1 What is .NET?

At the beginning of 2002, after several years of research and development, Microsoft brought .NET (pronounced: *dot net*) to market. .NET is not an operating system in the narrow sense of the word and so it is not a successor to Windows. It is more a layer on top of Windows (and perhaps on other operating systems in the future). It mainly adds the following two things:

- ❑ A **run-time environment** that offers automatic garbage collection, security, versioning and, above all, interoperability between programs written in different languages.
- ❑ An **object-oriented class library**, providing a rich set of functionality for graphical user interfaces (*Windows forms*), web interfaces (*web forms*), database connectivity (*ADO.NET*), collections, threads, reflection and much more. In many cases it replaces the current Windows API and goes beyond it.

However, .NET is more than that. It is an open platform that Microsoft developed to merge diverging streams of software development and thus to offer their clients a uniform technology again.

- ❑ Web applications (for example online stores) are currently developed with a technology that differs from that with which desktop applications are written. Desktop applications are programmed with compiled languages such as C++ or Pascal and make use of object-oriented class libraries and frameworks. Web applications, on the other hand, are written in HTML, ASP, CGI and interpreted languages such as JavaScript or PHP. With .NET both styles of applications can now be developed with the same techniques, for example, with compiled languages such as C# or Visual Basic .NET, as well as with a comprehensive object-oriented class library.
- ❑ In recent years, organizations have invested a great deal of money in software developed in various languages such as C++, Visual Basic or Fortran. They don’t feel like throwing these investments away. They are much more interested in seeing programs that have been written in different languages working together smoothly. .NET makes this possible with an unprecedented degree of interoperability.

- ❑ Recently there has been a huge growth in the use of small computers such as hand-helds, palmtops and systems with embedded micro-controllers. Special languages and operating systems have been developed for them too. With .NET they can be programmed with the same languages and libraries as PCs and web servers. This allows the development of software for mobile and embedded systems to move closer to conventional programming.

A field that has had an underrated role so far is that of distributed systems that co-operate via the Internet to perform tasks that cannot be done locally. In the future such systems will have a much greater significance. For this sort of work .NET offers *web services*. Web services are implemented by means of remote procedure calls and work together by using XML and protocols such as HTTP.

.NET also makes a range of tools available, the most prominent of which is Visual Studio .NET. This is a multi-language development environment with a debugger and a GUI designer that is specially developed for use with web forms or web services.

Under the title .NET Microsoft also subsumes various servers such as the *SQL Server*™ [MSSQL] or the *BizTalk*™ *Server* [MSBiz] as well as a range of ready-made services such as the *.NET Passport service* [Pass], that provide frequently needed information and operations via web services.

So what then is .NET? It is a concerted set of system components, libraries, tools, web services and servers that is hard to define in a single sentence. In combination it aims to make the programming of Windows and web applications more straightforward and more uniform. Software development under .NET is radically different from the current style of programming for Windows and the web. It is simpler, safer and more elegant. However, one must be prepared to learn new APIs and new concepts.

1.1 The .NET Framework

The .NET Framework forms the core of the .NET technology (see Fig 1.1). It consists of a run-time environment and an object-oriented class library that covers all areas of Windows and web programming. With this comes the new programming language C# that has been specially designed for .NET. This chapter will briefly describe the various parts of the framework. The remaining chapters of the book look at each topic in more detail.

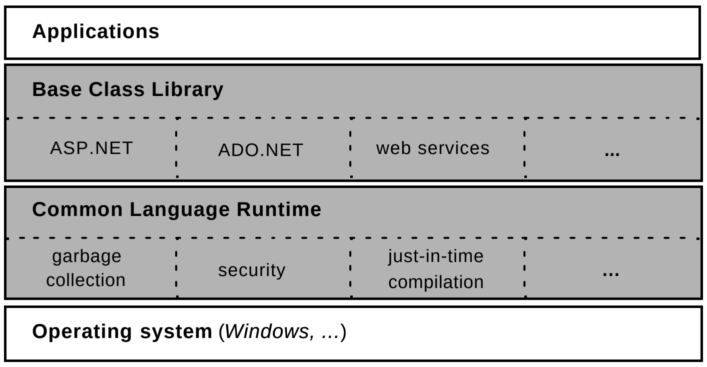

Figure 1.1 *Outline of the .NET Framework architecture*

Common Language Runtime

The Common Language Runtime (CLR) is the run-time environment under which .NET programs are executed. It provides, among other things, garbage collection, security and interoperability.

