

Meeting the Requirements of High-Demand Distributed Applications: An Enterprise Architect's Blueprint

Enterprise architects are working in a more demanding environment than ever before. They are being asked to deliver applications that are unprecedented in their complexity, their degree of integration with other applications, and their level of distribution. Architects have used considerable ingenuity leveraging their traditional palette of capabilities, but building high-performance distributed applications remains very challenging.

New tools and techniques have emerged to make the job easier, and some come from unexpected sources. The network layer, for one, has gradually become an important provider of architectural services. Classic transport networks have been optimized to increased throughput, availability, and configurability and are increasingly application-aware. But many enterprise architects still think of the network layer as mere “plumbing” and remain unaware of the new application-related, value-add services that today’s network can provide.

This paper is meant to introduce enterprise architects (EAs) to the expanded capabilities residing in the network and to provide guidance on taking best advantage of them.

The Challenge to EAs

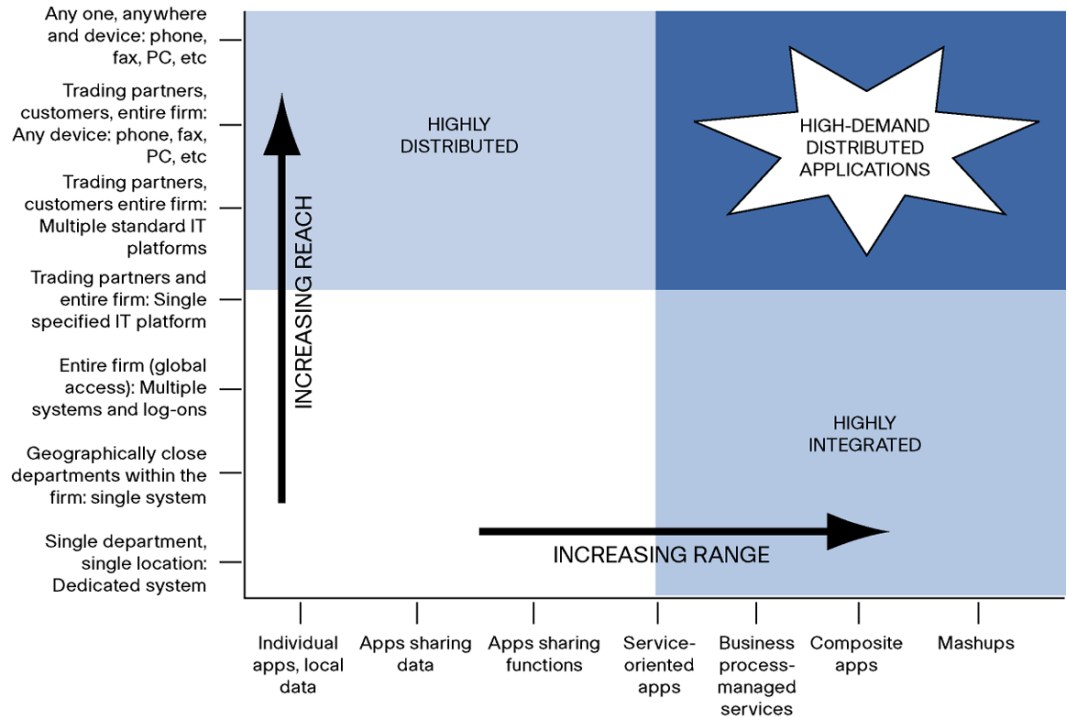
As organizations expand and affiliate, their enterprise applications become more complex, more distributed, and more integrated with other applications both inside the enterprise and beyond. Forward-looking enterprises want applications that offer an extensive range of capabilities plus the ability to connect a dispersed community of people who are using a variety of devices. This is a significant challenge.

The burden falls heavily on enterprise architects. The traditional EA responsibilities — charting the enterprise’s growing complexity; managing that complexity with models and best practices; ensuring that systems align with the needs of the current and future business; serving as liaison between the business and development communities; and specifying the technologies, techniques, and tools that make up the environments in which applications are delivered — are all more demanding in this new environment. The most coherent and mature enterprise architecture practice, even one supplemented with standard EA frameworks such as the Open Group’s TOGAF or the federal government’s FEA initiative, needs to look for new tools and resources to improve performance.

Reach and Range

One way to understand the demands on today’s EA is to use the Reach/Range analysis developed in the 1990s by IT consultant and Harvard professor Peter G.W. Keen. This way of thinking about emerging patterns in technology integration has been productively applied to many areas of IT, including the analysis of messaging systems, supply chains, and business process management solutions.

Figure 1. Reach and Range



In Keen’s model, Reach refers to how distributed an application is. The earliest IT architectures (the bottom of the y-axis in Figure 1) had no reach whatsoever. Applications ran in the data center and the user’s only involvement was to supply the punchcards going in and read the reports coming out.

Over time, business innovation and technical breakthroughs increased application distribution. The power of information reached beyond the glass house, out to the users’ desktops, then to these users’ in-house clients, then to other lines of business in neighboring states, then into other countries. By the 1990s, as extranet access beyond the enterprise seemed within grasp, the global Internet and the World Wide Web shook remaining assumptions as to how far Reach could extend. Reach now pushes beyond browsers into phones, badge readers, and wherever RFID technology can be imagined.

