: Collaborative Records Management: Editing a Document Offline



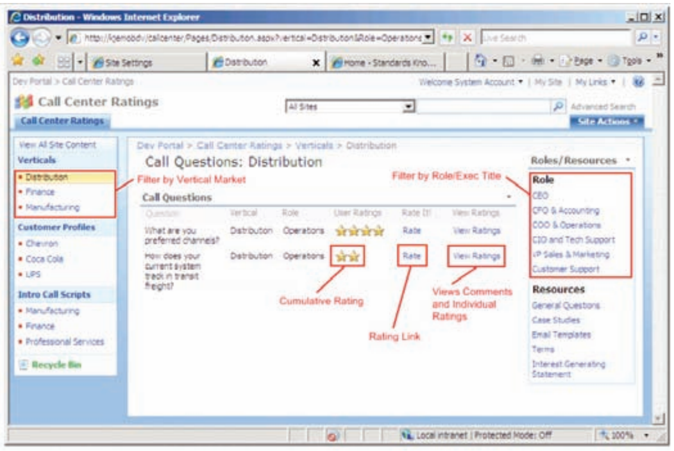
Offline Server Document To modify this document, edit it offline and save it to the server later.

Standards and Interop Alerts

Standard/Interoperability Issue: China-Security Issue
Find key information about this Standard/Interoperability Issue in both the table below and in the Supporting Materials section.

Instructions:
- Use this form to manage the data for your SIA Issue; simply enter or select values in table below
- Add Supporting Materials (e-mails, embedded file

Figure 6: Call Center Questions Filtered by Vertical and Role



Distribution - Windows Internet Explorer
Site Settings
Distribution
Home - Standards Kno...
Welcome System Account | My Site | My Links
Call Center Ratings
All Sites
Advanced Search
Site Actions
View All Site Content
Filter by Vertical Market
Filter by Role/Exec Title
Cumulative Rating
Rating Link
Views Comments and Individual Ratings
Recycle Bin
Local intranet | Protected Mode: Off
100%

Dev Portal > Call Center Ratings > Verticals > Distribution

Call Questions: Distribution

Verticals: Distribution, Finance, Manufacturing

Customer Profiles: Chevron, Coca Cola, UPS

Intro Call Scripts: Manufacturing, Finance, Professional Services

Call Questions

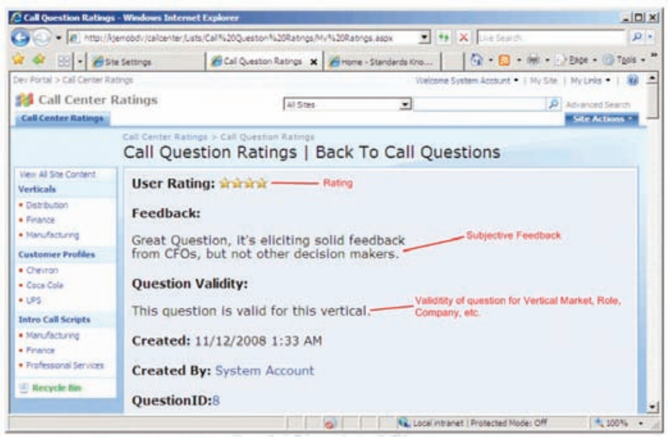
Question	Vertical	Role	User Ratings	Rate It!	View Ratings
What are you preferred channels?	Distribution	Operations	★★★★★	Rate	View Ratings
How does your current system track in transit freight?	Distribution	Operations	★★	Rate	View Ratings

Roles/Resources

Role: CEO, CFO & Accounting, COO & Operations, CIO and Tech Support, VP Sales & Marketing, Customer Support

Resources: General Questions, Case Studies, Email Templates, Terms, Interest Generating Statement

Figure 7: Call Center Rating Drill Down



Call Question Ratings - Windows Internet Explorer
Site Settings
Call Question Ratings
Home - Standards Kno...
Welcome System Account | My Site | My Links
Call Center Ratings
All Sites
Advanced Search
Site Actions
View All Site Content
Rating
Subjective Feedback
Validity of question for Vertical Market, Role, Company, etc.
Recycle Bin
Local intranet | Protected Mode: Off
100%

Call Center Ratings > Call Question Ratings

Call Question Ratings | Back To Call Questions

User Rating: ★★★★★

Feedback:

Great Question, it's eliciting solid feedback from CFOs, but not other decision makers.

Question Validity:

This question is valid for this vertical.

Created: 11/12/2008 1:33 AM

Created By: System Account

QuestionID:8

the enterprise; additionally, MOSS provides features for presence as well as basic visibility into activities of peers that are part of the social network.

- *Analytics:* MOSS provides out-of-box usage analytics as well as event logging.

Enterprise Social Computing Clients

Although MOSS is primarily accessible via Web Brower clients, there is rich integration with the Microsoft Office 2007 productivity suite and through mobile-enabled interfaces. MOSS also includes Web Services, which can be called from other applications to integrate, process, and display data useful to users. An excellent example of this is Vista Gadgets that display custom views of MOSS data.

Examples of Social Computing Within the Enterprise

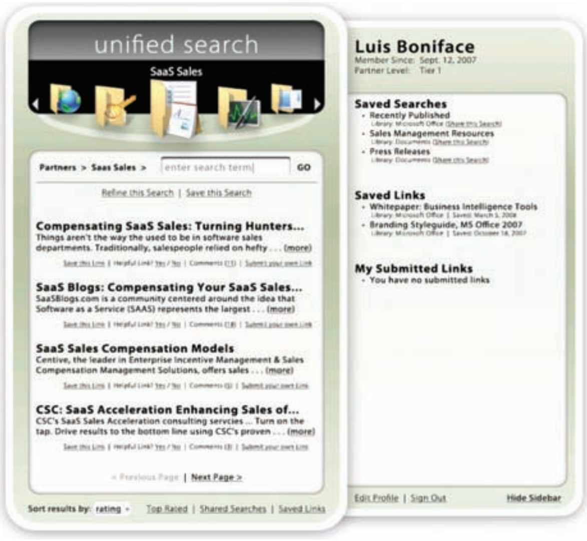
The following examples are intended to for both platform features and social computing features within an enterprise. The first two examples demonstrate solutions that fit more clearly into the "weak" social software experience, and the remaining examples are solutions falling squarely into the "strong" social software experience.

Collaborative Records Management

This solution shows how an organization of geographically distributed subject matter experts leveraged MOSS' collaboration features to better share large numbers of Briefing Documents among themselves (Figures 3, 4, and 5, page 26). Although this example doesn't feature the same functionally available on consumer sites, it demonstrates the strength of an enterprise platform like MOSS in supporting "weak" social scenarios where users already know each other and work closely together. Additionally, it is an excellent example of a "provisional application" being quickly composed by workers to meet a specialized business need. Specifically, the solution:

- Connects a SharePoint Document Library to Microsoft Outlook 2007 for offline browsing and editing.
- Provides customized views of the Document Library to enable quick scanning of information or views grouped by predefined criteria.
- Connects a Groove 2007 Workspace to the SharePoint Document Library to make relevant information available in an interface already adopted by users.
- Makes available key document metadata and descriptions via RSS feeds.

Figure 8: Social Search: Results Interface



unified search
SaaS Sales
GO
Refine this Search | Save this Search
Save this Link | Helpful Link? Yes / No | Submit your own Link
Previous Page | Next Page >
Sort results by: rating
Top Rated | Shared Searches | Saved Links
Edit Profile | Sign Out
Hide Sidebar

Partners > Saas Sales > enter search term

Compensating SaaS Sales: Turning Hunters...
Things aren't the way the used to be in software sales departments. Traditionally, salespeople relied on hefty . . . (more)

SaaS Blogs: Compensating Your SaaS Sales...
SaaSBlogs.com is a community centered around the idea that Software as a Service (SAAS) represents the largest . . . (more)

SaaS Sales Compensation Models
Centive, the leader in Enterprise Incentive Management & Sales Compensation Management Solutions, offers sales . . . (more)

CSC: SaaS Acceleration Enhancing Sales of...
CSC's SaaS Sales Acceleration consulting servcies ... Turn on the tap. Drive results to the bottom line using CSC's proven . . . (more)

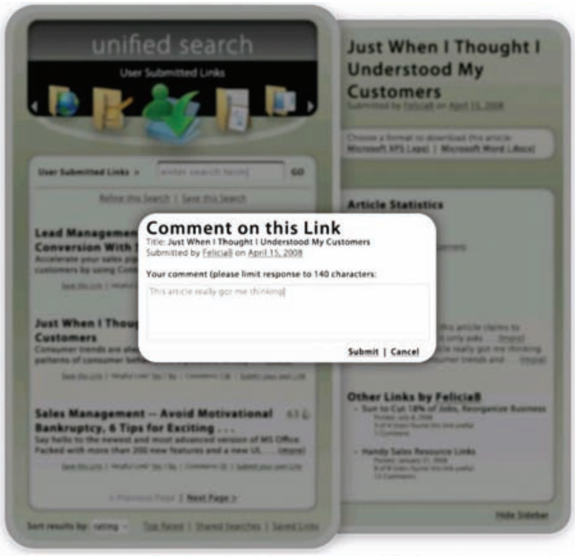
Luis Boniface
Member Since: Sept. 12, 2007
Partner Level: Tier 1

Saved Searches
- Recently Published
- Sales Management Resources
- Press Releases

Saved Links
- Whitepaper: Business Intelligence Tools
- Branding Styleguide, MS Office 2007

My Submitted Links
- You have no submitted links

Figure 9: Social Search: Commenting on a Search Result



Just When I Thought I Understood My Customers

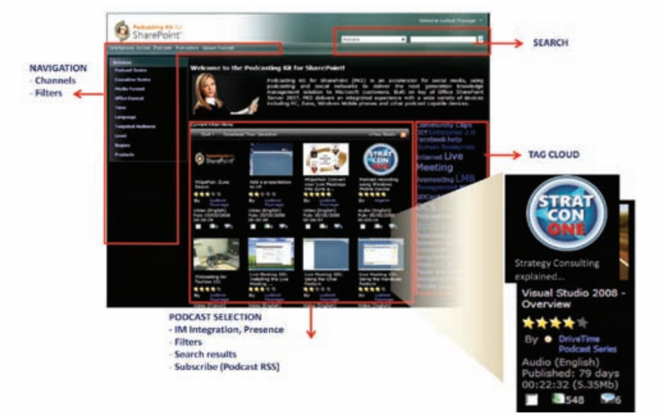
Comment on this Link
Title: Just When I Thought I Understood My Customers
Submitted by FeliciaB on April 15, 2008

Your comment (please limit response to 140 characters:

This article really got me thinking

Sales Management — Avoid Motivational Bankruptcy, 6 Tips for Exciting . . .