Similarly to the Java environment, the CLR is based on a *virtual machine*, with its own instruction set (CIL - *Common Intermediate Language*) into which programs written in any .NET language are translated. Just before they are run, CIL programs are compiled (*just in time*) into the code of the target machine. The CIL code guarantees interoperability between different languages as well as code portability. JIT compilation (*just-in-time compilation*), on the other hand, ensures that programs are nonetheless efficient.

However, for different languages to be able to cooperate it is not enough for them just to be translated into CIL. They must also use the same sort of data types. Therefore the CLR defines a *Common Type System* (CTS) that describes how classes, interfaces and primitive types are represented. The CTS not only allows a class, implemented in C#, to be used in a Visual Basic .NET program, but it even allows the C# class to be extended by a subclass written in Visual Basic .NET. Similarly, an exception raised by a C# program can be handled by a program written in any other .NET language.

The *Common Language Specification* (CLS) is the minimal subset of the CTS that all languages must support if they wish to make use of .NET's language interoperability. At the moment there are more than 20 such languages, both from industry and from academia. Alongside the Microsoft languages such as C#, Visual Basic .NET and Managed C++ are Fortran, Eiffel, Java, ML, Oberon, Pascal, Perl, Python and Smalltalk. Managed C++ is a variant of C++ that is translated into *managed code* that runs under the control of the CLR.

The CLR offers mechanisms for making .NET programs safer and more robust. Among these are the *garbage collector*, which is responsible for reclaiming the storage of objects once they are no longer needed. In older languages such as C and C++ the programmer himself was responsible for freeing this space. So it could happen that an object's space was freed when it was still in use by other objects. This left the other objects staring into a void and could lead to unrelated storage areas being destroyed. Similarly a programmer could forget to release the space of an object. This then remained in the memory as a *memory leak*, wasting space. Such errors are hard to find but, thanks to the garbage collector, can never occur under .NET.

When a program is loaded and translated into machine code the CLR uses a *verifier* to check that the type rules of the CTS have not been violated. For example, it is illegal to treat a number as an address and use it to access storage areas that belong to other programs.

The CLR defines a common platform for all .NET programs irrespective of what language they are written in and what machine they run on. A system of well thought-out rules ensures that all programs have the same view of types and methods, and that method calls, exceptions and threads are all handled in the same way. The CLR as a part of the .NET architecture is described in Chapter 3.

Assemblies

.NET supports component-based software development. The components are called *assemblies* and are the smallest units that can be individually deployed. An assembly is a collection of classes and other resources (for example, images). It is stored either as an executable EXE file or as a DLL file (*Dynamic Link Library*). In some cases an assembly is even made up of multiple files.

Each assembly also contains *metadata* in addition to code. The metadata holds the full type information of the classes, fields, methods and other program elements in the assembly. An assembly also contains a *manifest*, which can be thought of as a table of contents. Thus assemblies are self-describing and can be inspected and used by loaders, compilers and other tools, by means of *reflection*.

Assemblies are also used for version control. Each has a multi-level version number that also applies to all the classes within the assembly. When a class is compiled, the version numbers of referenced classes are recorded in its object code. The loader then asks for those classes (i.e. assemblies) that correspond to the expected version numbers.

Under .NET several DLLs with the same name but different version numbers can coexist without getting in the way of each other. This spells the end of the "*DLL hell*" under Windows, where the installation of new software could cause old DLLs to be overwritten by new ones with the same names so that existing software would suddenly cease to work.

Assemblies do not have to be recorded in the Windows registry. They are simply copied into the application directory or into the so-called *global assembly cache* and they are equally easy to remove when they are no longer needed.

