Table of Contents for Chapter 10

Part 2. Architect’s Toolbox – ‘Bird’s View of the Terrain’

<<< … >>>

Chapter 10. Microsoft .NET

.NET is the next generation strategy and the suite of products from Microsoft aiming to close the gap between traditional application software development and Web development.

.NET – instant *de-facto* standard

Due to the market share and the mind share of its technologies, any offering delivered and backed up by Microsoft is bound to be a *de-facto* industry standard to be reckoned with, and not without the merit.

.NET Framework

.NET has a potential to do to the Internet what Microsoft Windows and Office have done to the desktop. Conveniences of commodisation possess a strong ability of steamrolling the technical and long-term economical arguments. Stakes for the dominance on the information super-highway even higher than on the desktop – imagine if someone owns the toll-booth there and milks every commercial and infotainment transaction that human society will rely on in the near future in all its day-to-day activities!

.NET is a complete platform for the Enterprise and the Web. .NET is easily extensible by adding more languages, services and components.
Scaled-down version of .NET Framework - .NET Compact Framework – provides platform for building and deploying distributed Web applications on smart devices, such as cellular phones and PDAs.

.NET Framework consists of three main parts – Common Language Runtime (CLR), Unified Classes representing .NET Class Frameworks and Libraries, and Web Services and Applications.

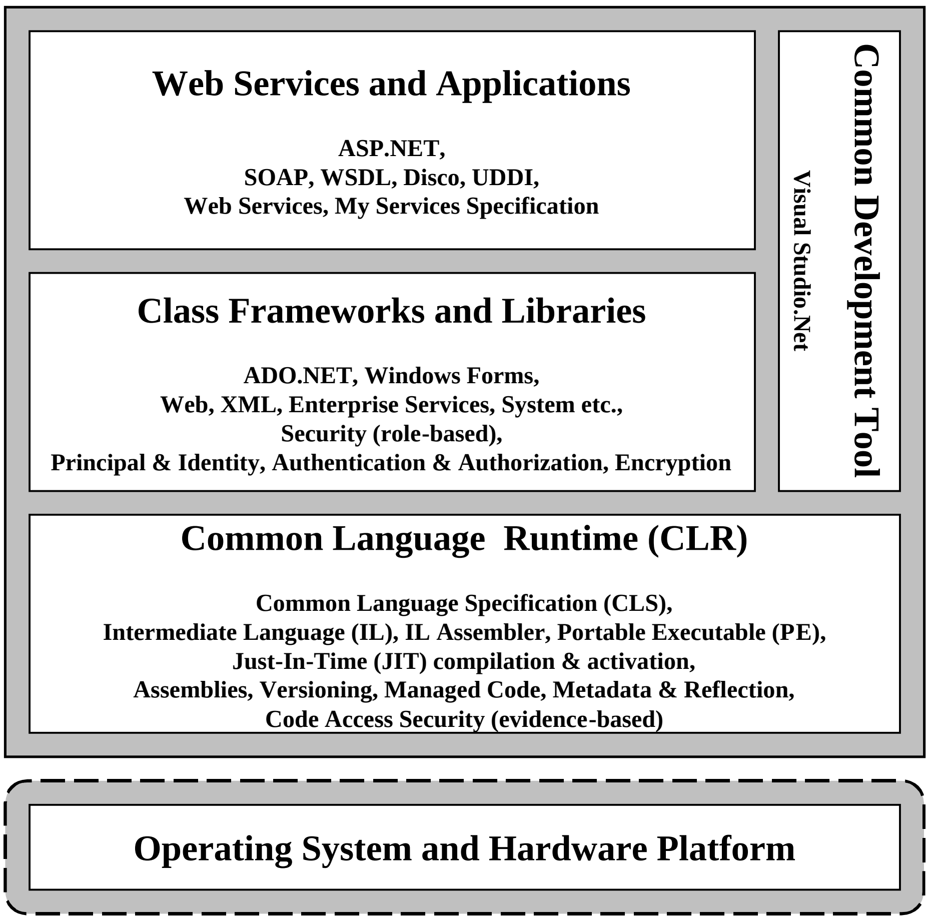
Web Services and Applications
ASP.NET,
SOAP, WSDL, Disco, UDDI,
Web Services, My Services Specification
Class Frameworks and Libraries
ADO.NET, Windows Forms,
Web, XML, Enterprise Services, System etc.,
Security (role-based),
Principal & Identity, Authentication & Authorization, Encryption
Common Development Tool
Visual Studio.Net
Common Language Runtime (CLR)
Common Language Specification (CLS),
Intermediate Language (IL), IL Assembler, Portable Executable (PE),
Just-In-Time (JIT) compilation & activation,
Assemblies, Versioning, Managed Code, Metadata & Reflection,
Code Access Security (evidence-based)
Operating System and Hardware Platform