Figure 10: PKS: Home Page Features



- Leverages SharePoint Designer 2007 workflows to allow users to do a "Quick Submit" for a Brief without needing to completely populate the project brief template. Users are optionally able to use the mobile interface for the "Quick Submit" feature.

Call Center Questions Management

This solution was rolled out to a small team of inside sales professionals who call C-level executives as part of their sales activities (Figures 6 and 7, page 27). The team had been managing all call scripts in Microsoft Word and Excel, and was looking for a way to use MOSS 2007 to allow team members to rate the effectiveness of questions and also provide subjective comments. The display mechanism ensured that call questions rated as the most effective were "bubbled to the top" of the call questions list, increasing the likelihood of using questions more effectively.

Social Search: Silverlight Search Application

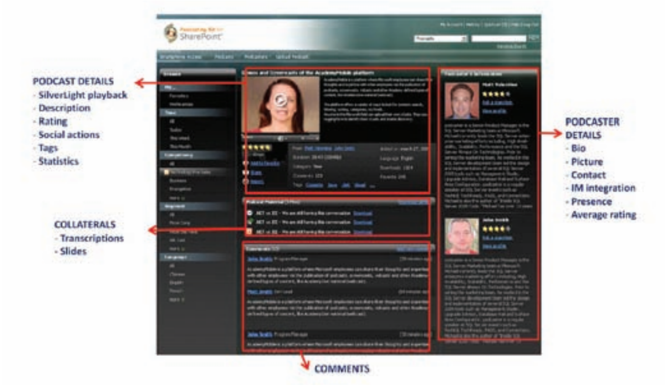
This social search application was designed as a prototype to demonstrate how MOSS' enterprise search capabilities could be enhanced (Figures 8 and 9, page 27). The MOSS Enterprise Search catalog was supplemented by other search sources, as well as by social search features:

- The application UI was built in Silverlight for its rich visualization and UI; a standard ASP.NET version was developed for users without the Silverlight plug-in.
- The solution consumed custom-developed Web Services that aggregated several search catalogs into a single, master search index.
- The solution introduced social features that are common in the consumer social search and bookmark space: Users could rate search results, comment on search results, save favorite searches and links, and submit their own links into a catalog of user-generated content.

Enterprise Social Media: Podcasting Kit for SharePoint

The Podcasting Kit for SharePoint (PKS) represents one of the best examples of a "strong" social-computing experience available on the SharePoint platform (Figures 10 and 11). Designed as a solutions accelerator, PKS enables enterprises to use podcasting and common social-computing features (ratings, comments, favorites, download statistics, user profiles, faceted browsing, mobile interfaces, taxonomy/tagging) to manage and aggregate knowledge within organizations. PKS is distributed under Public License with its source code and is free to use if you're already using MOSS 2007.

Figure 11: PKS Podcast Download and Details Page



References

[1] This treatment of enterprise needs is borrowed from Scott Jamison's presentation from the 2007 Strategic Architect Forum. These needs continue to be just as relevant, and are perhaps more immediate, given the current global economy.
[2] Salkowitz, 2008, pp. 85-88
[3] Appearing as presenters at the Enterprise 2.0 Conference in June 2008, Don Burke and Sean Dennehy from the CIA discussed the CIA's deployment and adoption of Intellipedia; Shawn Dahlen and Christopher Keohane demonstrated Lockheed Martin's social software platform.
[4] See Lt. General Jeffrey Sorenson's example of finding almost 30,000 instances of malware on the Army's host computers
[5] See Mark Zuckerberg's comment about Facebook's focus on enterprises. He states that although Facebook is not an enterprise application, someone will make a lot of money developing a social networking app for the enterprise.
[6] Nikos Drakos, 2008

Bibliography

Nikos Drakos, A. B. (2008). *Tutorial: Social Context, Not Technology, Definies Social Software*. Gartner.
Salkowitz, R. (2008). *Generation Blend*. Hoboken, New Jersey: John Wiley & Sons, Inc.
Tapscott, D. (2008). *Grown Up Digital*. New York, New York: McGraw Hill.

About the Author

Kendrick Efta is co-founder and Principal Consultant of Allyis. With more than a decade's experience conceptualizing, designing, and building enterprise solutions, Ken is responsible for driving Allyis' innovation and thought leadership efforts as well providing strategic insight and direction for clients. Prior to co-founding Allyis, he served as a technology consultant to a number of Seattle-area businesses. Ken was recognized by Western Washington University, along with Allyis co-founders Richard Law and Ethan Yarbrough, as its inaugural "Young Alumnus of the Year."

Follow up on this topic

- How to Get the Most Value from Social Computing for Business with Microsoft: http://www.microsoft.com/downloads/details.aspx?FamilyId=C5844123-7F31-49D4-811C-7B90E6217B1D&displaylang=en
- Social Computing in the Microsoft Platform: http://www.microsoft.com/sharepoint/capabilities/collaboration/social.mspx

Pragmatic Approach to Describing Solution Architectures

by Mike Walker

Summary
The return on investment of technical documentation is often not realized and documentation is frequently looked upon as a necessary evil. Creating the right architecture descriptions can help guide decision making at the various stages of the IT life cycle, however, there are limitations on formalized structures, information models, and intelligent tooling that takes the current architecture documentation to the next level of usefulness. In this article, we look at how we view, approach, and maintain architecture descriptions, and consider how this process can be improved.

Since the dawn of information technology (IT), engineers and architects have created reams of documents to describe their solutions. These have been published in three-ring binders and placed in obscure places on bookshelves, eventually to collect dust and be forgotten — something with which engineers and architects have had to deal as a fact of life.

As time has passed, documentation has evolved from a listing of detailed methods and procedures to the separation of multiple aspects of the solution from the detailed APIs and user-interface (UI) design to the architectural aspects. Integration with developmental and architectural processes furthered this activity and, eventually, became a required and common practice. By doing so, governance processes in many ways restricted forward progress. In many ways, governance has created both justified and unjustified restrictions to forward-moving progress on designs and coding, if a set of documentation had not been completed. This has led to frustration and productivity blockers.

Current architecture documentation does not live up to the promise of providing return on investment in the design process. Often, the design that is articulated in architecture documentation is not realized in the final implemented design that is hosted in a production environment.

This pervasive growth of documents shows how critical the information is to the support of the software-development life cycle (SDLC). We now see the same with the support of architecture efforts, too. There are a number of challenges that occur with this paradigm.

As Figure 1 shows, we are adding more and more documents to an already large portfolio of documents that must be completed. In the past, this was manageable; now, however, documents can be bloated, and there is a higher probability of information being duplicated. Often, this is a result of tightly coupling a document with a process step.

The goal is to solve the deficiencies and challenges with current architecture documentation while preserving the aspects that do work

"The most common and pervasive interface for creating architecture descriptions is Microsoft Office Word. Office Word is part of the Microsoft Office suite of productivity tools. With documentation, there are many other tools that play a role in the documentation processes."

and have been assimilated into the daily lives of architects. To do this, we will explore the following concepts:

- *Alleviating challenges with current documents* — Templates and new thought-provoking ideas will be introduced that challenge the existing ways of documenting architectures.
- *Living architecture designs* — Designs often are static and do not have a life outside of the development life cycle. We will introduce ways to change that.
- *Enhancing decision support* — Often, there are templates and checklists that give an architect a common way to think about problems. This is an accelerator to solving problems that have yet to be solved or identified.
- *Deriving to solutions* — Given how the human mind works, writing down a design to a specific problem in "black and white" often shows gaps in current thinking.
- *Common means of collaboration* — These provide a working information store to share and collaborate on architectures with team members.
- *Supportability and maintainability* — Documents provide important information on how a system was built. This provides support personnel with vital information for solving postproduction issues. For architects, the understanding of current system architecture will allow them to build out a set of strategies for the enterprise.

Figure 1: Documentation, increasing at an accelerated rate

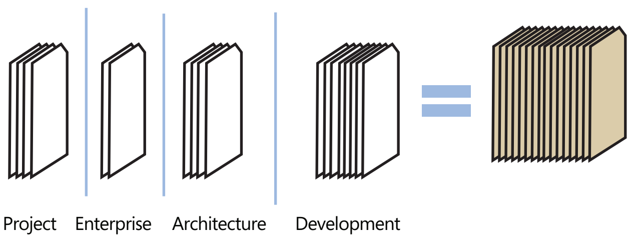
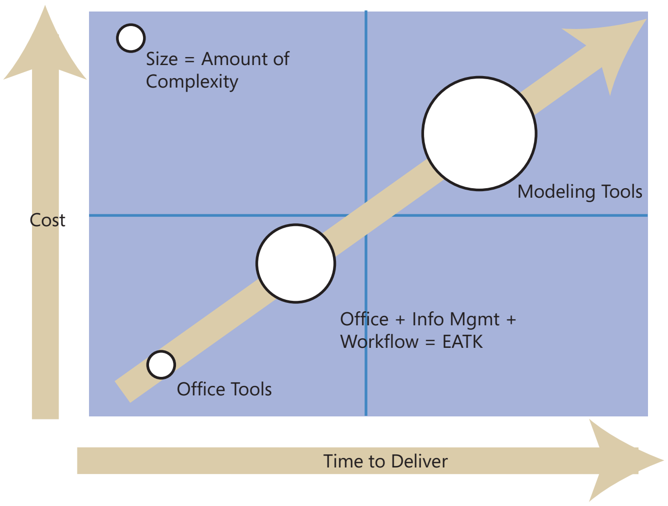


Figure 2: Microsoft Office Word reduces complexity and cost



Rethinking the Traditional Architecture Document

The most common and pervasive interface for creating architecture descriptions is Microsoft Office Word. Office Word is part of the Microsoft Office suite of productivity tools. With documentation, there are many other tools that play a role in the documentation processes.