Assemblies are effectively the successors of COM components. Unlike under COM (*Component Object Model*) one doesn't have to describe them via an IDL (*Interface Definition Language*), because they contain comprehensive metadata that has been gathered by the compiler from the source code. The common type system guarantees that software written in different languages uses the same sort of metadata and is thus binary compatible. However, investment in COM components is not lost. It is still possible to use COM components from .NET classes and vice versa.

The Language C#

Although .NET programs can be written in many different languages, Microsoft has developed a new "in-house" language that fully exploits the power of the CTS. C# is an object-oriented language that looks like Java at first sight but goes beyond Java in its capabilities.

C# supports single code inheritance and multiple interface inheritance. It facilitates the development of components in the sense of component-oriented programming by providing *properties*, *events* and *delegates*. All types in C# (and in other .NET languages) form a uniform type system that allows primitive types such as numbers and characters to be treated as objects. In addition to *reference types*, such as classes and arrays, there are also *value types* that do not get stored on the heap but on the method stack. This relieves the garbage collector from keeping track of those objects and so makes the programs more efficient.

Number-crunching applications often work with multi-dimensional arrays. Unlike in Java, such arrays can be stored in contiguous memory locations in C#. This increases the efficiency of such applications.

User-defined indexing operators (so-called *indexers*) allow accessing the elements of lists and other collections with the familiar array-index notation. A new loop construct—the *foreach loop*—makes working with such object collections particularly simple and readable.

C# is one of the first languages that can be extended by using so-called *attributes*. Attributes are metadata that can be attached to almost all program elements (classes, fields, methods, parameters etc) and that can be queried at run time by means of *reflection*. This is how conditional compilation, serialization of objects, COM interoperability and other useful mechanisms are implemented in .NET.

Reflection does not only allow access to attributes but also to other metadata. One can, for example, discover at run time what methods a class has. It is even possible to call methods or to access fields whose names were still unknown at compile time. This makes it straightforward to implement debuggers, analysis tools and test environments.

Other notable features of C# are *exception handling*, *operator overloading* and user-defined *conversion operators* between different types.

Many of the characteristics of C# are also found in other languages. So C# is not revolutionary but, rather, a mixture of the best features of other programming languages. It is a practically-oriented selection of modern sofware engineering concepts that every .NET programmer needs. Therefore C# has a chapter of this book dedicated to it.

The Base Class Library

The Base Class Library (BCL) contains the most important classes of .NET and can be used by all .NET languages. It provides functionality for a wide range of purposes. In most cases it supersedes the current Windows APIs that were infamous for their complexity and inconvenience. However, it is still possible to call classical Windows functions from .NET. The BCL is divided into namespaces, each of which deals with a particular functionality. Among the most important namespaces are the following:

- ❑ System.Collections contains classes that manage collections of objects. These include lists, sets, trees, dynamic and associative arrays and hash tables.
- ❑ System.IO contains classes for input and output including general data streams, files, directories as well as formatted input and output of data.
- ❑ System.Threading provides classes for parallel programming. Among these are threads and thread pools as well as synchronization mechanisms such as semaphores and monitors.
- ❑ System.Net is deals with network programming. It includes sockets and network streams, protocols, such as HTTP and the corresponding request and response classes as well as cookies.
- ❑ System.Reflection allows access to metadata and thus to run-time type information. It contains classes such as Assembly, Type, MemberInfo, or MethodInfo that can be used not only to get information about programs but also to manipulate them dynamically. It is even possible to create and execute programs at run time.
- ❑ System.Windows.Forms is concerned with graphical user interfaces. There are classes for windows, dialogs and other user-interface elements. This namespace is one of the richest and most complex parts of the BCL and largely replaces the *Microsoft Foundation Classes*. Visual Studio .NET makes it possible to build graphical user interfaces by drag-and-drop and to install methods that will react to user input.

- ❑ System.XML contains classes for the creation and reading of data in XML (*Extensible Markup Language* [XML]) format. XML plays an important role in web services and other parts of .NET.

This core of the BCL is discussed in Chapter 4. Further important namespaces are System.Web that is used for programming dynamic web pages under ASP.NET, and System.Data that contains classes for accessing databases under ADO.NET. Both of these are described in their own chapters of this book.

