Table of Contents for Chapter 12

<<< … >>>

Part 2. Architect's Toolbox - 'bird's view of the terrain'

<<< … >>>

Chapter 12. Enterprise Computing and Integration

Enterprise IT Architecture is a complex system requiring most respectful due diligence in designing, building, implementing and running it in support of the core business of the given Enterprise. (Yes, as heretic as it may sound to some, Enterprise IT Architecture exists for the Enterprise, not the other way around).
Sheer volume and variety of business requirements, distributed context of operating environment, multitude of stakeholders, choice of platforms and technologies, organisational politics, industry and technology trends – all combined may seem to present a daunting challenge to the Enterprise Architect.

Enterprise Architecture 101

Enterprise Architecture should be able to support current business imperatives, and to morph with the business in whatever direction Enterprise chooses to evolve in the future.
Competitive edge and very livelihood of the modern Enterprise may depend on its Enterprise Architecture.

Earlier chapters outlined tools of trade and building blocks of the Enterprise Architect – methodologies and standards, technologies and platforms, components, integration techniques and patterns.
Enterprise Architecture is where all these pieces of the knowledge base come together.

Fundamental building blocks and technologies are in the core of any elaborate Enterprise Architecture or product offering. Understanding of underlying technologies gives Enterprise Architect ability to make an informed decision on selection of product and vendor that suit business goals and operating environment of your Enterprise best.
In addition to the good grasp of the overall 'big picture', Enterprise Architect should be able to look 'under the hood' of the Enterprise solution to ensure cohesion and smooth integration of platforms and components.
If nothing else, understanding of internal workings of the product lets you cut through the hype and smokescreen of sponsored white papers and promotional materials. Often, it is what was not said (or, intentionally glossed over) may be more important in your feasibility analysis.

This chapter focuses on some core patterns in building Enterprise Architecture, and points you to resources for more elaborated treatment of the subject, possibly platform- or vendor-specific.

Building Enterprise Architecture is markedly different from the *software development*.

Main emphasis in the Enterprise Architecture is shifted from conventional construction of software products, packages and libraries towards *integration*, putting together a cohesive system from sometimes seemingly disparate components.
Enterprise Architecture primarily concerned with identifying main components and processes, positioning these components in the Enterprise and against each other so that complex Enterprise Architecture as a whole provides a good support to the core business, and does it efficiently (hopefully, better than competition).

Building Enterprise Architecture is a balancing act of finding the workable and feasible compromise between two ever-present conflicting forces or tensions in the Enterprise:

- Need to achieve *loose coupling* of components or *separation of concerns*, and
- Enterprise-wide *cohesion* in shared management of data and transactions

On one hand, we attack complexity by de-composing the complex system through layering, segmentation, partitioning, and breaking it into smaller, more manageable and better-known components. And, we repeat this de-composition process of architecture analysis and design iteratively, until all identified interfaces and components are clearly 'doable', or may be procured 'off-the-shelf'. Having many points of control in the architecture gives more options for integration and enhancement solutions.
Monolithic architectures may perform specific tasks well, but can present an integration challenge when enterprise finds likely gaps, and then attempts to enhance capability of monolithic architecture outside the prescribed paradigm.
If 'monolithic' or 'closed' architecture serves you well, and, you think, will continue to do so for the lifespan of your application, go for it. However, this comfortable situation is becoming increasingly unlikely in the modern large, diverse and agile Enterprise.
Enterprises strive to deploy corporate content and services in places where they are more needed at the time when highly mobile customers request them, thus achieving better performance and availability of the service.

On other hand, indiscriminate distribution and segmentation of corporate data and processes may create issues of data currency and integrity across the Enterprise, as well as manageability and transactional assurance (see *ACID* properties of transactions) in complex distributed business processes.
Single Data Warehouse or database can save a lot of grief when you have a single and reliable shared source of your precious corporate data, and a single copy of data to be updated under the strict assurance of the DBMS's two-phase commit protocol. You have to appreciate well what advantages you leave behind when you venture into the wild distributed, segmented and layered world of the Enterprise Architecture.
Some data integrity and manageability issues in the Enterprise may be alleviated by going back to basics, i.e. consolidating data and transaction processing back in large data centres, on the pooled server farms or powerful mainframes. That consolidation can be done separately for particular layer or segment of the Enterprise Architecture.
In your designs, you should remember why your Enterprise did distributed architecture in the first place. For instance, distributed architecture prevented database lockup in case of massive updates, or, possibly, it increased performance and availability of service to the customer when it counts, or eliminated the Single Point of Failure to ensure business continuity in case of natural or man-made disaster.

Architectura Sapiens Revisited

In Chapter 6, we followed Enterprise Architect's train of thoughts in iterative process of building the specific Enterprise solution.
This section will recap, with more generic view, the fundamental patterns in designing scaleable multi-tier enterprise-grade technical architectures - main iterations, layers and components.

We are not saying here that every site has to evolve through every level of growing complexity and capability of architecture. Enterprise context and requirements may necessitate starting 'big' from day one. Another Enterprise may present a challenge of upgrading the architecture in order to keep up with changing and growing business, without disruptions to the core business.
However, it is important for Enterprise Architect to see how architecture can be de-composed into components and layers, and what levels of complexity in putting them together ought to be considered before technical architecture takes its final shape.

Presented evolving patterns for the deployment of Enterprise Architecture do not describe only, or the most comprehensive approach in architecture design, but rather common considerations and challenges facing the Enterprise Architect.