As Figure 2 shows, Office provides a way to reduce complexity and costs significantly. The Office suite is easy to understand and most users already have them on their desktops. Augmenting the existing Office tools to make them fit the usage of architects is particularly ideal. It maintains a consumable amount of complexity while limiting the overall costs.

However, Office Word is not the only tool that is used in the architectural processes. There are many other tools that have roles to play in how we document our architectures. These tools include the following:

- Office Visio
- Office PowerPoint
- Office Excel
- Office SharePoint

It is important to understand the context in which these tools interact. Figure 3 shows an illustration of how other productivity tools play a part in architectural processes.

We can assert quickly that there is not just one tool that used in the documentation of architecture; myriad tools are used for collecting information from or publishing to. If we want to solve the challenges with the process, we should keep this in mind.

Optimizing Office Word to Describe Architectures

The underlying goal is to change the role of Office Word from simply a word processor to more of a UI for designing architectures. Applying structured UI concepts to Office Word provides many benefits to the architecture document, including the following:

- *Structured content* — Information can be better described in the document. We want to do this because of the challenges mentioned regarding how information does not integrate well with process or future design activities. One example is the process of importing a model into an architecture document. Often, we import a picture that represents a model of the specific viewpoint of an architecture. If we had an information model, we could specify that the model imported is indeed the logical model, instead of a generic image file.
- *Extensible* — With a little more structure, information has meaning and definition. This makes the information extensible to other processes downstream.
- *Consumable* — Ability to consume external content is also possible with a more structured interface. As an example, if you so choose, you could import external architecture information from other systems to automate your design efforts.

Through the use of the Office Word 2007 features that center on XML, we can truly start to extend this interface. Word does so by providing:

- *Embedding XML documents* — embed XML file into document for full data qualification.
- *Office Word XML format* — fully describes the formatting from the data through XML to have true separation of the formatting versus the information.
- *Content controls* — map most content controls to elements in XML data attached to a document.

Building the new UI for the architecture documents is easier in Office Word. A series of out-of-the-box functions are provided in the Office Word interface. For the architecture document, we will use the built-in tools that Office Word provides to create this new UI, which will enable us to describe our information in a meaningful way.

The Architecture of the System-Architecture Document

To change fundamentally how architects use a system-architecture document, we must look at what services Office Word provides to add additional capabilities that will automate and create rich metadata integration services.

For this solution, we will use a real world reference architecture called the Enterprise Architecture Toolkit (EATK). This toolkit will show how to alter Office Word from a word processor to a UI for describing architectures in a new way. The Microsoft Office environment provides rich extensibility that will allow developers to extend in a meaningful way. To do so, an architectural approach will have to be taken. By separating out layers and capabilities, approaches and technologies can be identified to derive to the right solution.

Figure 3: Other productivity tools playing a role in the documentation process

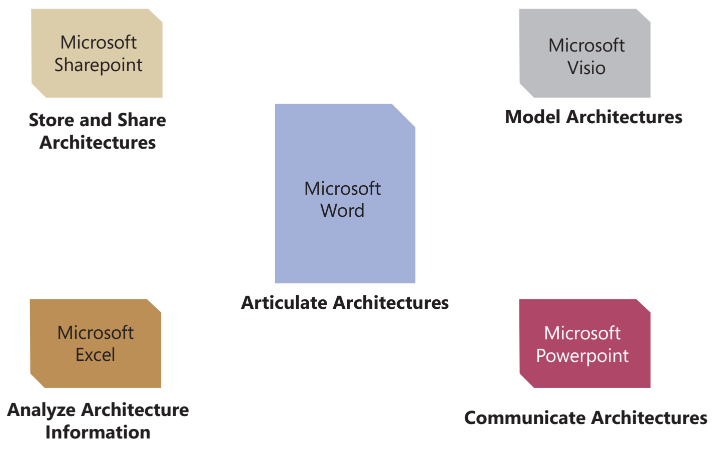
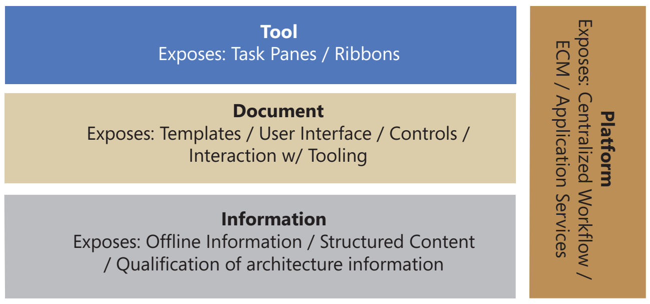


Figure 4: Architectural layers


Tool
Exposes: Task Panes / Ribbons
Document
Exposes: Templates / User Interface / Controls / Interaction w/ Tooling
Information
Exposes: Offline Information / Structured Content / Qualification of architecture information
Platform
Exposes: Centralized Workflow / ECM / Application Services

As Figure 4 shows, there are four architectural layers that expose discrete services for this solution. These include the following:

- *Platform* — This is what the solution-architecture document and the integration services connect to. The platform components integrate with Office SharePoint Application Server environment for rich line-of-business (LOB) integration.
- *Tool* — The mechanism of Office Word is referred to here. The tool provides extensibility into the UI that provides Microsoft Office ribbons to execute tasks easily with a level of context and Microsoft Office task panes that extend the UI with additional entry or listings of information.
- *Document* — The document provides the way in which a user can enter architecture descriptions. This is different in the EATK, as the document acts as the glue between the tool and the information itself. This is accomplished through an architecture-document template and the use of Office Word custom controls.
- *Information* — The information in the document is managed completely different from a traditional document. All of the information that is typed into the document is linked back to an XML node behind the scenes. This fully qualifies what is typed. Not only is the information rich, but it is extensible.

"To change fundamentally how architects use a system-architecture document, we must look at what services Office Word provides to add additional capabilities that will automate and create rich metadata integration services."

Platform Architecture

A great deal of work has been done in building componentized add-ins at the tooling level in Office Word, but this is not enough to change fundamentally the architecture document into a tool for designing architectures. The components that are applied to Office Word build upon a larger architecture canvas. The EATK provides a server-side Architecture Portal and Architecture Meta-Data Repository (AMR). This architecture interacts with the document-level activities when we create an architecture design.

Figure 5 shows a logical representation of the architecture that the EATK provides. It leverages a series of Microsoft-platform technologies, which include the following:

- *Microsoft Office SharePoint (MOSS) 2007* — Used as an Architecture Portal, Document Management Services, and Workflow
- *Windows Workflow Foundation (WF)* — The hosted workflow on the portal that interacts with the desktop application and add-ins
- *Microsoft Project Server* — Can be used as the platform for interacting with project and portfolio data
- *Microsoft SQL Server* — Used as the database platform for the AMR that the Office Word add-ins Patterns Browser and Architect Lookup will use to get their information
- *Microsoft IIS 7.0* — Used as the hosting platform for the AMR Services layer

Figure 5: Architecture behind EATK system-architecture document

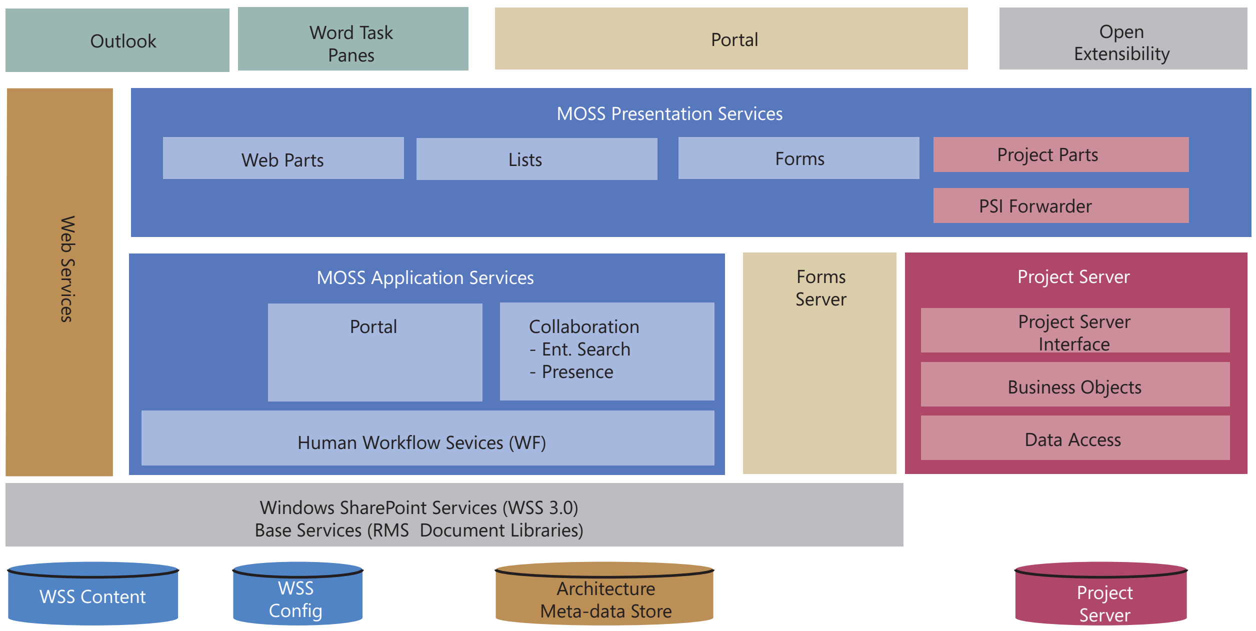
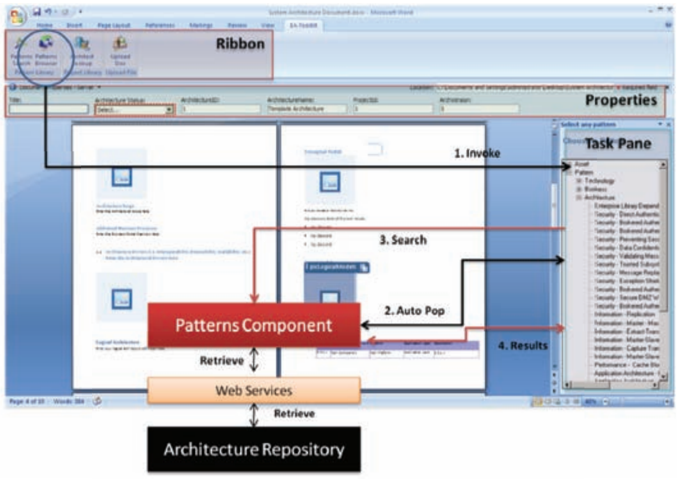


Figure 6: Components of Office Word interface



The server components in the EATK play a part in how we change the role of an architecture document. All of the components on the server leverage platform capabilities of Office SharePoint, but change the context in which they can be used. The context is the architecture-development process. In doing so, we can take generic capabilities such as Enterprise Content Management (ECM) to automate how information is audited and versioned.