Figure 10.1. Parts of .NET Framework

Microsoft Visual Studio was a dependable development platform even before .NET.
Visual Studio.Net builds on this strong foundation towards a common software development and debugging environment shared by many languages. Visual Studio.Net also allows third-party vendors to plug in their tools.
Visual Studio development platform has been complemented and reinforced by acquisition of Visio design tool. Visio serves the needs of complex architecture designs that include major notations for UML, database, and network design diagrams.

Common Language Runtime and Execution Model

Common Language Runtime (CLR) provides a lowest level foundation for the .NET platform that executes programs regardless of their original programming language. Any .NET-aware language compiles the source code of the program into common Microsoft Intermediate Language (MSIL), or Common Intermediate language (CIL), or just IL.
Text version of IL is an Assembler-level programming language that is compiled into the Portable Executable (PE) format.

If code complies with the .NET, it is known as “managed”. .NET knows everything about the program module it needs to know to control the execution runtime. .NET does not require to compile the whole application up-front – it can compile modules on the fly, by the process of Just-In-Time (JIT) compilation, or “jitting” (or, if you prefer more scientific terms, you can call this process Judiciously Incremental Translation).
JIT compiles IL code (or, rather PE “executable” version of it) into the native executable machine code that is ready-to-run on the target platform.

Common Language Runtime controls module loading, verification, and memory management including garbage collection, version control, and interfacing with non-.NET code.
At CLR level, .NET provides capability of running over various hardware and software platforms, ranging from high-end servers, to PCs, to PDAs and other handheld devices (on .NET Compact Framework), and to Internet appliances.

.NET object model is the basis for language interoperability; it provides the lowest level denominator that guarantees, if language complies, to work well with modules written in other compliant programming languages.
The .NET object model is a set of Object-Oriented concepts that are documented in Common Language Specification (CLS) – similar to an Object-Oriented language, but without the syntax.
CLS enables such seamless capabilities as inheritance or debugging across different OO programming languages, cross-language exception handling, strong typing, memory management, and code access security.

.NET component model is based on Object-Oriented concepts also. .NET builds “assemblies” – set of classes with well-defined interfaces. Component Model removes distinction between a program module and software component. Due to the rich metadata stored in the “manifest” of the assembly, and absence of Interface Definition Language (IDL), assembly can use other assemblies directly, without the need for any additional wrapping as in CORBA or COM.

.NET Versioning Model defines a standard version numbering policy for the assemblies. This way .NET satisfies the needs of applications for Change Management and Configuration Management, and eliminates incompatibilities known as “DLL hell” that haunted many Windows developers.
.NET versioning of assemblies eases the deployment of applications and components, and allows various versions of applications to run side by side in the same address space without confusion.
Furthermore, CLR is able to validate the baseline versions of the components in the running application and repair it on the fly by loading the correct version of the assembly if required.