Architecture Embryo – '… if it was that simple'

Early Web sites did just fine by serving static HTML pages in response to HTTP request.
CGI programs performed all presentation and rendering logic, business logic and data processing that was required to fulfil requested business function. Web Server was the only piece of architecture that mattered. Not anymore…

Even most basic Web sites will have distinct layers or tiers – *Presentation*, *Business* or *Domain Logic*, and *Data Source* layers to start with.
However, even in simplest architectures things may get tricky if we factor in increased requirements for security, high availability and integration with other services or legacy applications. Also, simple logical architecture still presents numerous options for mapping of logical components and layers in the deployment architecture onto physical servers and network devices. And, spare the thought for simple Web sites that are outsourced and use the shared hosting facility – you may wish to check SLA with this hosting facility against your business requirements, e.g. for security and availability.

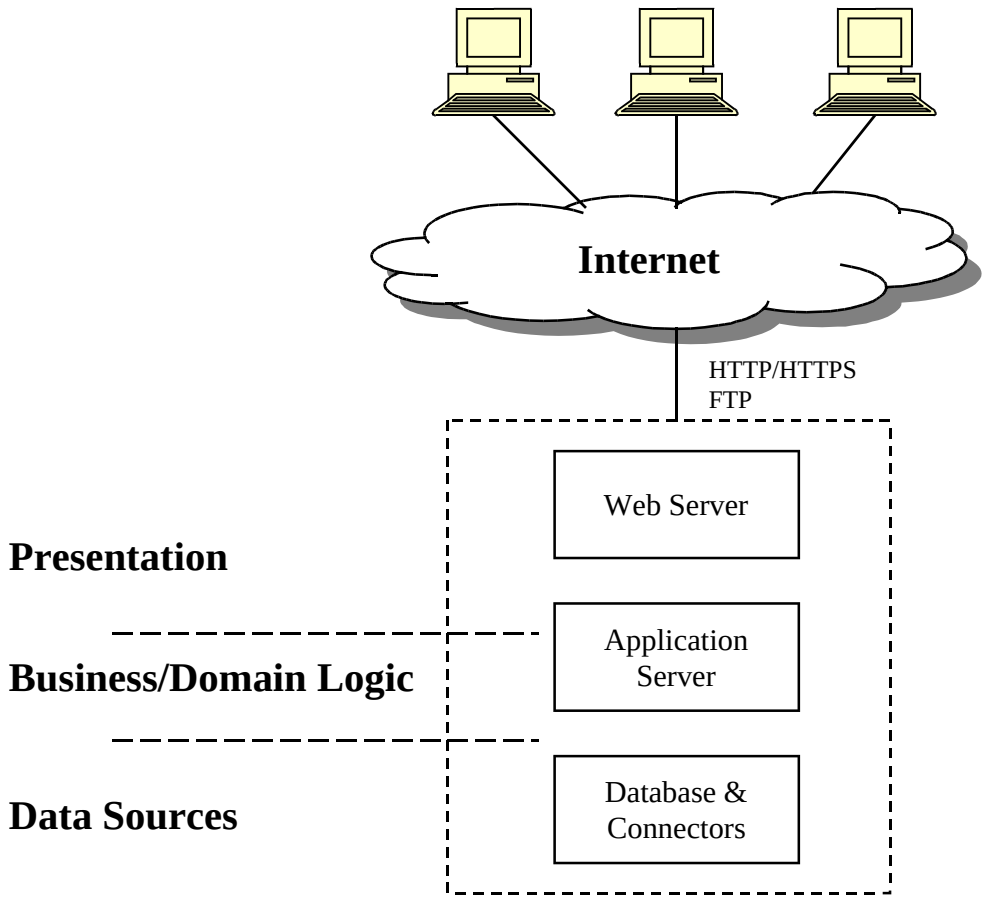

Figure 12.1. Step to the Web

On the diagram, *Business Logic* layer occupies middle tier of the architecture, and represented by the *Application Server*.

Note subtle detail concerning the demarcation line between *Presentation* and *Business Logic* layers. We recognise the fact that it is sometimes very difficult to distinguish *Presentation* from the presentation-related *Business Logic*, and the latter ftom the *Business Logic* 'proper'.

Role of HTTP Web Server in modern Enterprise Architecture has been narrowed down to front-door gatekeeping activities:

- Authentication and, possibly, coarse-grained authorisation and access control
- Actual communication with the client over the Internet. This may include in part Session Management to ensure 'sticky' connection to allocated Application Server for the duration of session over stateless HTTP connection
- Content delivery and caching
- Routing of incoming requests downstream for further processing on internal layers of the Enterprise Architecture. Possibly, load balancing of incoming workload between several instances of Application Servers

Application Server performs bulk of the *Presentation* and *Rendering* for the application, as well as presentation-related *Business Logic*, and *Business Logic* 'proper'.

De-coupling of *Presentation* logic from the *Business Logic* 'proper' is very important high-level pattern of the modern pervasive Enterprise Architecture. In the core of this pattern lies our ability to separate *Content* from the presentation styles and capabilities (like limitations on rendering capability or capacity of various client devices). In addition to separating content from presentation, our application should be able to leverage this separation to the full extent in its architecture.