The following are new interfaces and EATK components that were developed specifically for the architectural process and to interact directly with the system-architecture document:

- *AMR Web Services* — Services layer that allows for programmatic interaction with the AMR. It provides extensibility into not only the AMR, but also other related services such as PPM or Application Portfolio Management (APM).
- *AMR Data Services* — The base information services that delivers information such as patterns and existing IT assets to the Office Word task panes.
- *Document-Management Services* — An Office SharePoint–based set of services that are used to manage documents. It comprises functionality such as check-in, auditing, versioning, integrating with workflow, security, and archiving.
- *Workflow Services* — WF is used as the base of the workflow capabilities. Also, it is hosted on the server, which allows the architecting workflows that are applicable to the entire enterprise, instead of to just one architect.

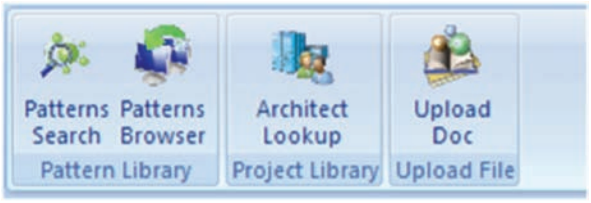
Tool Architecture

This aspect of the solution is key to bridging the system-architecture document to both the platform for LOB integration and the document itself, which will be the interface in which architects will describe their solutions. All of the application logic is encapsulated in this layer. Because this is the layer in which code is developed, this solution is dependent on Office Word API and other standard integration technologies.

Figure 6 shows the extended capabilities of Office Word. We have extended the following aspects of Office Word:

- *Ribbons* — We use this functionality as a launch pad for downstream activities, workflow triggering, collaboration, and information retrieval.
- *Task panes* — We can use this functionality for various aspects in which we want to create other interfaces from within Office Word.
- *Properties* — Assigning metadata to a document

Figure 7: Ribbon components of system-architecture document



System-Architecture Document Ribbons

Using the ribbon for the system-architecture document, we will have a series of functions that relate directly to architectural processes and information.

As Figure 7 shows, there are four ribbon components in the EATK:

- *Patterns Search* — It displays patterns and existing IT assets that will help solve specific architecture-design questions and challenges.
- *Patterns Browser* — It allows for surfacing the right patterns in a more intuitive way for solving a business problem.
- *Architect Lookup* — Looking up an architect who is assigned to a project is streamlined with Architect Lookup by integrating collaboration via office communication server (ocs)
- *Upload Doc* — Automatically have a way to integrate architecture information into hosted workflow and consumed into a metadata repository.

System-Architecture Document Task Panes

Task panes can be very useful to architects, as they can surface meaningful information at a moment's notice with a click of a ribbon component. This information is not only visible, but also interactive.

Tangible ways in which these task panes empower IT architects are the following:

- *Systematic reuse* — Many of the architectures that we build have repeatable patterns built within. By surfacing patterns in a composite

Figure 8: Interacting with elements in document

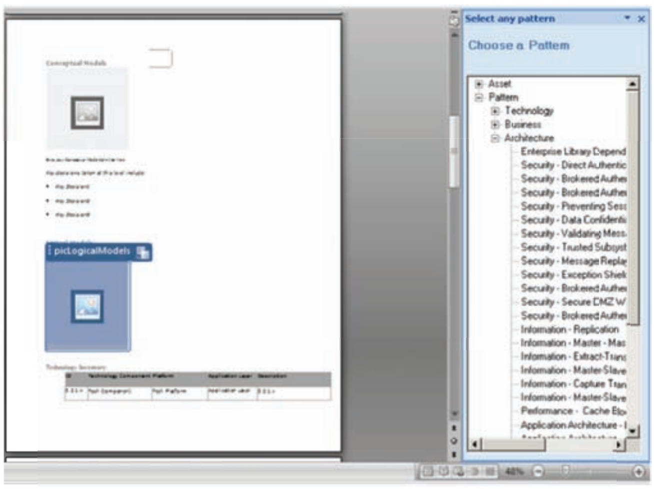
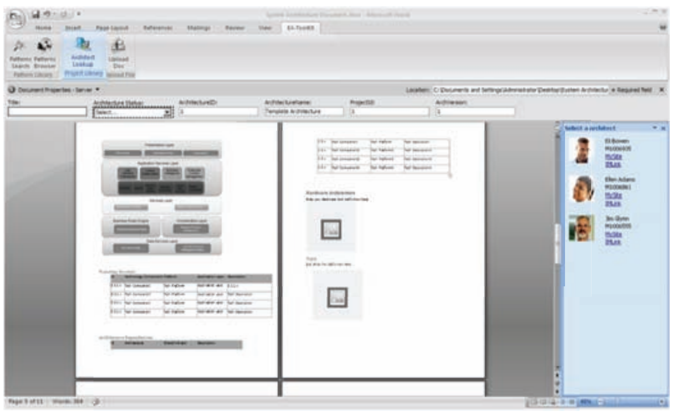

Select any pattern
Choose a Pattern
Asset
Pattern
Technology
Business
Architecture
Enterprise Library Depend
Security - Direct Authentic
Security - Brokered Authen
Security - Brokered Authen
Security - Preventing Sess
Security - Data Confidenti
Security - Validating Mess
Security - Trusted Subsyst
Security - Message Replay
Security - Exception Shiel
Security - Brokered Authen
Security - Secure DMZ W
Security - Brokered Authen
Information - Replication
Information - Master - Mas
Information - Extract-Trans
Information - Master-Slave
Information - Capture Tran
Information - Master-Slave
Performance - Cache Exp
Application Architecture - I
picLogicalModels

Figure 9: Architect Lookup task pane



manner, we can reuse the aspects and views of architectures in a systematic way.

- *Decision support* — By reviewing existing architectures, we can review how solutions were developed based on a set of drivers. Not only can you review, but also you can contrast decisions on these architectures with the challenges of the current architecture efforts.
- *Automation* — Not only can you view the patterns and assets in the task pane, but also you can apply them to your architecture design. By reusing models and architectural descriptions, you can automate the architectural process by eliminating unnecessary work.
- *Traceability of technology choices* — Not only can you surface this information and apply it to your architecture designs, but now you can create relationships between what was imported and the architecture that you are designing.

As one example of this is the Patterns Browser. The intent of the Patterns Browser is to surface pattern information into the design environment. Two types of information are displayed:

- *Assets* — Shows what has been built.
- *Patterns* — Shows what should be built.

As Figure 8 shows, patterns can be applied to the elements of an existing architecture. In the preceding case, it is the reuse of a logical architecture model. Appling the selected model is as easy as double-clicking the pattern in the Patterns Browser. The pattern then is applied to the element that is selected in the document.

Another innovative way the Task Pane is utilized is by introducing collaboration. The architectural process is no longer a one-architect job. Typically multiple architects with specific roles design systems architecture. Examples of these roles include; Application, Information, Hardware, Security, or Infrastructure architect.

Other roles in review processes will have an impact on the validity, quality, and level of completeness of the designs. This affects the architecture in a significant way, too. These roles usually are part of an Architectural Review Board of sorts. In this function, it is critical that the architecture document give this group of individuals the information that it needs to make decisions on the architecture. By introducing the collaborative aspects to the architectural process, we can reduce the number of issues ahead of time, instead of at the end of the process — thus, changing the architecture-design process from its current form of a reactive process to a proactive process.

Architect Lookup would return the names and photos of the Hardware, Security, and Information architects. As Figure 9 shows, the Architect Lookup task pane reveals the architect resources, based on the project that is assigned to the architecture work. The way in which Architect Lookup determines this is by using the assigned project ID. The project ID is found in the meta-data of the document. This meta-data can be entered either manually or through automated means when the architecture-design request is sent.

Architect Lookup will take the project ID and perform a query against multiple systems to return a set of results. Project Server (or a custom information store) can be used to retrieve the list of architect resources, while Office SharePoint and Active Directory are used to take the resources in the PPM repository and correlate them with the images of the person, along with links to IM and personal portal sites.

Use of Architect Lookup eliminates the need to go to project plans, portals, or file shares to find this information manually. It is a simple click away to get instant access to other resources on the project, to get answers to questions.