ADO.NET

ADO.NET comprises all the classes of the .NET library that are concerned with accessing databases and other data sources (such as XML files). In contrast to its predecessor technology ADO (*ActiveX Data Objects*) ADO.NET is object-oriented and therefore more structured and straightforward to use.

ADO.NET supports the relational data model, with transactions and locking mechanisms. Therefore it is independent of different data providers and database architectures. The differences between concrete data sources like MS SQL Server, OLE DB (*Object Linking and Embedding Database*) and ODBC (*Open Database Connectivity*) is abstracted away with common interfaces.

Database access can be *connection-oriented* or *connectionless*. In the first case, a permanent connection to a data source is established. In the second, a snapshot of a part of the database is fetched into a DataSet object and then processed locally. In both cases the data can be accessed by SQL statements (*Structured Query Language*). ADO.NET is described in Chapter 5 of this book.

ASP.NET

ASP.NET is the part of the .NET technology that deals with programming dynamic web pages. Its name is reminescent of the predecessor technology ASP (*Active Server Pages*). However, the programming model is fundamentally different.

With ASP.NET, web pages are constructed dynamically on the server from current data and are sent to the client in the form of pure HTML, so that any web browser can display them. In contrast to ASP, ASP.NET uses an object-oriented model. Web pages, as well as the GUI elements that appear in them, are objects whose fields and methods can be accessed in programs. All this is done in a compiled language such as C# or Visual Basic .NET and not, as in ASP, in an interpreted language such as JavaScript or VBScript. Thus web pages can take advantage of the entire class library of .NET.

User input is handled in an event-driven way. When a user fills out a text field, clicks a button or selects an item from a list, this raises an event that can be handled by code on the server side. Although the server is stateless—as is usual for the

Internet—state information is retained automatically between individual user interactions, in fact in the HTML code itself. This represents a considerable simplification over the former programming model, where the programmer himself was responsible for maintaining state information.

ASP.NET offers a rich library of GUI elements that go far beyond what is supported by HTML, although all GUI elements are eventually translated to HTML. Programmers can even build their own GUI elements and thus adapt the user interface of their web pages to their particular needs. It is particularly straightforward to display the results of database queries as lists and tables, since ASP.NET has largely automated this. Validators are a further new feature. They allow user input to be checked for validity.

Authentication of user access to protected web pages is supported by various methods, ranging from standard Windows authentication through cookie-based authentication to Microsoft's external passport authentication service.

Visual Studio .NET allows the user interface of a web page to be built interactively, in a way familiar from the development of desktop applications. GUI elements can be dragged into windows with the mouse. Values of properties can be assigned using menus and property windows, and methods can be specified that will be called in response to user input. All this sweeps away the difference between programming desktop applications and web applications and simplifies the development of online stores and pages that show frequently updated information (for example, stock data). ASP.NET is treated in depth in Chapter 6 of this book.

Web Services

Web services are regarded as a core of the .NET technology, although they also exist outside .NET. They work via *remote method calls* using textual protocols such as HTTP and SOAP (an application of XML).

The Internet has proved itself to be tremendously powerful for accessing information and services distributed around the world. Currently, access is mainly through web browsers such as Internet Explorer or Netscape Navigator. Web services, on the other hand, allow a new style of cooperation between *applications* by making them communicate without web browsers. Ordinary desktop applications can fetch information such as current exchange rates or booking data from one or more web services that are running as methods of applications on other computers and which respond over the Internet.

The calls and their parameters are generally coded to conform to SOAP, an XML-based standard that is supported by most large firms. Programmers need to know nothing of this. They call a web service in just the same way as a normal method and .NET takes care of translating the call into SOAP, sending it over the Internet and decoding it on the target machine. On the target machine the chosen method is invoked and its result is transmitted back to the caller transparently, again using SOAP. The caller and the callee can therefore be written in quite different languages and can run under different operating systems.

In order for .NET to be able to carry out the coding and decoding correctly, the web services, together with their parameters, are described in WSDL (*Web Services Description Language*). This is also done automatically by .NET.