Business Logic should deliver desirable *Content* to the *Presentation* layer, and *Presentation* layer should be able to *re-purpose* this content without changes to the *Business Logic* layer.
Typical examples of content re-purposing are:

- Re-formatting of the content for different views on the web page
- Personalisation of content for different customers, and
- Content transformation for different client devices

Benefits of separation of *Content* and *Presentation* are easily understood if we imagine common scenario of Internet application serving customers using different client devices, of different makes and models. Enterprise desires to reach its customers on any device of any type (possibly available now, or in the future), anywhere and at any time customer needs service. Idiosyncrasies of different browser version or new type of mobile device should be encapsulated in the *Presentation* layer of your architecture.
XML and XSL Transformation are the integral part of your solution in achieving separation of *Presentation* and *Content*, and of the *Business Logic* delivering this *Content*.

Business or *Domain Logic* layer encapsulates behaviours and idiosyncrasies of the application domain, its 'normal' procedures and special cases that real-life scenarios supply us in abundance. Martin Fowler calls this seemingly convoluted but realistic knot of business rules a "complex business 'illogic'" [Fowler, p.4]. This behavioural complexity is a compelling enough reason to insulate 'business illogic' in a layer separate from *Presentation* and *Data Sources*.

Even in simplest cases of rendering some basic content through the *Presentation* layer straight to the customer, we need to keep in mind this fundamental three-layer pattern, and allow for business logic to be captured in its own layer where it belongs.
Even if your application started small, with some rudimentary functionality, enterprise application is likely to evolve during its life cycle. Good discipline in adherence to three-layer pattern while implementing new business logic and numerous exceptions will preserve cohesion of your Enterprise Architecture and prevent the 'architecture rot'.

Data Source layer represents the storage and supply of raw content required for executing the domain business logic and for presenting transformed content for customer's perusal.

Typically, *Data Source* layer implemented as databases, content management systems, or interfaces to other enterprise applications.
Hence lies the understated beauty and power of the fundamental three-layer pattern – it can (and should) be applied to every component, sub-component, and to the whole system recursively. One component's presentation is another component's data source (we assume that 'presentation' is used here in the broad sense of delivery the content to the consumer in general, and not just on the display of the client device for visual pleasure of the customer).

Bouncing Baby – 'long road ahead'

In the Enterprise Architecture, distinct logical layers for *Presentation*, *Business* or *Domain Logic* and *Database* or *Data Sources* (or, in a broader sense, layer for integration with back-end or legacy applications) will be mapped onto physical deployment layers.
Architect may face variety of options in mapping the logical architecture on physical servers. However, in Enterprise-grade solution that needs to address security, load balancing and high availability issues, logical layers will give you a good idea on how you will partition your application.

Figure 12.2 still presents a high level logical view of layers, but physical deployment diagram will likely be very similar.

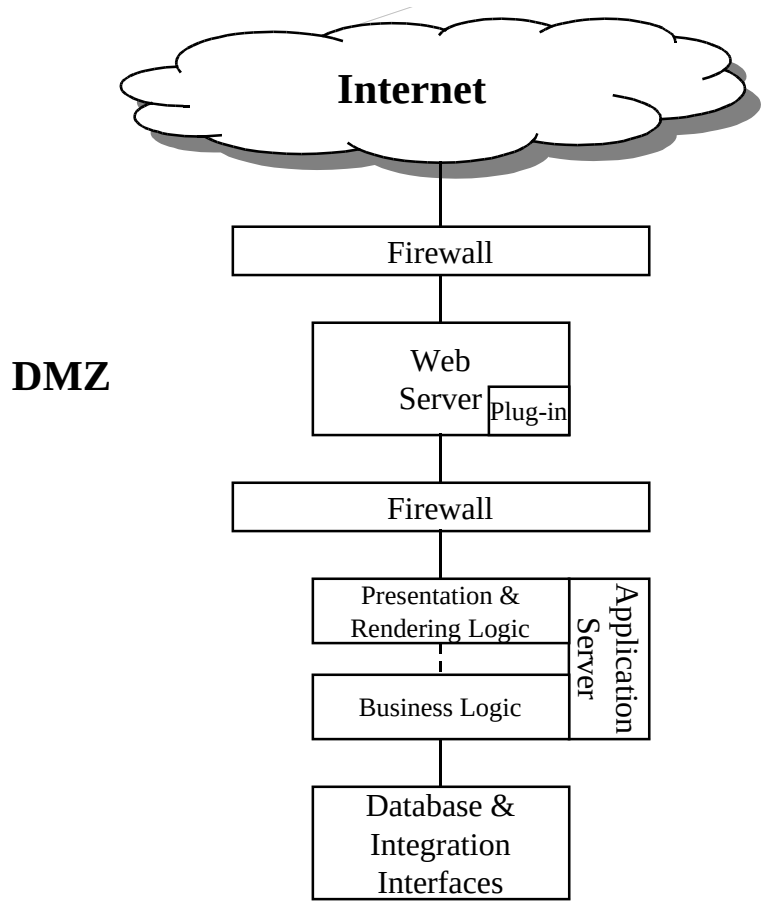

Figure 12.2. Layers and Firewalls

Internet is a jungle. As soon as you open the entrance door of your Enterprise to the 'Wide Wild World' of the World Wide Web, expect the broad range of behaviours from your customers, curious innocent onlookers, competitors, or malicious perpetrators.
Valuable Information Assets and Services of your Enterprise need to be protected from unwanted attention. On the other hand, your Enterprise should be able to adapt easier to changing usage patterns, and to deliver business functionality to the client in adverse, or even hostile, circumstances.