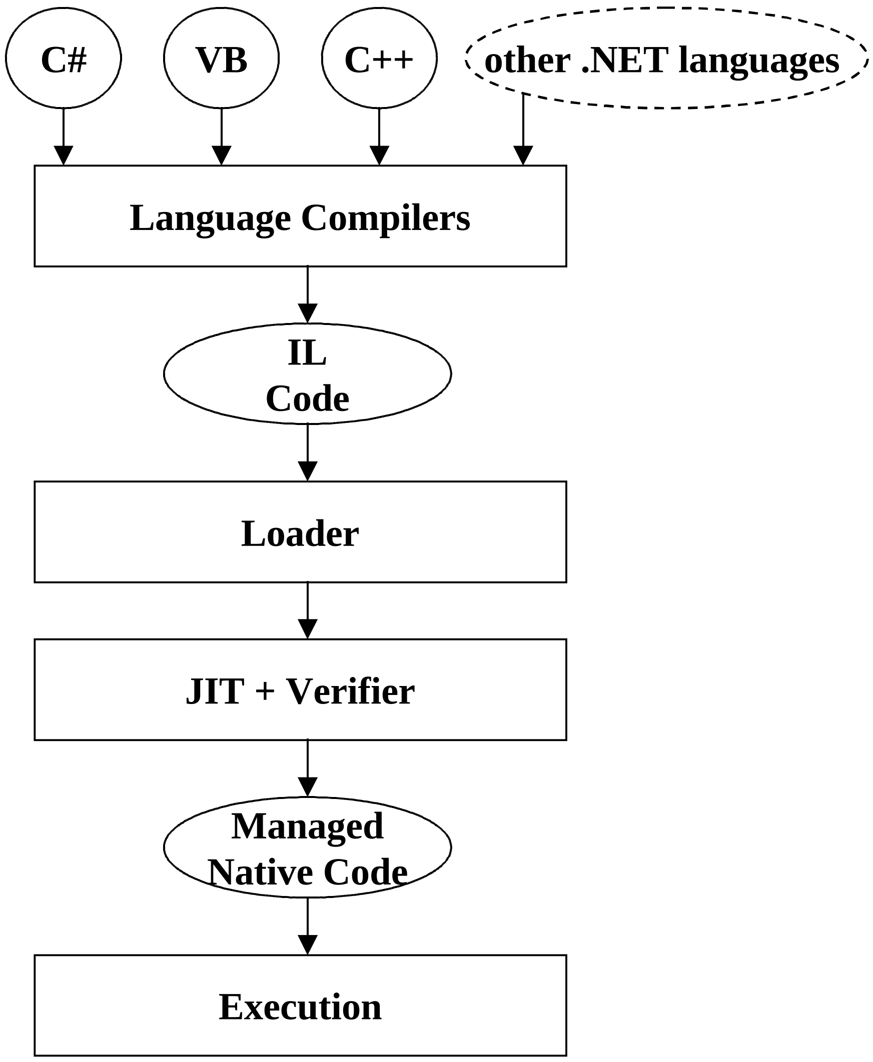

Figure 10.2. .NET Program Execution

As with Java Virtual Machine, if you need to run .NET on your platform other than Windows, you need the version of CLR made available for this platform.
Potentially, CLR may steal the Java thunder known as WORA, or “Write Once Run Anywhere”. Even more so – you can “write once” in any .NET-aware language.

Microsoft joined forces with other industry players and submitted main parts of CLR to ECMA for standardisation. CLR portability may be assured by canonisation of Common Language Specification with common type system, unified classes, IL, and C# to boot (which is good to have, but, strictly speaking, is not required for portability).
With CLR standards, portability efforts will revolve around delivering the JIT compiler that compiles IL into the native code for the target platform. Some brave open source projects are aiming to deliver portable .NET incarnations.

As with any well-designed framework, .NET portability is a viable and exciting option. Industry longs for the true realisation of cross-platform technologies for the distributed business applications. However, portability to the great extent relies on goodwill from the Microsoft as a major force behind the .NET Framework. Real de-coupling of .NET and Windows may take some light away from the Windows, as well as Office (both Front and Back Office), and somewhat undermine flagship Microsoft offerings.

Is Microsoft prepared to open up .NET completely by making the special effort in avoiding the tight coupling with Windows and Office, and facilitating the true portability and zero-effort interoperability of the .NET Framework across non-Windows platforms and other standard-compliant offerings?
This is more than a million dollar question. If history is anything to go by, we should be hopeful, but not overly optimistic.

For instance, some interoperability opportunities have been lost already by exclusion of Java and mis-alignment with the Java class framework in .NET.
Concepts of IL and Portable Executable in .NET could have been aligned with Java p-code and share the same JIT downstream, and the same development and testing environment upstream. We could have seamless single view of the world with the best of Java and .NET universes rolled into one, if two camps were on speaking terms and willing to do more together when ideas were taking shape.
Enterprise Architect, get over it – you do not live in a perfect world.

.NET Security

.NET Security represents a systematic attempt to shed the image of Windows as having a poor security record, and worth a special mention.