Microsoft expects that there will be countless web services worldwide that will offer useful information. To find the right web service, UDDI (*Universal Description, Discovery and Integration*) has been developed. This can be regarded as an address book that helps find the appropriate service for a particular requirement. Thus UDDI takes over the role of a search engine for web services. Web services are described further in Chapter 7.

1.2 What Does .NET Offer?

Compared to current Windows and web programming .NET contains many new features. But what tangible benefits does it offer? How is .NET useful to programmers and users?

Robustness and Safety

Type checking and verification of CIL code make sure that programs cannot perform illegal instructions and so cannot, for example, access the memory of other programs or manipulate pointers. The garbage collector guarantees that there will not be any storage reclamation errors. All this eliminates the majority of the problems that used to bring programmers to the brink of despair.

The versioning of assemblies permits DLLs of the same name to coexist and avoids the complications that can arise when an old DLL is overwritten by a new one. Some call it the end of "*DLL Hell*".

System administrators can not only define access rights for individual persons (*role-based rights*) but also for particular parts of code (*code-based rights*), which are checked regardless of who is running the code. Assemblies can be signed using *public key cryptography* so as to be sure that they are in their original form and have not been altered or extended later. This significantly reduces the problem of viruses.

Simple Installation and De-installation of Software

Software is installed under .NET simply by copying all the program files into a directory. DLLs no longer have to be kept in a global system directory nor recorded in the Windows registry. Assemblies that are used by many programs can be stored in the so-called *Global Assembly Cache* in which there can also be several

DLLs with the same name but different version numbers. De-installation is equally easy. The files are simply deleted from the disk. No registry entry or other remnant is left behind.

Interoperability

Under .NET the various parts of a program do not all have to be written in the same language. For each part the appropriate language can be chosen—for example, Managed C++ might be used for the system level parts, C# or Visual Basic .NET for the user interface, and ML for those parts that are best expressed in a functional language.

Program parts written in different languages can work together seamlessly, thanks to the common language runtime and its common type system. Not only is it permissible to call methods from another language, but it is also possible, for example, to create in Visual Basic .NET an object of a class that has been declared in Eiffel, or to handle in a C# program an exception that has been thrown in a Managed C++ program.

The object-oriented programming model defined by the common type system assists the development of software that is object-oriented and thus modular, extensible and easy to maintain.

Uniform Software for Desktop and Web

With .NET, object-oriented programming is now available for web programming in the same way as it has been in desktop programming for many years. Web pages and their contents are objects with fields and methods that can be used in server page code. Cryptic mixtures of HTML and script code, as were usual with ASP, are a thing of the past. Furthermore, web services make accesses to programs that run on remote computers appear as conventional method calls.

Thus, software development techniques for desktop and web applications, which have diverged in recent years, come closer together again under .NET.

Standards

Although the .NET technology originated at Microsoft, its core consists of several open standards. The ECMA standard 335 [CLI], for example, defines the *Common Language Infrastructure* (CLI), including the CLR and part of the BCL. The ECMA standard 334 [C#Std] defines the language C#. SOAP is based on the W3C standards for HTML and XML. It is itself presented as an IETF standard (*Internet Engineering Task Force*) [SOAP]. WSDL will also become a W3C standard [WSDL]. Finally, UDDI is a de facto standard that is supported by more than 200 firms, including Boeing, Cisco, Fujitsu, Hitachi, HP, IBM, Intel, Microsoft, Oracle, SAP and Sun [UDDI].

1.3 Differences from Java

.NET bears considerable resemblance to Java and so is often compared with it. As with Java, .NET is based on a virtual machine that presents a run-time environment into whose code all programs are translated.

Whereas CIL programs are *always* translated into machine code, programs in Java bytecode are initially interpreted. Only when a Java method has been called a certain number of times does it get translated into machine code by a background task. This has the advantage that Java programs can start up immediately in the interpreted mode. Under .NET the JIT compiler must always run first and so a program experiences some tenths of a second delay when it is called for the first time. On the positive side, however, there is no need for an interpreter under .NET. The designers of the CIL code also had more degrees of freedom because the CIL code was never intended for interpretation.