Every physical layer in your architecture is a potential candidate for applying security and load balancing techniques, depending on specific to your enterprise threats, volumes of traffic, and requirements for availability and performance. Every layer may potentially become a Single Point of Failure (SPoF).
However, there is one obvious point in architecture where common high level security and scalability deployment patterns *must* be applied – at the 'entrance door' to the Enterprise. In other words - where external traffic first hits your Enterprise Architecture. The further inside the Enterprise fortress you let unwanted traffic – the greater risks and impacts of negative exposure. First defence line of the Enterprise should be able to absorb most or all of the security threats, Denial of Service (DoS) attacks, or surges in legitimate access traffic.

Analogy with warfare does not seam farfetched if you consider criticality of Internet-based services to the core business of your Enterprise, and malicious threats from all your potential adversaries. We will exploit this analogy further to explain the motivation for common deployment patterns.

On Internet, the 'entrance door' into your Enterprise is the edge of the Enterprise's Intranet, as well as Web Servers and other application gateways facing the Web. (Strictly speaking, thick client, caching and routing facilities of your ISP on a path towards Enterprise Intranet may be considered in part an 'entrance door' into your Enterprise also).
These elements of the Enterprise Architecture face the first wave of 'assault'. This is where Enterprise first attempts to filter traffic, secure access, apply load balancing and performance-enhancing techniques (like caching), and do all of this with as less exposure of Enterprise internals as possible.
In a sense, edge of the Enterprise Intranet and Web Servers are the 'sacrificial lambs' – if worst comes to worst, damage should not percolate deeper into the Enterprise Architecture.

Usually, Enterprises deploy network routers, load balancers, caches and firewalls to provide initial protection of and access to Web Servers.
Ideally, Web Servers should not contain business logic or direct access paths to the Information Assets in the back-end of the Enterprise (like databases and interfaces to legacy applications) – this is the job for the Application Server.
Thus, Web Server becomes a lightweight architecture component that will host plug-ins for connections to the Application Server and for high level Security and Access Control functions (like user authentication, coarse-grained authorisation and SSL end-point).
If Web Server is compromised, there is second firewall to contain the damage to Web Server and to prevent the spread of the negative impact further inside the Enterprise.

Appropriately, network segment between two external firewalls is called Demilitarized Zone (DMZ), where we do not want bad things to happen, but can get by if they will.

As a rule of thumb, no sensitive business logic or data should be deployed in DMZ – only what is absolutely necessary for routing the requests for further processing to the Application Server.
Inbound traffic from DMZ is carefully monitored and restricted on need-to-have basis.
Outbound traffic to DMZ is contained to legitimate responses to client requests, as well as one-way publishing of content and code updates outside the firewall onto Web Servers and cache proxies.

As a result, distinctive feature of the enterprise-grade architecture is a physical separation of Web Server and entire Application Server, including presentation-related *Business Logic*, usually implemented in the Application Server.

For instance, in J2EE architecture this means that Web Container of the Application Server, including all JSP and Servlets, will be deployed on the box separate from Web Server. Web Server will be connected to J2EE Application Server's Web Container via HTTP plug-in through firewall.

JSP and Servlets (or alternative Java presentation solutions like Struts framework) will implement whole of the *Presentation* layer of application architecture and, possibly, some of the *Business Logic*. Note, mixing *Presentation* and *Business Logic* is undesirable and needs to be avoided by consistent use in application design and development of J2EE Patterns and Best Practices.
Connection to databases or external interfaces in the J2EE Application Server can be performed either from Java or JavaBean code in Web Container, or from the EJB in the EJB Container. EJB Container may be deployed on the separate box also, and be protected by the firewall or traffic filter.

In essence, end-to-end architecture described in this section is capable to deliver all functions and integration requirements for the small to medium Enterprise. Greater volumes, integration diversity, security and availabilty requirements of large Enterprises are yet to be addressed.

Promising Teenager – 'no fear'

It would be a safe bet to say that business demands of the modern Enterprise cannot be satisfied by the single, self-contained application or COTS (Commercial Off The Shelf) package, without the need for integration and alignment with other components of the Enterprise Architecture.
Or, if you think you found such a comfortable solution, in all likelihood your comfort will not last for long. Ever changing technology and dynamic nature of the core business surely will challenge your Enterprise Architecture soon enough.

Openness, extensibility and ease of integration have become a vital feature of the Enterprise Architecture.
Interfacing and integration between two architectures or components may be implemented in various points in your Enterprise Architecture, on different layers, using different technologies and protocols. When we provide interface between two islands of business components or technology in the Enterprise Architecture, we may use such terms as connectors, adaptors, wrappers, bridges, gateways, API, plug-ins, agents to name a few – all these are kinds of integration interfaces.
Intuitive understanding of these terms will suffice in most discussions between various stakeholders in Enterprise Architecture.
Integration Interfaces represent a pragmatic application, plain-English incarnation of the notion of the *public interface* of the object in object-orientation. As in object-oriented design, integration interfaces implement and execute *contract* for the public interface of the business component.

Figure 12.3 focuses primarily on variety of possible integration points in your Enterprise Architecture. Figure does not show all possible points, layers and technologies for integration, but rather high-level depiction of most likely Integration Interfaces, as a frame of reference for other possible options and variations for integration.