Security capabilities in-built into every layer of .NET Framework. Core elements of .NET security architecture include Evidence-based Code Access Security, verification process, Role-based Security with authentication and authorisation procedures, Cryptography, and Application Domains.

CLR enforces evidence-based Code Access Security for the ‘managed’ code during loading the assembly, at JIT-time, and on code execution. One assembly may cause loading of another assembly. Code Access Security is applied all the way on every step in the call chain, thus preventing malicious or untrustworthy code to ‘sneak in’ unverified.

Code Access Security is based on *policies* that match *permissions* to *evidence*.

Permissions are associated with code when managed code assemblies are loaded for execution. Permissions describe resources or runtime objects, and associated access rights.

At runtime, CLR evaluates the assembly’s evidence. Evidence can be gathered from the assembly itself, or from the execution context. Evidence may include namespace, software publisher identity, or code origin such as URL.

Role-based Security gives developers highly configurable and flexible access control based on concepts of Principal and Identity, Authentication and Authorisation.

Authentication is the process of determining the Identity of the user, or verifying with sufficient level of comfort that user is who he or she claims to be.
.NET provides several methods of authentication including Windows Authentication, Form-based (Cookie) Authentication, Basic (encoded) or Digest (somewhat better encrypted) Authentication, Microsoft Passport, or Custom Authentication.

Principal represents the security context of the running code while Identity represents the user associated with that security context. Roles determine the access rights of the Principal.

Authorisation determines Principal’s access rights to the URL, including the HTTP request method (GET, HEAD, POST etc.).

Application Domains provide secure level of isolation between applications running within the same CLR process.

Web Applications

.NET XML-based interchange standards open up .NET components and services to the outside world via SOAP, WSDL, and the suite of Web Services standards at large, embraced by the IT industry in general.

Web developers on .NET platform will benefit from ASP.NET.
ASP.NET has its heritage in Active Server Pages (ASP) scripting language, but more than just an incremental update to the ASP.
ASP.NET leverages power of the .NET Framework to the full extent. With ASP.NET web developers close the gap between Web development and traditional software development.

Microsoft greatly influenced development of XML-based Web interchange standards. Web Services standards, such as SOAP and WSDL, achieved broad industry adoption and enthusiasm, and will be described in detail elsewhere in a book.

As a natural continuation of the effort in building the end-to-end framework for Web applications, Microsoft caps .NET with ‘pre-cooked’ business components that are exposed as a Web Services for all to use.
My Services Specification defines these Web Services.

General idea of provisioning the set of re-usable business components is not new, but universal acceptance of the common middleware glue hindered the efforts.
My Services may remind you of CORBA Services and Facilities that relied on CORBA middleware to succeed. Many bright ideas and high quality object-oriented designs went into CORBA Services. Hopefully, industry is more mature and ready now for broad adoption of yet another incarnation of business components in the guise of My Services.

.NET My Services require user Authentication through another Microsoft service - .NET Passport.
.NET Passport is Microsoft’s implementation of Kerberos distributed security protocol for the .NET My Services.

.NET Passport is a strong proposition for making a commodity out of common User Identity and Single Sign-On, provided you are happy to stay within the boundaries of the Microsoft .NET My Services world. This is a single and most contentious point that introduces proprietary flavour into .NET My Services, as opposed to the standard W3C Web Services.
Enterprise Architect should watch closely interoperability between .NET My Services and non-.NET Web Services. Industry standards only create a foundation for interoperability of Web Services, but their implementation may compromise true interoperability.

.NET My Services are designed as an extensible set of business components. More services will be made available in response to business demands. Some Core .NET My Services include:

- ***.NET Profile*** – Stores user personal information
- ***.NET Inbox*** – Provides user access to their e-mail box from any device
- ***.NET Calendar*** – Stores user’s calendar information in one place. User can choose what part of the calendar to share, and with whom. For instance, a spouse might have a full access, a boss can access work-related appointments only, ticket purchasing agency can access only free un-allocated timeslots
- ***.NET Documents*** – Provides users with secure document storage
- ***.NET Wallet*** – Storage for commercial payment, shipping and billing information, like credit card details or shipping address
- ***And many, many more services***…

<<< **…** >>>