Range is an application’s degree of integration with other applications, the “range” of its functionality. Range, shown on the x-axis in Figure 1, was for a long time limited to individual applications working on local data. Even as architectural Reach was improving, business applications remained islands unto themselves, often described as stovepipes of isolated functionality.

Range took a longer time to improve because it really is a bigger challenge. Increasing an application’s Range involves sophisticated per-project planning around the meaning of data integration, function integration, service integration, and more. This is precisely the kind of thinking that constitutes modern EA practice, and Range remained limited until architectural thinking about application integration became more sophisticated.

Range did improve over time. Applications that had kept their own data locally evolved into applications that shared first their data stores, then their data models. Architects came to recognize that the path to scalable integration lay in having applications share not just data but behavior as well. The twin forces of business opportunity and technological enablement worked again, this time

leading to more loosely coupled architectures such as service-oriented architectures. Internet experiments with social networking showed how application components can form and reform more quickly and easily than conventionally programmed solutions. The current state of the art is those composite applications called mashups that go beyond integrating data and text to incorporate maps, photos, music, and video-on-demand.

High-Demand Distributed Applications

For most of IT's history, the evolution of Reach and Range have proceeded independently, as shown by the arrows in Figure 1 heading off in different directions. There was a sort of mutual exclusivity at work. Long-Reach applications were light on integrated functionality because integration depends on speed, and long distances are hard to cover quickly. On the other hand, broad-Range applications required sophisticated integration interfaces and rich media transports that were hard to run over an extended network.

That is no longer the case. Thanks to Web 2.0 technology, applications with both long Reach and broad Range are now common. They might be called High-Demand Distributed Applications and, as Figure 1 shows, they are today's gold standard. Business users, customers, partners, and prospects experience such applications in their personal lives and they are coming to expect similar capabilities in enterprise applications.

But building a high-demand enterprise application that supports enterprise-quality scalability, availability, and security is quite difficult. Today's virtualized data center adds yet another layer to the complexity of distributed apps. Virtual servers can easily be moved from one data center to another, causing a sudden change in performance characteristics to the client (Reach) and altering interoperability characteristics to collaborating components (Range). Such a move can have beneficial effects for one constituency while crippling another, and EAs have to anticipate this possibility.

Clearly, there is a looming capability gap between application complexity and existing architectural practice. EAs must look for help in new places. In at least one case, help is surprisingly close by. Networks — the ubiquitous, unglamorous architectural infrastructure that has historically provided application “plumbing” — can provide application-enhancing services, many of which run more efficiently at the network level than their equivalents in the application server.

Application Delivery Services in the Network

Network technology providers have recognized the architectural logjam created by the need for simultaneously longer Reach and broader Range. As the performance burden on servers increases, a re-visioned network seems a natural place to offload some scalability/performance responsibilities. Data already passes through the network; it makes sense to enhance it en route.

The response is a new, enhanced application delivery network. Where networks of previous generations dealt with message transport, routing, and similar basic issues, the application delivery network actively contributes to the scalability and performance of the applications that pass data through it. Many activities can now be performed “on the wire.” These include elemental operations such as data compression, resource caching, optimal content distribution, and various protocol optimizations, as well as higher-level operations for the performance-hungry XML message processing that service-oriented architectures require.

As the term “application delivery” implies, the network is prepared to deal with more than its customary task, the generic delivery of bits. It has become aware of higher-level layers of the

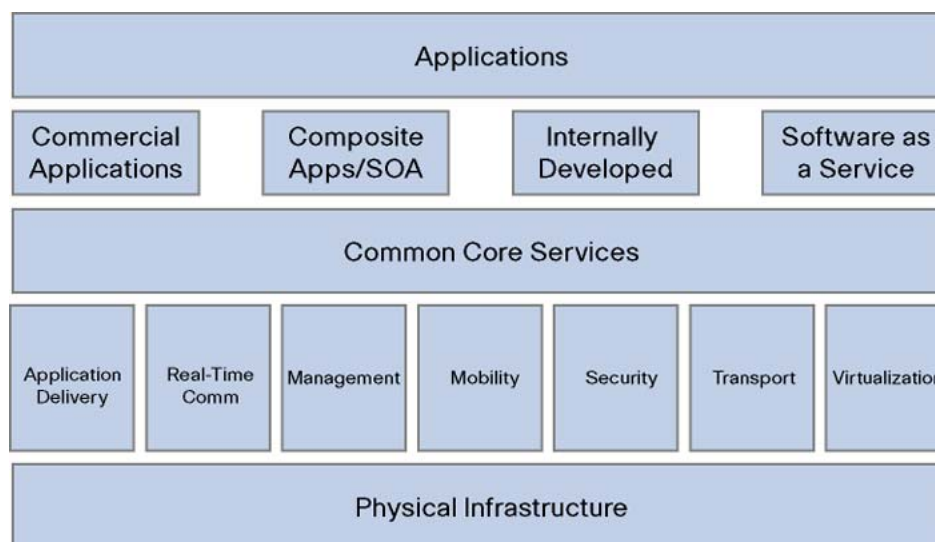
communication stack, especially the presentation and application layers. For example, an application delivery network can go beyond generic compression techniques and apply an extra measure of image-specific compression when an image (a kind of presentation element) is requested.