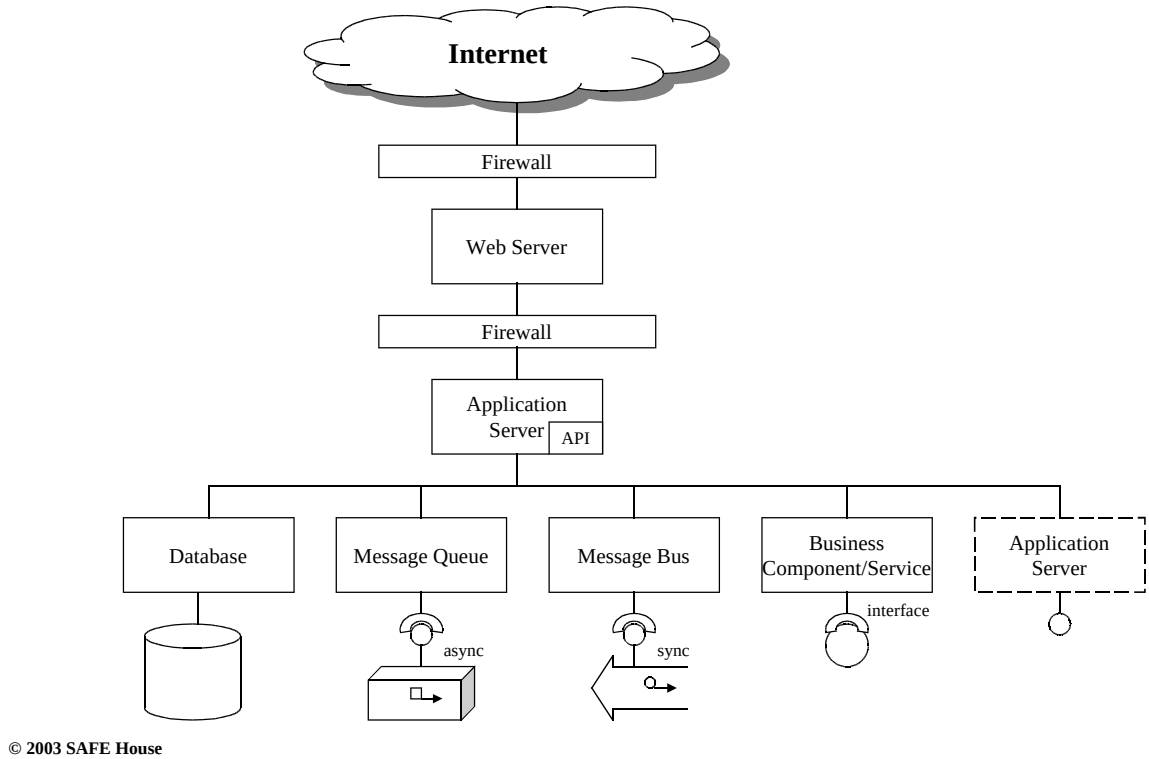

Figure 12.3. Integration with Interfaces, Connectors, Adaptors

Calling the external process or tapping into database may present a formidable challenge in itself. However, with more or less development efforts, any software product can be made accessible from any point in your Enterprise Architecture if necessary. For instance, you could call back-end EJB or COM object directly from the client's browser on Internet, but good design practices and security issues will likely steer you towards some other solution.

Integration Interfaces are not simply about providing technology for connecting two services in any way possible (which is not a bad start by all means), but about integrating on the right layer of architecture, using right technology, in compliance with many requirements for security, performance, transactional and data integrity, manageability, extensibility, total costs throughout the life cycle etc. Yes, Integration Interfaces address the need for connectivity in the first instance, but also aim to eliminate the *impedance mismatch* between technologies, protocols, platforms, data structures, timeliness and business logic of the processes on the client and server sides of the Integration Interface.

Coming back to the Figure, main types of Integration Interfaces are:

- Database access. This may include interfaces like JDBC, ADO.NET or database stored procedures for accessing relational data sources via SQL, or APIs for Document and Content Management Systems
- Message Queues for asynchronous publishing and exchange of messages, like MQ Series, MSMQ, Sonic MQ and others. In Java, Message Queues typically will be wrapped into JMS interface
- Message Buses for synchronous exchange of messages between clients and servers on the bus, like CORBA, DCOM, TIBCO, SeeBeyond
- APIs, Agents, Adaptors, Connectors etc., typically provided by vendors of software products and suites for integration with selected target platforms. For instance, SAP, PeopleSoft or Siebel adaptors and connectors, or Netegrity SiteMinder Web Agents
- Web Services for interoperability of loosely-coupled components and architectures
- Purpose-build or customised APIs for integration of components and technologies

Programmatically, any type of the Integration Interface can be implemented in any point of your Enterprise Architecture. However, Application Server is the most likely place where your integration efforts will be directed.
Note that integration is an iterative and recursive process. If we follow the control and data flow through the Integration Interface, we find that service on another end of the interface also runs on Appliacation or Database Server, and has Integration Interfaces of its own, connecting this service to other services.

Young Adult – 'can do'

Complete end-to-end Enterprise Architecture will have identifiable common core components, layers and integration interfaces as discussed in previous sections.
In this section, we take a look on how some complementary but vital Enterprise components fit into overall high-level architecture. We focus on such functions like Security, Identity Management and Access Control, and deployment of Portals or Product Suites.

It will be a very good sign if your chosen Application Server platform can become a lowest level denominator where the new Package, Portal or Product Suite can be deployed without the need to change underlying core infrastructure.
Despite advances in multi-platform protocols and technologies, cohesion in the technical architecture greatly eases the pain of integration and support efforts. We can build heterogeneous architectures where we mix and match 'fit for purpose' pieces of hardware and software, or find in other technologies missing pieces that close gaps in required functionality. However, we should not abuse our ability to integrate and adapt.
Painless integration, efficient and streamlined support, and containment of skills and costs represent a strong argument for consolidation of preferred platforms, products and vendors.