Figure 10: Collaborative architectures

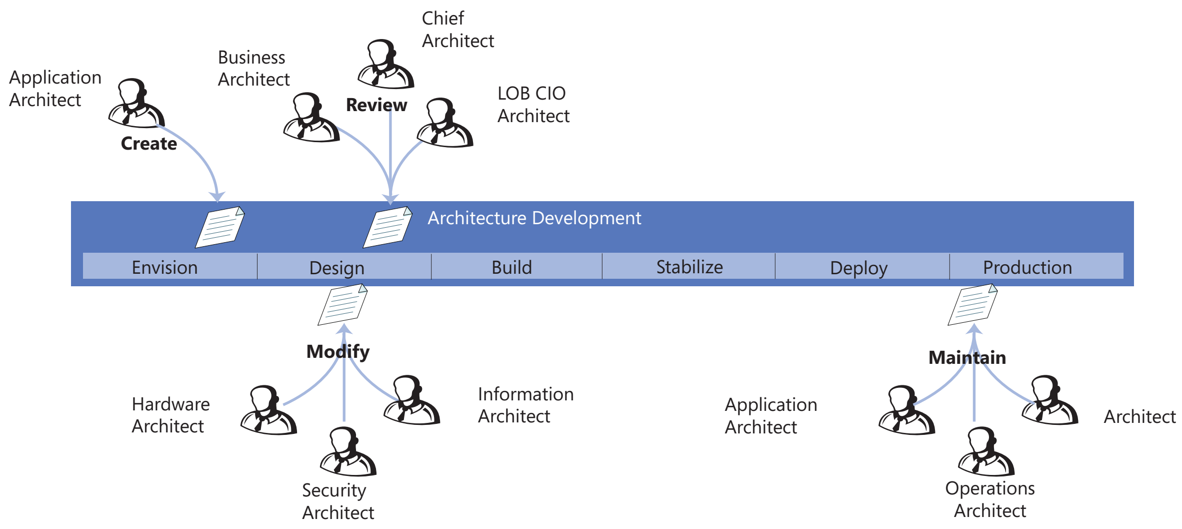
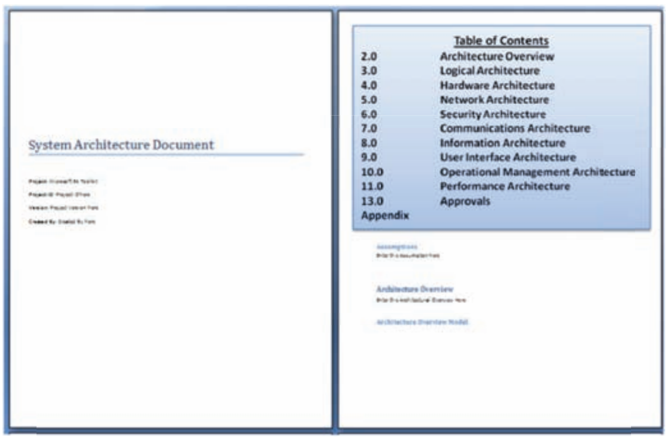


Figure 11: System-architecture document template



System Architecture Document

Table of Contents

2.0	Architecture Overview
3.0	Logical Architecture
4.0	Hardware Architecture
5.0	Network Architecture
6.0	Security Architecture
7.0	Communications Architecture
8.0	Information Architecture
9.0	User Interface Architecture
10.0	Operational Management Architecture
11.0	Performance Architecture
13.0	Approvals
Appendix	

Each role has a part to play in the architectural process. Given the illustration in Figure 10, we see that an Application architect might start the process, but then might need the aid of other architects. In this case, the Application architect might be the owner of the document and be the primary contributor. The other roles that are shown — such as Hardware architect, Security architect, and Information architect — are some that would interact. These roles will validate, add, and modify architecture-design descriptions.

In this context, a sample of common questions to these roles would look like the following:

- How will the security model affect how I design my application?
- I know that I must design an external portal, but what does the server-tier model look like for the DMZ?
- What will the network stack look like, and how will that affect performance or scalability?
- What is the right hardware?

Document-Template Architecture

Along with enhancements to the tool, the next layer of concern is the document template.

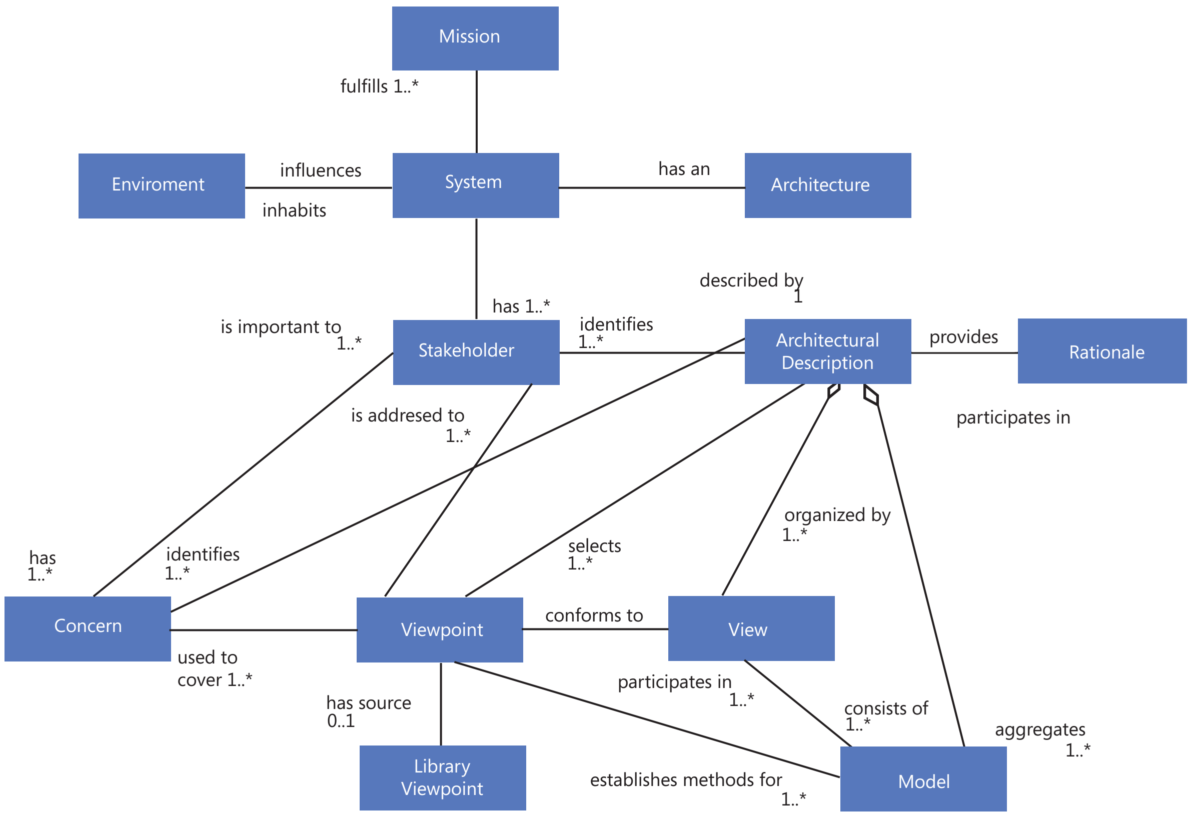
The document-template layer does not have a great deal of intelligence to it; however, it does allow interaction from Office Word to the document. One example is that the EATK allows information from an external source to be "clicked and dragged" into the document. This pulls data from a database and populates the document with architecture models.

The EATK provides not only the Office Word add-ins that are described in the previous sections of this article, but also a system-architecture document template, as Figure 11 shows:

The views that are listed in the table of contents are a set of common views. These views can be renamed or removed, and additional views can be added. These views provide a starting point for architects. Whereas you can use the system-architecture document template as your corporate standard in your company, it will be common practice to modify it.

With many documents that are surrounded by process, we want to create templates to ensure that they are used in a consistent way. This is the same case with the system-architecture document template. A template for the architecture document will allow for:

Figure 12: IEEE framework for architectural description



- Everyone using the right version of the architecture document.
- Ensuring that the information models stay intact.
- Allowing for information to be generated by the system.
- Enabling information to be consumed by downstream processes.

Information Architecture

The last piece of the puzzle is the information itself. Having great add-ins and template is great; however, without a meaningful way to express and ensure that the information is long living in a connected process, it is not as useful as it could be. Without information architecture, it would be difficult to qualify architectures.

The information layer is the base of the solution; it is where the majority of the consideration is made. All layers interact with each other in some way, but the information layer is connected directly to all. It is consumed in workflows, information-entry processes, and automation through Microsoft Office ribbons and task panes.

By using industry-standard techniques to derive to the target information architecture. Two fundamental aspects of the EATK that we must explore are the following:

- *Ontology* — We want to define a set of terms that are understood commonly within the enterprise. By doing so, we can relate information properly and consistently.
- *Taxonomy* — This will allow you to correlate architecture information with other aspects of architecture.

The architecture document should use the terms in the proper usage to qualify what the information is. Publishing an online ontology mapping will be useful toward understanding the information within the document. In the context of the system-architecture document, ontology provides agreed-upon terminology for architecture. As an example, it would define the meanings of a platform, system, subsystem, application, and components.

Defining what these architecture elements are, however, is one piece of the puzzle. How these elements relate to each other is the next logical step. We will use taxonomy for this. The EATK uses an industry standard from IEEE to solve this challenge. IEEE 1471 is used as the basis for the taxonomy and ontology of the system-architecture document.

IEEE 1471 is the first formalized standard for describing architectures and how they relate to other aspects of the software-development process. It leverages a scalable meta-model that shows the relationships between common elements in the architecture-development process.

In Figure 12, IEEE 1471 provides a meta-model that allows us to relate architectures with other aspects of the software-development process (page 34). The system-architecture document focuses on specific areas of the taxonomy, while other EATK components focus on other aspects of the standard.

The aspects that are implemented to support IEEE 1471 in the system-architecture document are the following:

- *Structured content* — An architecture schema that represents the information that we want to gather in our architecture document was created that has a viewpoint-based model applied to it.
- *Qualifying information* — Links between decisions that have been made and other systems or architecture descriptions are built into the schema. As a user enters information or applies patterns to the architecture document, it will correlate that information by unique ID.
- *Publishing mechanisms* — Provides facilities to store information from the schema, so that it can be related to other non-document–related information.
- *Generating information* — Provides a mechanism to rebuild or generate sections of an architecture document.