Cisco has extended the application delivery concept into application-specific protocols such as the basic web-transport HTTP and its web service cousin SOAP, into data formats such as XML, and into specific media formats for voice and video. This application-awareness, sometimes called “application fluency,” can even extend into detailed knowledge of specific application suites such as SAP and other certified offerings.

Network-based Services in Context

Network-based application delivery services are usually part of a larger structure of enhanced network services. Cisco, for example, has designed a framework view of network-based services called the Cisco[®] Service Oriented Network Architecture (SONA). The SONA framework consists of three layers: applications, core common services, and infrastructure.

Figure 2. Cisco SONA Framework



As Figure 2 shows, the Core Common Services Layer comprises a library of network-based service categories working together to create functionality that can be used by the Applications Layer and are implemented in the physical infrastructure layer. The service categories in the Core Common Services Layer are:

- Application Delivery Services which use application awareness to optimize performance.
- Real-Time Communication Services which offer session and media management capabilities, contact center services, and presence functions.
- Management Services which offer configuration and reporting capabilities.
- Mobility Services which offer location information as well as device-dependent functionality.
- Security Services which help protect the infrastructure, data, and application layers from constantly evolving threats.
- Transport Services which help deliver on the overall QoS requirements of the application, as well as providing routing and topology functions.

- Virtualization Services which delivers abstraction between physical and functional elements in the infrastructure.

This paper focuses specifically on Application Delivery Services.

The Power of Network Services

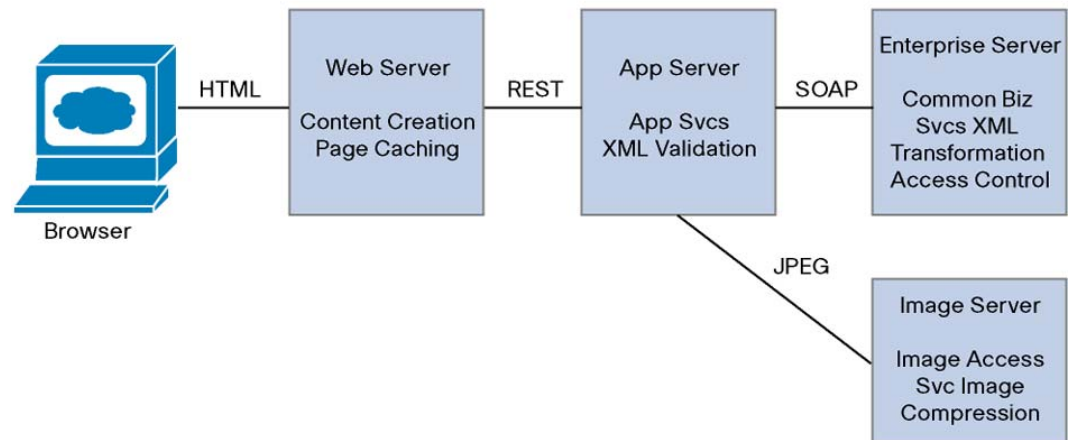
There are several advantages to locating certain services in the network. These include:

- Improved performance: Dedicated hardware can perform many tasks faster than software can.
- Shared effort: As the network assumes certain application responsibilities, the application server is freed up for the sorts of things that only it can do.
- Tuneability: Services located in the network can be tuned so that, for example, mission-critical applications receive a higher quality of service than other applications.

One challenge around putting services into the network is making architects aware of them. This is a layer of the technical architecture that EAs know little about and rarely venture into, and they may not realize what a properly provisioned network can provide. It is ironic that even as up-to-date SOA governance charges EAs with actively maintaining a catalog of the services that are the reusable building blocks of their enterprise systems, many services are going uncataloged (and thus unused) because they reside in the network and EAs don't know about them.

Here is graphic evidence that architects pay far more attention to servers than to the network: the kinds of diagrams that they draw. Figure 3, which represents an end-to-end cross section of the components that make up a web-application architecture, is a good example.

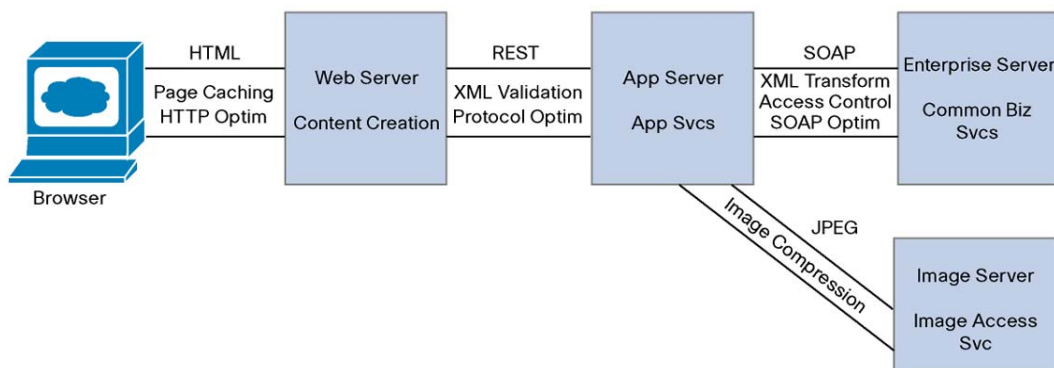
Figure 3. Typical web application profile



The labeled boxes that represent the behavior of the system are richly annotated; the thin lines connecting them, which represent the pipes through which message traffic passes, are minimally labeled. Architects, concerned with behavior, assume that the pipes are not of much interest.