Enterprise may have settled on preferred platforms and products, but aggressive bundling or lack of standards compliance from some Product Suite vendor may challenge this committment.
For instance, you may have to bring in product that does not run on your chosen .NET platform (likely due to the lack of Java support). Or, if your Enterprise committed to J2EE technology, bundled product may not be certified to run on the Application Server from your preferred vendor, like IBM WebSphere, SUN SunONE, BEA WebLogic, Macromedia JRun etc.

Figure 12.4 shows how Portals and Product Sutes fit into the Enterprise Architecture. In addition, need for common Identity Management and Access Control with Single Sign-On (SSO) is arguably the single strongest force that compels us to bring together all seemingly disparate pieces of infrastructure.

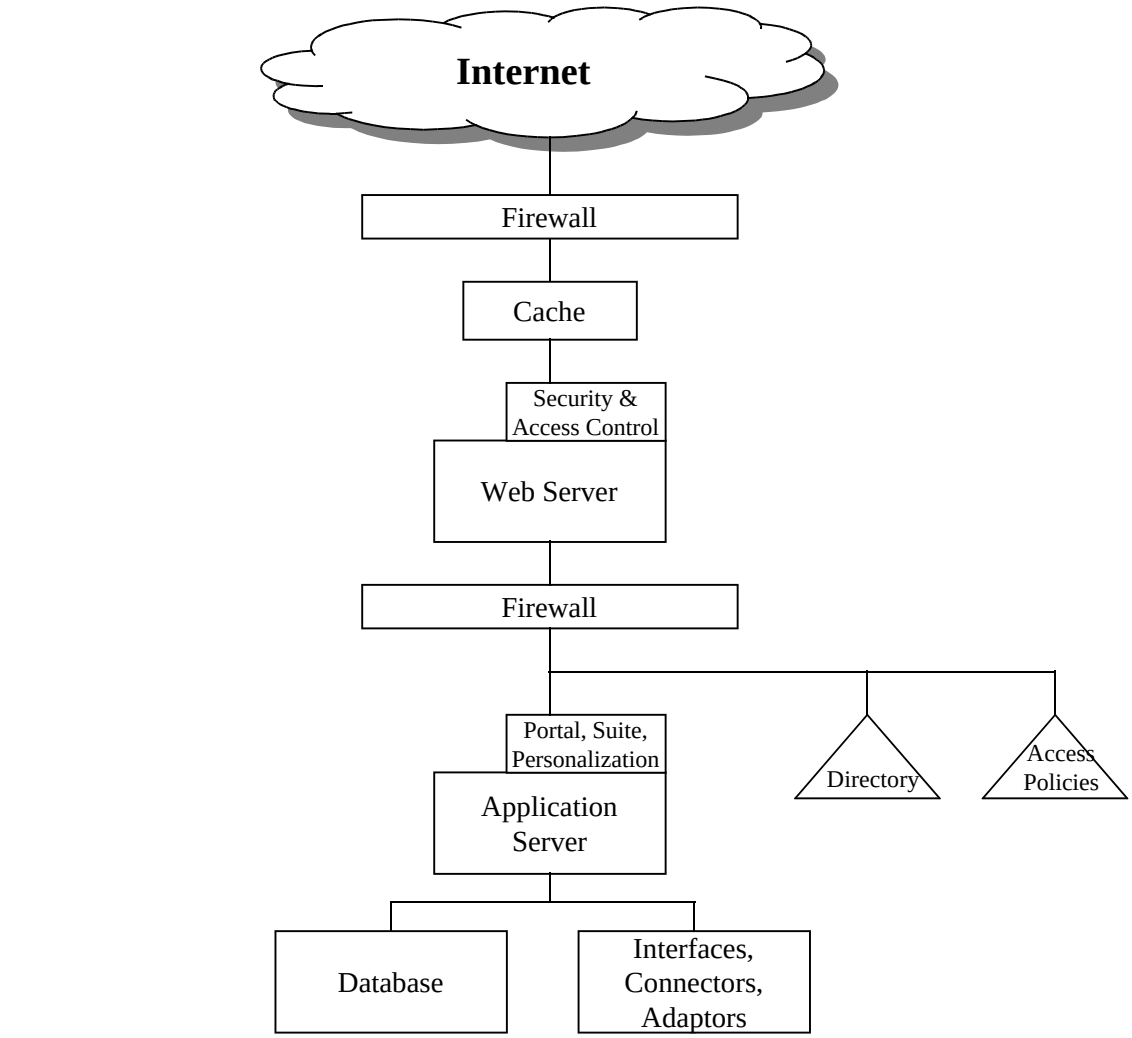

Figure 12.4. Security, Caching, Portals, Suites to the mix

Likely, Enterprise will deploy COTS solutions (rather than in-house developed ones) for Security and Access Control, customised and configured to suit the information model of actors, roles and protected resources in your Enterprise.
Being a complex *technical* problem, good solution for the Identity Management and Access Control in the Enterprise is even more important as the *business* challenge in general, irrespective of the chosen technology, going to the very heart of your Enterprise's Information Model.
You cannot manage your Enterprise if you failed to comprehend and capture in your Profile Directories and Access Policies repositories the taxonomy and semantics of your domain – your information model and conceptual model just will not fit your business. *Impedance mismatch* between various platforms and products in your Enterprise will fade into insignificance in comparison with conceptual incompatabilities between sub-domains that may be introduced.
This caution should help you in assessment of Access Control solutions for your Enterprise. Any such solution will not relieve you from responsibility to define the taxonomy and semantics of your domain that determine how repositories of the Access Control product will be populated, and what processes will be enabled.