System-Architecture Markup XML

With an ontology and taxonomy for architecture, we now can look at what this means from an implementation perspective. The system-architecture document has an underlying XML structure that is used to describe the information in the form of XML.

The System Architecture Markup XML is a way to provide clear separation between the presentation markup and the information markup. This greatly simplifies integration to hosted workflow, repositories, and third party tooling.

Conclusion

In this article, we reviewed how to change fundamentally the way in which we view, approach, and maintain architecture descriptions. There is no doubt that there could be significant value in capturing information about the designs of our architectures that will help guide decision makers through the processes of making the right architecture decisions. However, this is not always the outcome, as often no formalized structure, process, information model, or intelligent tooling is bundled with this process. Key takeaways include the following:

- Architects who want to implement these concepts can use the EATK, which provides a set of templates, processes, and a solution accelerator that can be used to accelerate this implementation for architects.
- Architects can achieve significant productivity gains through reduction of manual activities and process automation.
- Architecture information can be used in a much more meaningful way by eliminating the document graveyard effect, by integrating architecture descriptions with an AMR.
- The quality of decision making can be improved by eliminating points of duplication; thus, information quality can be increased significantly.
- Wikis and modeling tools complement this implementation—or, in some cases, replace it.
- Solutions such as COTS and custom-developed applications can be integrated with this solution through standard integration technologies.

About the Author

Mike Walker is a principle architect who delivers strategy for the enterprise architecture space at Microsoft. He is responsible for creating, driving, and evangelizing Microsoft's worldwide Enterprise 2.0 and Enterprise Architecture strategies. Specifically, Mike ensures that institutions around the world realize the full extent of Microsoft's vision and value proposition. He has also evolved many of the concepts behind mainstream architectural approaches and styles. His works are realized through publication in books, trade publications, authoritative architecture guidance, articles, code delivery and standardized frameworks. Those frameworks include: the Enterprise Architecture Toolkit (EATK), Loan Origination Reference Architecture, and Financial Services Component Library. You can visit his blog at http://www.MikeTheArchitect.com

Follow up on this topic

- Enterprise Architecture: http://msdn.microsoft.com/architecture/ea
- IEEE 1471: http://www.iso-architecture.org/ieee-1471/
- Describing the Enterprise Architectural Space: http://msdn.microsoft.com/en-us/library/ms978655.aspx
- Enterprise Architecture Design and the Integrated Architecture Framework: http://msdn.microsoft.com/en-us/library/aa480017.aspx

A Language for Software Architecture

by J.D. Meier

Summary
One of the most important outcomes of the patterns & practices Application Architecture Guide 2.0 project is a language for the space: a language for application architecture. Building software applications involves a lot of important decisions. By organizing these decisions as a language and a set of mental models, we can simplify organizing and sharing information. By mapping out the architecture space, we can organize and share knowledge more effectively. By using this map as a backdrop, we can also overlay principles, patterns, technologies, and key solutions assets in meaningful and relevant ways. Rather than a sea of information, we can quickly browse hot spots for relevant solutions.

A Map of the Terrain

One of the most effective ways to deal with information overload is to frame a space. Just like you frame a picture, you can frame a problem to show it a certain way. When I started the patterns & practices Application Architecture Guide 2.0 project, the first thing I wanted to do was to frame out the space. Rather than provide step-by-step architectural guidance, I thought it would be far more valuable to first create a map of what's important. We could then use this map to prioritize and focus our efforts. We could also use this map as a durable, evolvable backdrop for creating, organizing and sharing our patterns & practices work. Figure 1 shows the main map, the Architecture Frame, we created to help us organize and share principles, patterns, and practices in the application architecture space.

Mapping Out the Architecture Space

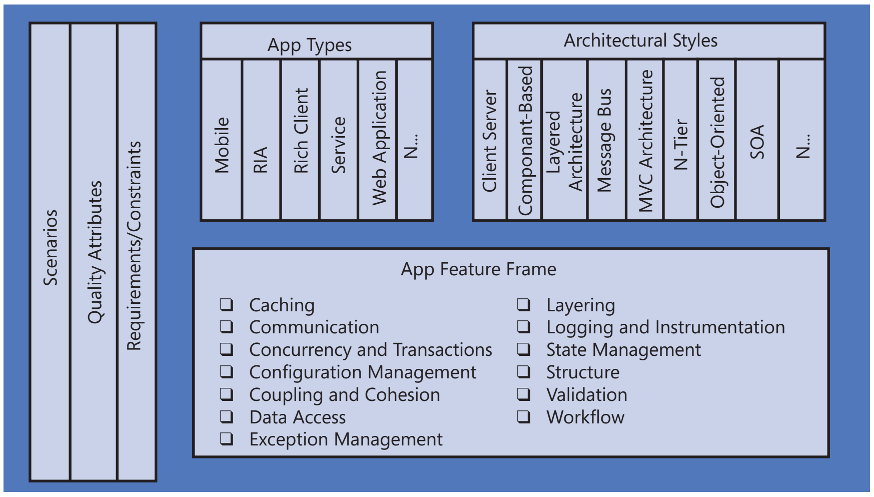
Creating the map was an iterative and incremental process. The first step was to break up application architecture into meaningful buckets. It first started when I created a project proposal for our management team. As part of the proposal, I created a demo to show how we might chunk up the architecture space in a meaningful way. In the demo, I included a list of key trends, a set of application types, a set of architectural styles, a frame for quality attributes, an application feature frame, a set of example deployment patterns, and a map of patterns & practices solution assets. I used examples where possible simply to illustrate the idea. It was well received, and it served as a strawman for the team.

Each week, our core Application Architecture Guide 2.0 project team met with our extended development team, which primarily included patterns & practices development team members. During this time, we worked through a set of application types, created a canonical application, analyzed layers and tiers, evaluated key trends, and created technology matrix trade-off charts. To create and share information rapidly, we created many mind maps and slides. The mind maps worked well. Rather than get lost in documents, we used the mind maps as backdrops for conversation and elaboration.

Key Mapping Exercises
We mapped out several things in parallel:

- *Key trends.* Although we didn't focus on trends in the guide, we first mapped out key trends to help figure out what to pay attention to. We used a mind map and we organized key trends by application, infrastructure, and process. While there weren't any major surprises, it was a healthy exercise getting everybody on the same page in terms of which trends mattered.
- *Canonical application.* The first thing we did was figure out the delta from the original architecture guide. There were a few key changes. For example, we found that today's applications have a lot more clients and scenarios they serve. They've matured and they've been extended. We

Figure 1: Architecture Frame



also found today's applications have a lot more services, both in terms of exposing and in terms of consuming. We also noticed that some of today's applications are flatter and have fewer layers. Beyond that, many things such as the types of components and the types of layers were fairly consistent with the original model.

- *Layers and tiers.* This was one of the more painful exercises. Early in the project, we met each week with our development team, along with other reviewers. The goal was to map out the common layers, tiers, and components. While there was a lot of consistency with the original application architecture guide, we wanted to reflect any learnings and changes since the original model. Once we had a working map of the layers, tiers, and components, we vetted the map with multiple customers.
- *Application types.* We originally explored organizing applications around business purposes or dominant functionality, customer feedback told us we were better off optimizing around technical types, such as Web application or mobile client. They were easy for customers to identify with. They also made it easy to overlay patterns, technologies, and key patterns & practices solution assets. The technical application types also made it easy to map out relevant technologies.
- *Architectural styles.* This is where we had a lot of debate. While we ultimately agreed that it was helpful to have a simple language for abstracting the shapes of applications and the underlying principles from the technology, it was difficult to create a map that everybody was happy with. Things got easier once we changed some of the terminology and we organized the architectural styles by common hot spots. It then became obvious that the architectural styles are simply named sets of principles. We could then have a higher level conversation around whether to go with object-based community or message-based and SOA, for example. It was also easy to describe deployments in terms of 2-tier, 3-tier, and N-tier.
- *Hot spots for architecture.* When you build applications, there's a common set of challenges that show up again, such as caching, data access, exception management, logging, and so on. These are application infrastructure problems or cross-cutting concerns. You usually don't want to make these decisions ad hoc on any significant application. Instead, you want to have a set of patterns and guidelines or ideally reusable code that the team can leverage throughout the application. What makes these hot spots is that they are actionable, key engineering decisions. You want to avoid do-overs where you can. Some do-overs are more expensive than others. One of the beauties of the architecture hot spots is that they helped show the backdrop behind Enterprise Library. For example, there's a data access block, a caching block, a validation block, and so forth.
- *Hot spots for application types.* When you build certain classes of application, there's recurring hot spots. For example, when you build a rich client, one of the common hot spots to figure out is how to handle occasionally disconnected scenarios. The collection of hot spots for architecture served as a baseline for finding hot spots in the other application types. For example, from the common set of hot spots, we could then figure out which ones are relevant for Web applications, or which additional hot spots would we need to include.
- *Patterns.* Mapping out patterns was a lengthy process. Ultimately, we probably ended up with more information in our workspace than made it into the guide. To map out the patterns, we created multiple mind maps of various pattern depots. We summarized patterns so that we could quickly map them from problems to solutions. We then used our architecture hot spots and our hot spots for application types as a filter to find the relevant patterns. We then vetted the patterns with customers to see if the mapping was useful. We cut any patterns that didn't seem high enough priority. We also cut many of our pattern descriptions when they started to weight the guide down. We figured we had plenty of material and insight to carve out future pattern guides and we didn't want to overshadow the value of the main chapters in the guide. We decided the best move for now was to provide a Pattern Map at the end of each application chapter to show which patterns are relevant for key hot spots. Customers seemed to like this approach, and it kept things lightweight.
- *Patterns & practices solution assets.* This was the ultimate exercise in organizing our catalog. We actually have a large body of documented patterns. We also have several application blocks and factories, as well as guides. By using our architecture frame, it was easier to organize the catalog. For example, the factories and reference implementations mapped to the application types. The Enterprise Library blocks mapped to the architecture hot spots. Several of the guides mapped to the quality attributes frame. For more information, see Cheat Sheet – patterns & practices Catalog at a Glance posted to CodePlex at http://blogs.msdn.com/jmeier/archive/2008/10/09/cheat-sheet-patterns-practices-catalog-at-a-glance-posted-to-codeplex.aspx .
- *Microsoft platform.* This was a challenge. It meant slicing and dicing the platform stack in a meaningful way as well as finding the right product team contacts. Once we had our application types in place, it got a lot easier. Depending on which type of application you were building — rich internet application (RIA), Web, mobile, for example — this quickly narrowed down relevant technology options. We created technology matrices for presentation technologies, integration technologies, workflow technologies, and data access technologies. Since the bulk of the guide is principle and pattern based, we kept these matrices in the appendix for fast lookups.