This is no longer a fair assumption. Today the pipes themselves provide services that merit the EA's exploration and exploitation. A diagram reflecting today's distribution of capabilities would look something like Figure 4:

Figure 4. Network service-enhanced web application profile



The lines are now fatter, almost boxes themselves, and rich with services. Architects interested in behavior have a new place to look.

The specific services that they will find in the network are detailed in the following section.

EA's Guide to Network Services

This section catalogs the network services that will be of interest to enterprise architects. It explains the two types of services available from this infrastructure — one familiar to EAs and one not — and then discusses specific services one by one.

Exposed and transparent network services

Network services come in two varieties. The first, **exposed services**, is familiar to developers, especially those who build service-oriented applications. An exposed service is one that is explicitly invoked when the application needs that service performed. The interface to the service comes in many flavors but follows generally accepted interface practices. For example, there might be an interface to a WSDL-based web service (standard SOA service), a REST web service (a widely used, simpler interface), a standard Java component interface, or some messaging interface.

The other kind of network services is **transparent services**. These are unfamiliar to most EAs. A transparent service is not explicitly invoked by the application, but rather activates itself on an as-configured, as-needed basis. It is performed by the network automatically whenever a message configured for that service passes through. Although no program invocation is involved, architects still have to know about and work with transparent services. Architects specify configuration parameters that indicate to developers and provisioners when these services are activated. They may also, depending on the service, specify that certain “best-practice” formatting conventions have to be observed.

Take as an example the transparent service of XML validation: performing a syntactic analysis on an XML message to ensure that it satisfies the message definition. This is extremely useful, especially when XML is coming in from an outside source and may be ill-formed. It can be done in the application server software, but the operation is so expensive there that it is frequently disabled. Fortunately, there is a service in an XML-aware network infrastructure that can perform this validation “in the wire,” offloading the task from the server. It is enabled by setting a network configuration parameter and including a header that specifies the XML schema, the metadata that describes the proper XML format. Such a header, while optional, is recommended as a best practice and should already be in the plans of a conscientious developer.

Both exposed and transparent network services can significantly increase the quality of service of a high-demand distributed application. Both place a responsibility on the EA and the development team. Exposed services will be activated only if they are known about and explicitly invoked. Transparent services will be activated only if they are provisioned in and complemented with architectural best practices. It is well worth getting familiar with both.

Catalog of Network Services for Application Delivery

This section catalogs the services available via the application delivery network. The catalog is divided into six categories:

- Cache
- Compression
- Content distribution
- Content-based routing
- Protocol optimization
- XML processing

Each category includes

- A definition of the service category
- A list of available services, exposed and transparent
- An optional note about special considerations

Becoming familiar with these services, an EA gains the conceptual tools for integrating into them into their requirements gathering, solution design, and application validation processes.

Cache

Definition:

Replicate frequently requested content in close proximity to where it will be used.

Since the beginning of the web era, caching has been the first line of defense for a high-quality user experience. In Reach / Range terms, caching mitigates the pain of Reach by eliminating the real distance in a distributed interaction. It provides a double benefit by reducing subsequent bandwidth. The trade-off is that either the network infrastructure or the application must work to ensure that caches do not become “stale” when there are updates to the underlying content.

Exposed services: None

Transparent services:

There are several kinds. In some cases, best-practice message construction can increase the caching quality.

Static caching

The simplest form of caching. Specific resource types (for example, applets or images) are cached in specified locations. It is static because the resources (for example, a company logo) are immutable.

Adaptive dynamic caching

Caches web page content that is dynamic, such as a retail catalog that changes periodically. This dynamic content can be cached by specifying various attributes of the URL such as query strings, HTTP headers, or cookies.

Load-based dynamic caching

Another form of dynamic caching that is dependent on run-time conditions. It can be configured to sense content expiration via time-to-live events, load, timing, or URLs. Load-based dynamic caching provides a rich set of mechanisms for avoiding staleness.

Dynamic browser caching

A subtle transparent service that caches web browser resources that application software has marked as non-cacheable. These resources are frequently used for images, JavaScript, or binary content such as Flash SWF resources. The application delivery network's caching service can second-guess the non-cacheable choice and cache the resource in such a way that browser keeps retrieving the resource from the cache. The cache is automatically refreshed when the network components sense that the true resource has changed.

Caching aspect of delta encoding

A sophisticated optimization to web page processing that accelerates web page traffic, a high priority in improving the user experience. While web traffic seems to move across the network a page at a time, most sophisticated web apps are more complex. Accessing a page may involve moving multiple resources from server to glass, some static and others dynamic. Delta caching replicates page resources and transmits only the **differences** (deltas) between them. Instead of an entire page being shipped over the wire, only these deltas are communicated, and the network node closest to the browser reconstructs the page accordingly. Because so many adjacent pages in a web application share some portion of the previous page, the improvement in page-display speed can be dramatic. (This feature also appears in the Compression services category, because the underlying feature automatically collaborates between the Cache side and the Compression side by automatically compressing the transmitted delta.)