Mature and Dependable – 'no limits'

Previous sections of this chapter defined main components and layers in the technical architecture of Internet-facing Enterprise. We described major high-level deployment and integration patterns in the Enterprise Architecture.
This section discusses *Load Balancing*, *Failover* and *High Availability* aspects of Enterprise Architectures primed for scalability and high-volume traffic.

In conventional approach to Requirements Analysis where we identify *Functional* and *Non-Functional Requirements*, these requirements fall into *Non-Functional Requirements* basket. All too often, importance of these requirements in the Enterprise Architecture is insufficiently appreciated.
Ask yourself, is there any use in having clever functionality if clients cannot access services because of poor performance or unreliable availability? There is nothing *non-functional* about it!

Figure 12.5 demonstrates approach to providing redundancy, load balancing and failover on every layer of infrastructure.

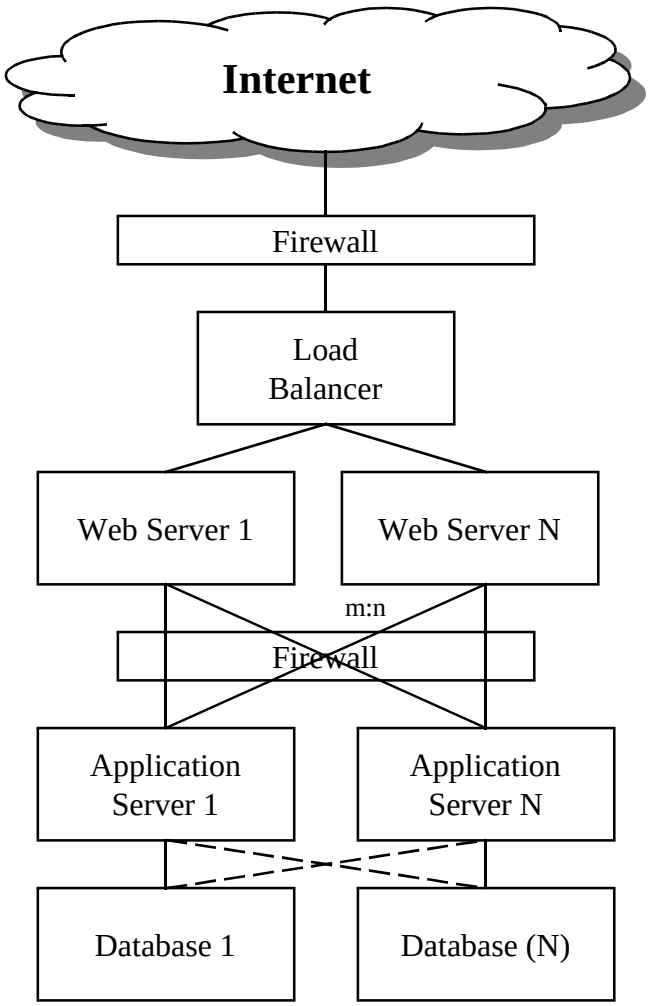

Figure 12.5. Load Balancing, Failover and High Availability

Note the intention of eliminating the Single Point of Failure (SpoF) on every layer. We can increase availability on every layer separately though redundancy and parallelism in replaceable elements. However, layers themselves form a sequential chain of the control and data flow where overall architecture can be only less available than the least available layer. In order to make this point, we discount for a moment an important practical scenario when selected critical services might remain available even in case of partial failures. Enterprise Architect should identify such critical services and make a special effort to ensure they are available when other, less critical services, are down.

For instance, registration of new customers may not require full access to back-end systems. We can attract and capture valuable customers even when service itself is not available, and politely invite new customer to come and visit the web site later, after storing his or her details in the Profile Directory or in the temporary database.
As the matter of fact, this may not be an unpleasant customer experience at all, if customer chooses to step through the demo of our application, which remained available as well.

Load Balancing deals with different protocols on every layer, and will employ different techniques. Whatever the protocol or technique, main challenge in Load Balancing remains the State Management of the connections to redundand components.
State Management of the connection with Load Balancing translates into:

- enforcing the 'sticky' session with the same redundand component, when this component maintains conversation with the client by keeping the persistent session state from one request to another within the session, or
- timely propagating or sharing the state between redundand components when 'stickiness' is not required anymore

Load Balancing of Web Servers may be achieved by DNS round robin routing of HTTP requests to the Web Servers farm.

On the Application Server layer, Load Balancing solution will depend heavily on technology of Application Server platform (with J2EE and .NET Frameworks as usual suspects), and on statefullness of the application code.
Furhermore, Load Balancing solution for the J2EE Application Server will be similar, but different for various vendor products, due to Load Balancing not being part of the J2EE specification and certification. SUN, IBM, BEA, Macromedia and other vendors of J2EE Application Servers were left to their devices as far as Load Balancing concerned.