Key Components of the Application Architecture Map

Over the weeks and months of the project, a very definite map of the landscape emerged. We found ourselves consistently looking for the same frames to organize information. While we tuned and pruned specific hot spots in areas, the overall model of common frames was helping us move through the space quickly.

- *Architecture frame.* The architecture frame was the main organizing map. It brought together the context (scenarios, quality attributes, requirements/constraints), application types, architectural styles, and the application hot spots.
- *Application types.* For application types, we optimized around a simple, technical set that resonated with customers, such as Web application, RIA, and mobile.
- *Quality attributes.* We organized quality attributes by key hot spots: system, runtime, design-time, and user qualities.
- *Architectural styles.* We organized architectural styles by key hot spots: communication, deployment, domain, interaction, and structure.
- *Requirements and constraints.* We organized requirements by key types: functional, non-functional, technological. We thought of constraints in terms of industry and organizational constraints, as well as by which concern (for example, constraints for security or privacy).
- *Application feature frame.* The application feature frame became a solid backdrop for organizing many guidelines through the guide. The hot spots resonated: caching, communication, concurrency and transactions, configuration management, coupling and cohesion, data access, exception management, layering, logging and instrumentation, state management, structure, validation and workflow.
- *Application type frames.* The application type frames are simply hot spots for key application types. We created frames for: Web applications, RIA, mobile applications, rich client applications and services.

- *Layered architecture reference model.* The canonical application is actually a layered architecture reference model. It helps show the layers and components in context.
- *Layers and tiers.* We used layers to represent logical partitions and tiers for physical partitions (this precedent was set in the original guide.) We identified key components within the key layers: presentation layer, business layer, data layer, and service layer.
- *Pattern maps.* Pattern maps are simply overlays of key patterns on top of relevant hot spots. We created pattern maps for the application types.
- *Product and technology maps.* We created technology matrices for relevant products and technologies. To put the technologies in context, we used application types where relevant. We also used scenarios. To help make trade-off decisions, we included benefits and considerations for each technology.

User, Business, and System Perspective

One thing that helped early on was creating a Venn diagram of the three perspectives, user, business, and system, as shown in Figure 2.

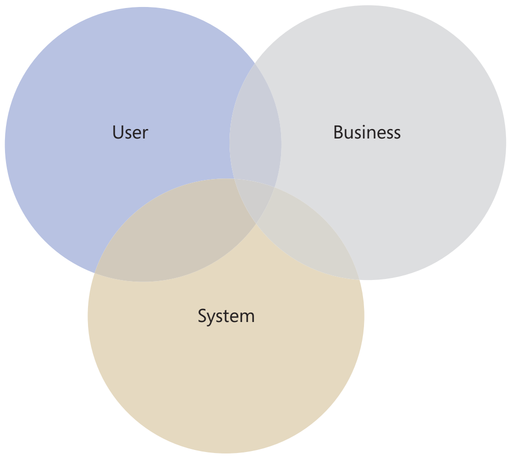
In application architecture, it's easy to lose perspective. It helps to keep three perspectives in mind. By having a quick visual of the three perspectives, it was easy to remind ourselves that architecture is always a trade-off among these perspectives. It also helped remind us to be clear which perspective we're talking about at any point in time. This also helped resolve many debates. The problem in architecture debates is that everybody is usually right, but only from their perspective. Once we showed people where their perspective fit in the bigger picture, debates quickly turned from conflict to collaboration. It was easy to move through user goals, business goals, and system goals once people knew the map.

Architecture Frame

The Architecture Frame is a simple way to organize the space (Figure 1). It's a durable, evolvable backdrop. You can extend it to suit your needs. The strength of the frame is that it combines multiple lenses.
Here are the key lenses:

- *Scenarios.* This sets the context. You can't evaluate architecture in a vacuum. You need a backdrop. Scenarios provide the backdrop for evaluation and relevancy.
- *Quality attributes.* This includes your system qualities, your runtime qualities, your design-time qualities and user qualities.
- *Requirements / constraints.* Requirements and constraints includes functional requirements, non-functional requirements, technological requirements, industry constraints and organizational constraints.
- *Application types.* This is an extensible set of common types of applications or clients. You can imagine extending for business types. You can imagine including just the types of applications your organization builds. Think of it as product-line engineering. When you know the types of applications you build, you can optimize it.
- *Architectural styles.* This is a flat list of common architectural styles. The list of architectural styles is flexible and most applications are a mash up of various styles. Architectural styles become more useful when they are organized by key decisions or concerns.
- *Application feature frame.* The application feature frame is a concise set of hot spots that show up time and again across applications. They reflect cross-cutting concerns and common application infrastructure challenges.

Figure 2: User, Business, and System Perspectives



Application Types

We defined a simple set of technical application types:

- *Web applications.* Applications of this type typically support connected scenarios and can support different browsers running on a range of operating systems and platforms.
- *RIA.* applications of this type can be developed to support multiple platforms and multiple browsers, displaying rich media or graphical content. Rich Internet applications run in a browser sandbox that restricts access to some devices on the client.
- *Mobile applications.* Applications of this type can be developed as thin client or rich client applications. Rich client mobile applications can support disconnected or occasionally connected scenarios. Web or thin client applications support connected scenarios only. The device resources may prove to be a constraint when designing mobile applications.
- *Rich client applications.* Applications of this type are usually developed as stand-alone applications with a graphical user interface that displays data using a range of controls. Rich client applications can be designed for disconnected and occasionally connected scenarios because the applications run on the client machine.
- *Services.* Services expose complex functionality and allow clients to access them from local or remote machine. Service operations are called using messages, based on XML schemas, passed over a transport channel. The goal in this type of application is to achieve loose coupling between the client and the server.

Application Feature Frame

This is the set of hot spots for applications we defined:

- *Authentication and authorization.* Authentication and authorization allow you to identify the users of your application with confidence, and to determine the resources and operations to which they should have access.
- *Caching and state.* Caching improves performance, reduces server round trips, and can be used to maintain the state of your application.
- *Communication.* Communication strategies determine how you will communicate between layers and tiers, including protocol, security, and communication-style decisions.
- *Composition.* Composition strategies determine how you manage component dependencies and the interactions between components.
- *Concurrency and transactions.* Concurrency is concerned with the way that your application handles conflicts caused by multiple users creating, reading, updating, and deleting data at the same time. Transactions are used for important multi-step operations in order to treat them as

though they were atomic, and to recover in the case of a failure or error.

- *Configuration management.* Configuration management defines how you configure your application after deployment, where you store configuration data, and how you protect the configuration data.
- *Coupling and cohesion.* Coupling and cohesion are strategies concerned with layering, separating application components and layers, and organizing your application trust and functionality boundaries.
- *Data access.* Data access strategies describe techniques for abstracting and accessing data in your data store. This includes data entity design, error management, and managing database connections.
- *Exception management.* Exception-management strategies describe techniques for handling errors, logging errors for auditing purposes, and notifying users of error conditions.
- *Logging and instrumentation.* Logging and instrumentation represents the strategies for logging key business events, security actions, and provision of an audit trail in the case of an attack or failure.
- *User experience.* User experience is the interaction between your users and your application. A good user experience can improve the efficiency and effectiveness of the application, while a poor user experience may deter users from using an otherwise well-designed application.
- *Validation.* Validation is the means by which your application checks and verifies input from all sources before trusting and processing it. A good input and data-validation strategy takes into account not only the source of the data, but also how the data will be used, when determining how to validate it.
- *Workflow.* Workflow is a system-assisted process that is divided into a series of execution steps, events, and conditions. The workflow may be an orchestration between a set of components and systems, or it may include human collaboration.

Architectural Styles

For architectural styles, we first framed the key concerns to organize the architectural styles, and then we defined some common architectural styles.

Organizing Architectural Styles

These are the hot spots we used to organize architectural styles:

- *Communication.* Service-Oriented Architecture(SOA) and/or Message Bus and/or Pipes and Filters.
- *Deployment.* Client/server or 3-Tier or N-Tier.
- *Domain.* Domain Model or Gateway.
- *Interaction.* Separated Presentation.
- *Structure.* Component-Based and/or Object-Oriented and/or Layered Architecture.

Architectural Style Frame

These are some commonly recognized architectural styles:

- *Client-server.* Segregates the system into two applications, where the client makes a service request to the server.
- *Component-based architecture.* Decomposes application design into reusable functional or logical components that are location-transparent and expose well-defined communication interfaces.
- *Layered architecture.* Partitions the concerns of the application into stacked groups (layers) such as presentation layer, business layer, data layer, and services layer.
- *Message-bus.* A software system that can receive and send messages that are based on a set of known formats, so that systems can communicate with each other without needing to know the actual recipient.
- *N-tier/3-tier.* Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.
- *Object-oriented.* An architectural style based on division of tasks for an application or system into individual reusable and self-sufficient objects, each containing the data and the behavior relevant to the object.
- *Separated presentation.* Separates the logic for managing user interaction from the user interface (UI) view and from the data with which the user works.
- *Service-oriented architecture.* Refers to Applications that expose and consume functionality as a service using contracts and messages.

Quality Attributes

For quality attributes, we first framed the key categories to organize the quality attributes, and then we defined some common quality attributes.