Special Considerations: Best-Practice consideration

A small amount of architectural attention to caching can vastly improve performance. Although some cache mechanisms (static caching and delta caching) need only to be enabled, others can benefit enormously by cooperating with the caching algorithms. Consider those that are sensitive to URLs or HTTP headers: By adding to these headers some cache-aware values such as message IDs, credentials, or expiration hints, an architect can help the caching mechanisms infer proper behavior. Furthermore, these headers operate with no changes on non-accelerated networks.

Compression

Definition:

Reduce the size of network traffic by encoding content in such a way that it requires fewer bits.

Compression provides many of the same benefits as caching. The trade-off is that compression uses up valuable processor cycles. This is why the compression capabilities that have long been available within app servers or applications are seldom used. Putting compression in the network can be far more efficient due to hardware accelerations, and costs no server CPUs at all.

Exposed Services:

None

Transparent Services:

Standard GZIP and DEFLATE compression

These are the same algorithms used in many computers, typically reducing bits transmitted by a factor between 2 and 5.

Compression aspect of delta encoding

Delta encoding, discussed above, can go further. After the deltas have been computed but before they are put on the wire, an optimized, delta-friendly compression algorithm further reduces the already small delta by 10 to 50 times.

Smart image reduction for configured usage

Images are among the largest resource drains on a computer. Compression can significantly reduce the network traffic. But not all uses of an image require the same kinds of compression; an image destined for a printer requires far more target resolution than one destined for a screen. Smart image reduction allows for a range of compression strategies based on the intended use of an image.

Content Distribution

Definition:

Reduces delivery time by prepositioning frequently requested content.

Exposed Services:

API to configure and monitor content distribution

This XML-based API supports dynamic configuration of content distribution, including information for channels, content engines, and content providers. Monitoring yields information about content engine status, replication status, and streaming and HTTP transaction statistics.

API to control distribution policy

The API (called the Manifest API) manages prepositioned resources and specifies their acquisition and distribution policies, content availability, channel creation and modification, and content engine channel assignment.

API to manage video specifics

This API (called the Program API) manages live or scheduled rebroadcast programs and controls the streaming engine.

Transparent Services:

Prepositioning of server data, metadata

Server content is prepositioned to configured destinations at the network's edge, near to where the resource will be used. Content can be unicast to a single point, or it can be multicast and broadcast to many destinations.

Prepositioning video

Streaming video support can be used to efficiently deliver movies to sites around the organization.

Content-Based Routing

Definition:

Reduces dynamic content response time by routing messages to the nearest point based on content.

Where content distribution reduces network traffic by prepositioning resources, content-based routing reduces traffic by ensuring that specified content goes directly to the service engine that is needed.

Exposed Services: None

Transparent Services:

Optimized routing

Content routing infrastructure examines the HTTP (web request) header or SOAP (web service request) header. It uses the information to optimize the path to the needed resources or to a cached version of the resource.

Protocol Optimization

Definition:

Decrease bandwidth usage by reducing the number of roundtrips required for an application transaction.

After various forms of caching, compression, and content management, another increment of performance can be squeezed out of network traffic by optimizing specific generic protocols, or even specific vendor-implemented protocols.

Exposed Services: None

Transparent Services:

TCP optimizations

The TCP protocol optimization improves the utilization of the underlying network. This results in significant reduction to network bandwidth for any TCP-based traffic. It also improves network utilization, making remote access look more like LAN access, in effect reducing the pain of Reach.

Application-vendor validated protocol-specific acceleration

Increasing the application fluency to the individual message formats offers further network efficiencies. These optimization techniques reduce the effects of network latencies and ensure intelligent cache strategies. Application protocols for Microsoft file systems, Microsoft Exchange, and SAP can use protocol optimizations.

XML Processing

Definition:

Increases application performance by doing processor-intensive XML processing in the network.

This family of services is different in kind from the other application delivery network services. The other services, perhaps excepting protocol optimizations, provide fairly low-level services to technical resources in the application. Low-level means that the resource

vocabulary (image, applet, HTTP message, and so on) is primarily about the technical elements of the software application. XML goes to a higher level. It is the message carrier for the principal application messages and transactions themselves. For example, an XML message might represent a purchase order, a time card, or a patient history.

XML is so widely used (it's considered the lingua franca of SOA) that acceleration at this level represents a major win for high-performing, high-demand, long-Reach, broad-Range applications. This is especially true because a significant amount of application server processing time is devoted to the XML operations discussed here; an Application Delivery solution can recover up to 90 percent of server resources related to XML.

Exposed Services:

API-based provisioning of XML behavior

The provisioning for the XML processing described here can be dynamically specified by an XML-based configuration API.

Translation from one XML format to another

Translation from one XML message format to another format (for example, breaking a single SalesOrder XML message into a number of LineItem XML messages) is a common operation, often done multiple times in a single business transaction. Custom translation rules are contained in another XML file called an XSLT document that, when applied to an input XML document, generates an output XML document.