J2EE Application Server itself has two layers or *containers – Web Container* and *EJB Container*. Despite potentially being part of the same application, these two J2EE layers will be treated separately for Load Balancing (also see Chapter on Java).
Web Container hosts Java Server Pages (JSP) and Servlets. Access to Servlets is performed over HTTP. Usually, J2EE Application Servers provide Web Server plug-in for connection to J2EE Web Container on the separate box, likely behind the firewall. J2EE Web Server plug-in can be configured to 'spray' HTTP requests randomly to the pool of J2EE Application Servers, thus performing workload sharing and Load Balancing. Also, Web Server plug-in can support 'sticky' HTTP connection for the session duration.
EJB Container, as name suggests, hosts EJBs. EJB clients connect to EJB Container over RMI/IIOP, Java-specific and not 'firewall-friendly' protocol (which may be considered an advantage by some security-conscious sites). By RMI/IIOP not being 'firewall-friendly' we mean that RMI/IIOP breaks if firewall performs Network Address Translation (NAT). RMI/IIOP does not like NAT because IIOP Interoperable Object Reference (IOR, sort of CORBA pointer) is carried as a TCP/IP payload and contains TCP/IP address of CORBA Server object. This TCP/IP address will be invisible through NAT. You will have to ensure that on the access path from the client application to EJB no NAT happens, or be prepared to do some cheating on the network, e.g. by doing NAT twice.
Your EJB Load Balancing solution will be proprietory to the vendor. For example, IBM Web Sphere delivers EJB Load Balancing across several instances of EJB Containers in the same administration domain through its Workload Management (WLM).

As mentioned, Load Balancing can be a very volatile technique. Whatever product vendor solution is, it requires strict adherence by Java application developers to programming guidelines in respect of state management.

In nutshell, application designer and programmer presented with choices:

- Do not store state between invocation of Servlet or EJB – stateless code, no problems for Load Balancing
- Store state in memory of object, page, session, or application – align your programming practices with configuration for Load Balancing, be prepared to sacrifice state in memory if server goes down
- Persist state into database between invocations – better data integrity and transaction recovery, simplified Load Balancing, but be prepared to face database sharing and lockout issues, and input-output performance overheads

Load Balancing on the database persistence layer is a perrenial challenge. We should not expect a simple straightforward solution for sharing database source between several applications, real-time seamless replication or propagation of data between several database instances, clustering of database engines, leveraging intelligent storage devices to achive similar effect of shared and distributed databases. Note, Figure does not show separately database engine and the database storage – this calls for more involved discussion of clustering and high availability in the database technology.
One example of the database technology that allows several database engines to share the same database is Oracle Real Application Cluster (RAC). Oracle RAC eliminates SpoF for the database engine, and database itself can be made more reliable by utilising highly available SAN storage devices.

.NET and DCOM managed to simplify some of these challenges by the tighter control over the state, i.e. by making objects in memory stateless. This is not to say the .NET has limited capability for connecting to data sources. .NET implements persistence and integration through ADO.NET, .NET and Windows interfaces and, of course, Web Services.

Agile, Service-Oriented, Real-Time Enterprise

We keep saying that Enterprise Architecture is driven by new demands of the *modern* Enterprise. Let's clarify what are the features of the *modern* Enterprise that make it so special:

- Diversity of the core business in the Enterprise. Even if Enterprise managed to carve a well-defined niche in its chosen market, in order to conduct business end-to-end, it has to elevate up to the task every business unit, component, system
- Need for speed in doing business, responsiveness in dealing with customers, partners and suppliers
- Quality, completeness and timeliness of service
- Growing inter-dependence of all stakeholders in business transactions, both inside and outside the Enterprise
- Geographical distribution of infrastructure, services, suppliers, partners, customers
- Mobility and accessibility of service
- Security, Privacy, Data and Transactional Integrity
- Resilience and High Availability of IT infrastructure supporting core business of the Enterprise. Protection of the business from natural or intentional man-made disasters
- Agility, ability to adapt to ever-changing environment with rapidly evolving technology, with shrinking shelf life for products and services, with changing behaviours and procedures of customers and suppliers, with changing makeup of the target market and of the Enterprise itself

This list of '-ilities' can go on and on – practically any important feature of the Enterprise needs to be enhanced so that Enterprise Architecture could catch up with increased business demands.
None of these characteristics is new. However, increased pressure to deliver these features, ferocity and speed of change, as well as need for all of them in combination without exception, determine new face of the modern Enterprise.

IT has become a critical intrinsic part of the modern Enterprise, and this mutual dependence increases rapidly.
Monolithic architectures and product suites are not able to provide an answer anymore.

In a nutshell, Enterprise Architectures need to be built from highly configurable lego blocks that could be easily snapped-in with one another, and all together deliver rich customisable functionality.

No news here either. We already explored the notion of re-useable Business Components, Services and Facilities. Furthermore, Business Component and Service based on more fundamental notion of Object-Orientation, graduated to the Enterprise – basic concepts of *public interface*, *insulation* and *encapsulation* of the business logic and data in the component still apply.
Service-Oriented Architecture (SOA), another term that often mentioned in the context of Enterprise Architecture, is yet another incarnation of the same familiar concept.

Multiple complex components participating in the transaction, and rather long control path through these components in the Enterprise in fulfilling the transaction, adds more pressure on performance and end-to-end responsiveness.
Term *Real-Time Enterprise* (RTE) used for the Enterprise Architecture to emphasise the criticality of delivery the sought-after Service in *time*, *place* and in the *form* required. RTE does not necessarily mean 'zero-latency', but rather appropriate latency that our technology is able to deliver without it costing us too much, and just *what*, *when*, *how* and *where* is required in order to keep a well-oiled mechanism of the Enterprise working.

<<< … >>>