Organizing Quality Attributes

Table 1 shows a simple way to organize and group quality attributes:

Quality Attribute Frame

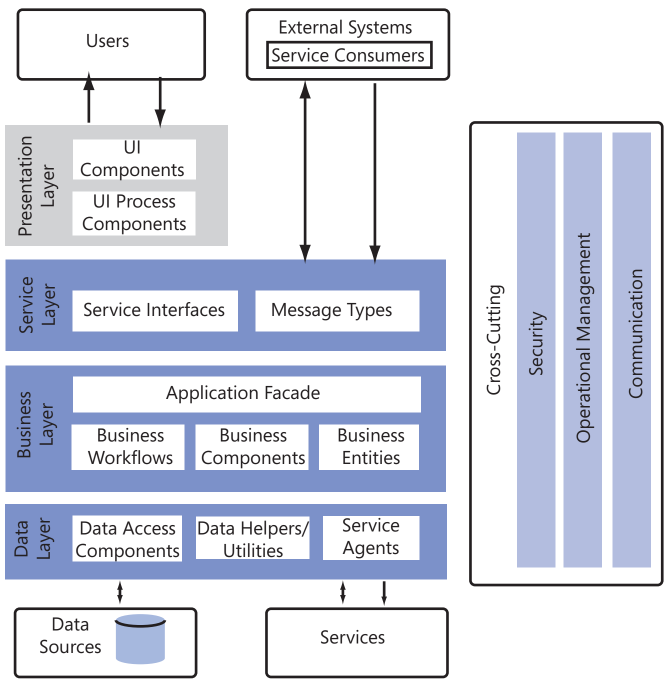
Some common quality attributes include:

- *Availability.* Availability is the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load.
- *Conceptual integrity.* Conceptual integrity is the consistency and coherence of the overall design. This includes the way that components or modules are designed, as well as factors such as coding style and variable naming.
- *Flexibility.* The ability of a system to adapt to varying environments and situations, and to cope with changes in business policies and rules. A flexible system is one that is easy to reconfigure or adapt in response to different user and system requirements.
- *Interoperability.* Interoperability is the ability of diverse components of a system or different systems to operate successfully by exchanging information, often by using services. An interoperable system makes it easier to exchange and reuse information internally as well as externally.
- *Maintainability.* Maintainability is the ability of a system to undergo changes to its components, services, features, and interfaces as may be required when adding or changing the functionality, fixing errors, and meeting new business requirements.

Table 1: Quality Attributes

Type	Quality attributes
System Qualities	• Supportability • Testability
Run-time Qualities	• Availability • Interoperability • Manageability • Performance • Reliability • Scalability • Security
Design Qualities	• Conceptual Integrity • Flexibility • Maintainability • Reusability
User Qualities	• User Experience / Usability

Figure 3: Layered Architecture Model



- *Manageability.* Manageability is how easy it is to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning.
- *Performance.* Performance is an indication of the responsiveness of a system to execute any action within a given time interval. It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within a given amount of time.
- *Reliability.* Reliability is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval.
- *Reusability.* Reusability is the capability for components and subsystems to be suitable for use in other applications and in other scenarios. Reusability minimizes the duplication of components and also the implementation time.
- *Scalability*. Scalability is the ability of a system to function well when there are changes to the load or demand. Typically, the system will be able to be extended over more powerful or more numerous servers as demand and load increase.
- *Security.* Security is the ways that a system is protected from disclosure or loss of information, and the possibility of a successful malicious attack. A secure system aims to protect assets and prevent unauthorized modification of information.
- *Supportability.* Supportability is how easy it is for operators, developers, and users to understand and use the application, and how easy it is to resolve errors when the system fails to work correctly.
- *Testability.* Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Good testability makes it more likely that faults in a system can be isolated in a timely and effective manner.
- *Usability.* Usability defines how well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, and able to provide good access for disabled users and a good overall user experience.

Layered Architecture Reference Model

Figure 3 shows our canonical application example. It's a layered architecture showing the common components within each layer:
The canonical application model helped us show how the various layers and components work together. It was an easy diagram to pull up and talk through when we were discussing various design trade-offs at the different layers.

Layers

We identified the following layers:

- *Presentation layer*
- *Business layer*
- *Data layer*
- *Service layer*

They are logical layers. The important thing about layers is that they help factor and group your logic. They are also fractal. For example, a service can have multiple types of layers within it. The following is a quick explanation of the key components within each layer.

Presentation Layer Components

- *User interface (UI) components.* UI components provide a way for users to interact with the application. They render and format data for users and acquire and validate data input by the user.
- *User process components.* To help synchronize and orchestrate these user interactions, it can be useful to drive the process by using separate user process components. This means that the process-flow and state-management logic is not hard-coded in the UI elements themselves, and the same basic user interaction patterns can be reused by multiple UIs.

Business Layer Components

- *Application facade* (optional). Use a façade to combine multiple business operations into a single message-based operation. You might access the application façade from the presentation layer by using different communication technologies.
- *Business components.* Business components implement the business logic of the application. Regardless of whether a business process consists of a single step or an orchestrated workflow, your application will probably require components that implement business rules and perform business tasks.
- *Business entity components.* Business entities are used to pass data between components. The data represents real-world business entities, such as products and orders. The business entities used internally in the application are usually data structures, such as DataSets, DataReaders, or Extensible Markup Language (XML) streams, but they can also be implemented by using custom object-oriented classes that represent the real-world entities your application has to work with, such as a product or an order.
- *Business workflows.* Many business processes involve multiple steps that must be performed in the correct order and orchestrated. Business workflows define and coordinate long-running, multi-step business processes, and can be implemented using business process management tools.

Figure 4: Two-Tier Deployment

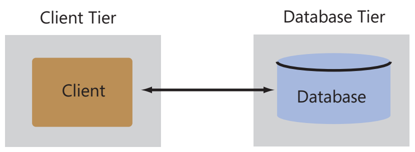
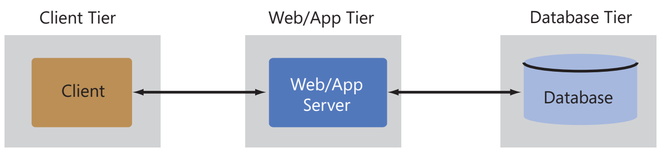


Figure 5: Three-Tier Deployment



Data Layer Components

- *Data access logic components.* Data access components abstract the logic necessary to access your underlying data stores. Doing so centralizes data access functionality, and makes the process easier to configure and maintain.
- *Data helpers / utility components.* Helper functions and utilities assist in data manipulation, data transformation, and data access within the layer. They consist of specialized libraries and/or custom routines especially designed to maximize data access performance and reduce the development requirements of the logic components and the service agent parts of the layer.
- *Service agents.* Service agents isolate your application from the idiosyncrasies of calling diverse services from your application, and can provide additional services such as basic mapping between the format of the data exposed by the service and the format your application requires.

Service Layer Components

- *Service interfaces.* Services expose a service interface to which all inbound messages are sent. The definition of the set of messages that must be exchanged with a service, in order for the service to perform a specific business task, constitutes a contract. You can think of a service interface as a façade that exposes the business logic implemented in the service to potential consumers.
- *Message types.* When exchanging data across the service layer, data structures are wrapped by message structures that support different types of operations. For example, you might have a Command message, a Document message, or another type of message. These message types are the "message contracts" for communication between service consumers and providers.

Tiers

Tiers represent the physical separation of the presentation, business, services, and data functionality of your design across separate computers and systems. Some common tiered design patterns include two-tier, three-tier, and n-tier.

Two-Tier

The two-tier pattern represents a basic structure with two main components, a client and a server (Figure 4).

Three-Tier

In a three-tier design, the client interacts with application software deployed on a separate server, and the application server interacts with a database that is also located on a separate server (Figure 5). This is a very common pattern for most Web applications and Web services.

N-Tier

In this scenario, the Web server (which contains the presentation layer logic) is physically separated from the application server that implements the business logic (Figure 6).

Conclusion

It's easier to find your way around when you have a map. By having a map, you know where the key hot spots are. The map helps you organize and share relevant information more effectively. More importantly, the map helps bring together archetypes, arch styles, and hot spots in a meaningful way. When you put it all together, you have a simple language for describing large classes of applications, as well as a common language for application architecture.

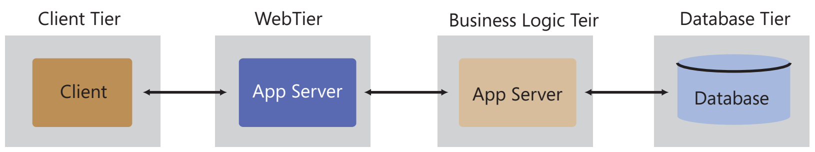
Resources

- Guide – The patterns & practices Application Architecture Guide 2.0 is available online in HTML and PDF at http://www.codeplex.com/AppArchGuide
- Knowledge Base (KB) – The companion knowledge base, which includes videos, How Tos, checklists and diagrams is available at http://www.codeplex.com/AppArch
- Project News – For projects news and announcements, you can follow along at http://blogs.msdn.com/jmeier

About the Author

J.D. Meier is a Principal Program Manager at Microsoft on the patterns & practices team. In addition to the Microsoft Application Architecture 2.0 guide, he has the following additional patterns & practices books under his belt: Improving Web Services Security, Performance Testing Guidance for Web Applications, Team Development with Visual Studio Team Foundation Server, Security Engineering Explained, Improving .NET Application Performance, Improving Web Application Security, and Building Secure ASP.NET Applications. You can follow along at his work blog at http://blogs.msdn.com/jmeier, his software success blog at http://ShapingSoftware.com and personal development blog at http://SourcesOfInsight.com.

Figure 6: N-Tier Deployment



Follow up on this topic

- Patterns and Practices (Solutions Architecture Guidance): http://msdn.microsoft.com/practices
- Solution Accelerator (Infrastructure Guidance): http://technet.microsoft.com/solutionaccelerators/

Platform
Architecture Team

Microsoft®

19

098-110486

subscribe at

www.architecturejournal.net