Transparent Services:

XML validation against an XML Schema

XML Schema was added to the XML family some time after XML was originally designed. XML Schema is an XML file whose job is to hold the metadata for an XML document. XML Schema is to an XML document what a database schema is to a relational database. For example, an XML Schema for the SalesOrder document would describe the shape, content, structure, and even some semantics of the SalesOrder document.

XML validation is useful but expensive. This service validates the XML document in the network on the way to its destination. Like so many application delivery network features it has a double benefit: it's faster and it offloads some of the demand from the app server.

XML Schema is not required to process XML. So this feature has to be enabled during provisioning and the to-be-validated document needs to contain in its own header a reference to the XML Schema that governs it. Using XML Schema references is another example of an architectural best practice enabling an application delivery network feature.

Access control and Policy

Data privacy and service access can be controlled through enterprisewide policy servers implemented in LDAP, Active Directory, and others.

Threat mitigation

Protections built into the network can mitigate various XML threats, including identity, content-based, message transport, and XML denial-of-service threats.

Routing

Dynamically routes XML based on content and context. Destinations can be determined via a variety of rules and parameters that inspect content of the message.

Encryption and signing of XML documents

This secures access to applications while maintaining confidentiality and message integrity.

Re-visioning the enterprise architecture work process

Becoming acquainted with the exposed and transparent services available in the network is an important step for enterprise architects. To really get the benefits of these services on an ongoing basis, though, it is important to incorporate them into the EA's day-to-day work habits.

Many aspects of the enterprise architecture process are affected by the existence of network services. This section summarizes the phases of designing a system and points out the impact of network-service awareness at each phase. As the enterprise architecture job varies from organization to organization, and also depends on whether an organization follows one of the many architectural frameworks or has crafted a custom EA process, this discussion is by necessity quite general.

Phase 1: Application Requirements

The first phase involves discovering the needs of the business that a new or reworked application is intended to satisfy. At this phase, any technology-related or solutions-related thinking is to be avoided. The documents produced should be completely understandable to the business people in the organization. In fact, the primary value of requirements documents is to promote conversations between the architect and the business people on one side, and the architect and the development teams on the other side. These documents include:

- **Use-case diagrams**

Describe interaction between users (actors) and system. These cartoonlike drawings clearly express the kinds of action that each actor is expected to perform.

- **Process models**

Chart the progression of a business process (usually as swimlanes). The swimlane diagram is an effective communication vehicle between business expert and architect as to the precise structure of actions performed.

- **High-level information model**

Summary description of data as seen by the business (not the system). This should not be confused with the detailed data model done in design. It expresses the business-view of data, using business vocabulary.

Because the requirements phase is expected to divorce itself from any solution, it is not a phase where network-service awareness would seem to have any impact. However, it is critical to recognize as early as possible when the system being designed is going to be a high-demand distributed application that will require an exceptional level of performance. Even thinking in business / requirements terms, it should be clear when:

- There will be long Reach (all field offices in the country will be using it)
- There will be broad Range (digital photos and real-time data feeds will be incorporated)
- The user community will be large
- The user community will expect a high quality-of-experience

These factors should be written into the system requirements so that developers can, from the start, focus on approaches that can deliver high performance.

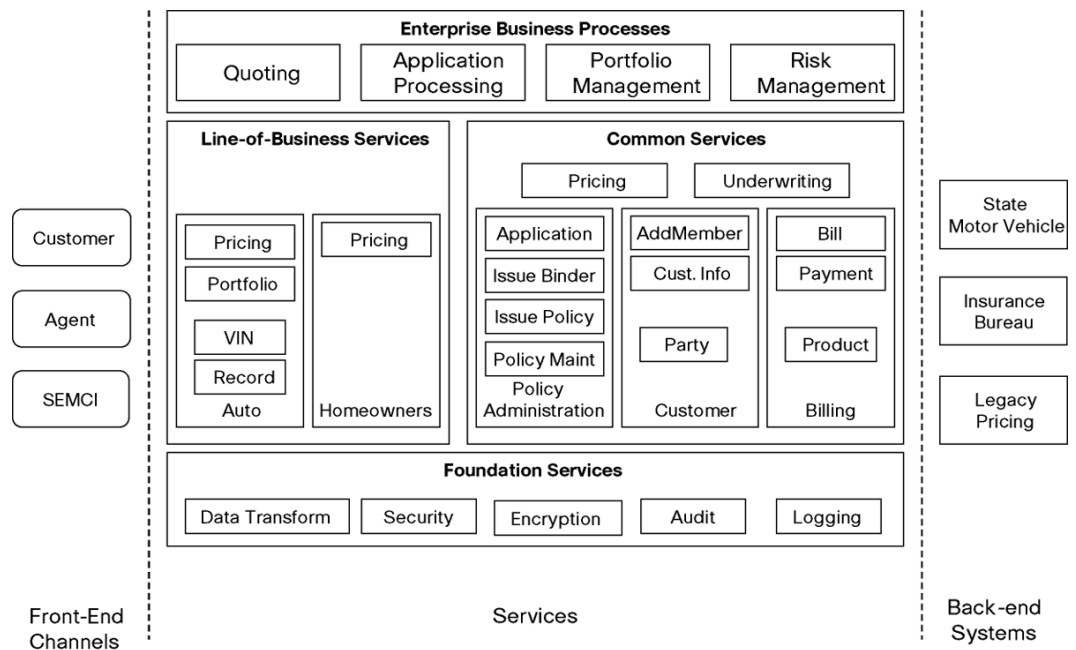
Phase 2: Solution Design

This phase focuses on designing a concrete solution. The process usually proceeds from a high-level, abstract view of the implementation down to the concrete details. The kinds of artifacts produced give a good idea of the progression:

- **Conceptual architecture:**

Layered components of the application, as shown in Figure 5. This is commonly referred to as a “fireplace diagram.” The left side shows consumers of the architecture and the right side shows resources that the architecture relies upon for data, services, or other processing. The center of the diagram shows the various services of the architecture. The lower-level services are more generic and act as a foundation to other services. The mid-level services are either common services shared among many stakeholders (for example, across Lines-of-Business). Also shown are more application-specific services (for example, per Line-of-Business). At the top are the enterprise applications or processes that consume the services.

Figure 5. Conceptual architecture diagram



- **Application profile**

End-to-end visualization of distributed system’s tiers. This gives a comprehensive view of the operational interactions of the applications processes. (See “Sample Profile” below).

- **Detailed process models**

Design-time process models including application steps as well as business steps. These are swimlane diagrams similar to the requirement-time artifacts mentioned above. However there is additional detail that reflects interactions between systems, error handling, and transaction boundaries.

- **Detailed data models**

Design-time data models depicting the full data model of the application. They include the system-support data as well as business data, and serve as models for the actual database structure.

- **Service discovery and design**

Inventory of services to be built or acquired. This is a list of services and their detailed requirements.

- **Class diagrams**

Detailed object-oriented design diagrams. Class diagrams show the technical design of the

components of the application or service. An EA may get involved with the more challenging design elements or may delegate it to the development teams.

This is the crucial phase for designing network services into high-demand enterprise applications. There are numerous touch points in the design that would point toward network services. An EA should:

1. Scan the **Conceptual Architecture** (Figure 5) for any foundation services that could be provided in the network. If there is a need for, for example, encryption services, the EA can refer to the network service catalog in the previous section to see if such a service is offered. If no exact match is found, it might be worth changing the conceptual architecture to take advantage of a service that does exist. For example, the data conversion foundation service called for in Figure 4 could be implemented by using the XML translation network service. Network services are so useful in a high-demand environment that it makes sense to use them as much as possible, even if this means adding to or modifying the design concept.
2. Take the **Application Profile** diagram and on it mark the communication protocols that connect adjacent boxes. If the EA sees “Http,” he or she knows that delta encoding and protocol optimization are available to radically speed up the traffic in that pipe.
3. Assist team leads in the **Service Discovery** phase. This is where the most concrete expression of an application’s needs appear. Look carefully to see if any of the service needs can be satisfied by a network service.
4. Make sure that, if accelerated behavior is one of the goals, the design includes as many network-layer services as possible.

Phase 3: Implementation

Even after the design phase is completed, the architect should remain involved. Activities might include:

- Consulting
- Validating
- Advising on test strategy
- Project guidance and review

An EA will want to make sure that the developers have gotten the message on network services and are invoking them properly.

The EA will also want to encourage early, frequent validation and testing. Network services make this easy. Unlike most hand-built services, network services have excellent built-in monitoring tools integrated into the existing network and system management infrastructure. So, for example, it is possible to precisely measure the performance impact of validating inbound XML documents from external suppliers. In fact, it is possible to go further and tune the XML service priority levels on a particular virtual port to ensure that an appropriate level of performance is provided.

An EA needs to recognize these new capabilities and responsibilities and communicate them to the organization.

Phase 4: Updating the Standard Operating Environment

An architect’s final job is to update the enterprise’s institutional memory with the new learning attained during the course of the project. Enterprises that have attained architectural maturity will maintain some form of database for the components, tools, even software patterns and skill

definitions that have been adopted. This is often called the Standard Operating Environment and usually has two parts:

- A catalog that records the tools, components, and infrastructure that the organization uses. Sample entries might include standard DBMS, development tools, frameworks, software libraries, service libraries, tools, PC standard software, and so on. Many organizations have a managed technology-introduction process that ensures the quality of any proposed catalog entry.
- Standard application profiles, which might include several profiles approved for the organization, such as Web 1.0 app, rich internet app, centralized processing, or fat client/server. An example of an application profile is shown in Figure 6.

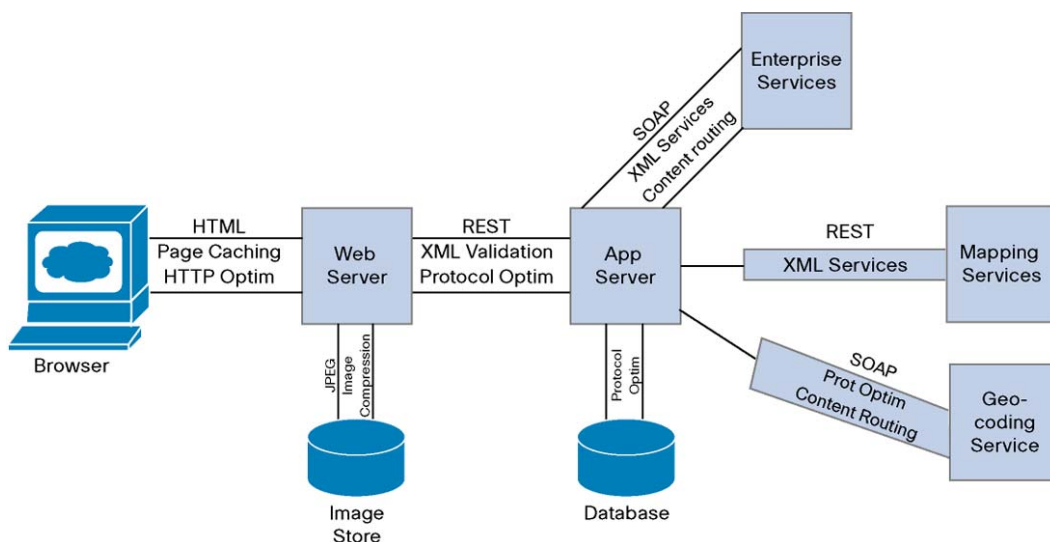
If an EA finds that network service awareness is important for delivering high-performance distributed applications, he or she should write this into the Standard Operating Environment. The specific services recommended should be added to the catalog, and the standard application profiles should include one or more end-to-end profiles (such as Figure 6 below) showing application delivery network components in context.

This document has described at length what an EA needs to learn to become network service aware. Recording this new knowledge into enterprise memory preserves and multiplies the effort.

A Sample Profile

Figure 6 shows an application profile of a web-based application. This particular application relates to retail banking, but the point is how it pulls together concepts of both enterprise architecture and network service awareness.

Figure 6. Application profile of an extended web application



The focus here is on the connectors between layers, discussed below from left to right:

Browser to Web Server

Browsers are connected to web server farms that serve up the web pages.

Protocol: HTTP

Network services: Caching and compression with delta encoding HTTP protocol optimization

Web Server to Image Store

Image store provides images.

Protocol: JPG

Network services: Compression with optional smart image reduction

If distant image server, content distribution to preposition images

Web Server to App Server

App server implements mainstream application-specific business services. The REST web service interface, shown here, puts XML directly into HTTP. This is faster than the alternative, SOAP, but has fewer architected features.

Protocol: HTTP / XML

Network services: Protocol optimization with HTTP

XML services (validation, transformation)

App Server to Enterprise Server

Enterprise Server implements enterprisewide, more generic services used by many application servers in distant parts of the organization. SOAP is used for extra robustness plus security.

Protocol: SOAP / XML

Network services: Protocol optimization with SOAP

XML services (validation, transformation, access control)

Content-based routing to get directly to right app server

App Server to Database

App server connects to the application's database.

Protocol: Database proprietary

Network services: Protocol optimization for TCP

App Server to Third Party Servers

Various services providers are used. A credit agency checks the creditworthiness of a client. Yahoo Maps provides a map to the nearest banking center. A geocoding service computes a route.

Protocol: SOAP / REST / HTTP / XML

Network services: Protocol optimization with SOAP, HTTP

XML services (access control, encryption)

Content-based routing to get directly to right app server

Conclusion

As Reach and Range continue to expand, the high-demand distributed applications of the future are going to require continuing innovation in architectural techniques, resources, and capabilities. New performance boosts will have to be found at every level of the system, from the lowest-level plumbing to the highest-level user interface. Enterprise architects will be working far beyond the app server, getting involved with services from many sources.

What has been happening at the network infrastructure level is exemplary. In addition to their primary job of moving bits through the wire, networks have begun dealing in higher-level,

application-related semantics. They now offer capabilities such as data compression, caching, content distribution, and protocol optimization — all at a point in the system that may be more efficient and less taxing than similar capabilities in the server. For the enterprise architect, this means expanded options and more flexibility.

Any EA whose organization is looking for high-demand, media-rich integrated applications would do well to become acquainted with the new, application-fluent network. The capabilities found there will almost certainly play an important part in the enterprise architectures of the future.



Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV
Amsterdam, The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

CCDE, CCENT, Cisco Eos, Cisco HealthPresence, the Cisco logo, Cisco Lumin, Cisco Nexus, Cisco StadiumVision, Cisco TelePresence, Cisco WebEx, DCE, and Welcome to the Human Network are trademarks; Changing the Way We Work, Live, Play, and Learn and Cisco Store are service marks; and Access Registrar, Aironet, AsyncOS, Bringing the Meeting To You, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, CCSP, CCVP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Collaboration Without Limitation, EtherFast, EtherSwitch, Event Center, Fast Step, Follow Me Browsing, FormShare, GigaDrive, HomeLink, Internet Quotient, IOS, iPhone, iQuick Study, IronPort, the IronPort logo, LightStream, Linksys, MediaTone, MeetingPlace, MeetingPlace Chime Sound, MGX, Networkers, Networking Academy, Network Registrar, PCNow, PIX, PowerPanels, ProConnect, ScriptShare, SenderBase, SMARTnet, Spectrum Expert, StackWise, The Fastest Way to Increase Your Internet Quotient, TransPath, WebEx, and the WebEx logo are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0812R)