At first sight, the languages Java and C# are very similar. Indeed there are dialects of Pascal that differ from one another more than C# does from Java. Closer inspection, however, shows C# to be more powerful than Java, even given that many C# features are mere syntactic sugar and can also be accomplished in Java in some way or another. For example, C# has a uniform type system. It offers value types as well as reference types, has reference parameters and many other useful features such as properties, indexers, enumerations, delegates and attributes. On the other hand, Java is simpler and stricter: There is, for example, no notion of code parts being marked as "unsafe", meaning that type rules can be quietly broken there. Java is also stricter regarding exceptions. It insists that the programmer must always handle them, which is not the case in C#.

The class library of Java and .NET are very similar, indeed so much that the names of many classes and methods are the same in both systems.

In place of ASP.NET, Java has a technology called JSP (*Java Server Pages*) that is in turn derived from the old ASP technology. As in .NET, JSP pages are translated into classes (*servlets*) that generate a specific HTML stream for the client. The main difference between JSP and ASP.NET is the fact that under .NET there is a clean separation between the HTML description of a web page and its program code. Under .NET, they can be in different files whereas in Java the HTML code is mixed up with the Java code fragments. In addition, the state management of a page, the object-oriented access to GUI elements and the event-driven style of reacting to user input are better engineered under ASP.NET.

For web services there is also a corresponding Java technology from Sun Microsystems called ONE (*Open Network Environment* [ONE]). As with web services under .NET, it is based on SOAP, WSDL, and UDDI. Web services under

.NET, however, are more closely integrated into the system than in Java. Here Microsoft is slightly ahead.

The main difference between Java and .NET lies in the different objectives of these systems. Whereas Java aims to support a single language on many different operating systems, .NET has the exactly opposite aim, namely, to support many different languages on a single platform. Just because the platform is currently called Windows does not mean that it must always be that way. There are already implementations of .NET for other operating systems (for example, Linux and FreeBSD) and other processors (for example, SPARC and PowerPC) [Mono]. However, the best support of .NET is likely always to be under Windows.

1.4 Further Reading

This book gives an overview and introduction to the whole of the .NET technology. For the individual aspects of .NET (for example, CLR, C#, ASP.NET, or web services) there are specialized books that treat them in considerably more detail than is possible here. Some of these books are listed in the bibliography.

There are also numerous online sources, from introductions, through tutorials, to detailed specifications. New developments are picked up and described more rapidly through the Internet than is possible in the printed form. Some of the more important .NET portals are listed here:

- www.microsoft.com/net/
  This is Microsoft's official .NET site, with general information about .NET, specifications and example programs. The latest version of the .NET Framework SDK can be downloaded from here. It contains the CLR, the base class library, compilers for C# and other languages, but not, however, Visual Studio .NET.
- msdn.microsoft.com/net/
  This is the site of the Microsoft Developer Network, with all kinds of technical information on various aspects of .NET.
- www.gotdotnet.com
  Another rich source of examples, articles and other useful information about .NET.
- www.devhood.com
  A site with many examples, tutorials and training modules on .NET.
- www.iBuySpy.com
  A well-documented example of a web-shop implementation with ASP.NET.
- dotnet.jku.at
  This is the web site for this book, with sample solutions for the exercises, the source code of all the examples in this book, as well as links, tools and various information on .NET.

One of the most useful sources for .NET is the online documentation [SDKDoc] that comes with the .NET Framework SDK (see the CD in this book). It contains introductory texts, tutorials, examples and detailed specifications. In particular, the reference documentation is an indispensable companion for all .NET developers. The documentation is thorough and readable. With this book as a guide and the online reference documentation it should not be difficult to become an expert in all details of the .NET Framework.

1.4 Exercises

1. **Getting started.** Install the .NET software development kit (SDK) from the CD that comes with this book. Look at the documentation that can be opened by Start | Programs | Microsoft .NET Framework SDK | Documentation and browse through the pages named *Getting Started*.
2. **Common language runtime**. What services does the .NET common language runtime provide on top of the services that are offered by Windows? What are the benefits of these services?
3. **.NET versus Java**. Which features of the .NET Framework resemble those of the Java environment? Which features are not supported by Java?
4. **C#**. Which software engineering principles are supported by C#? How does this language help in the development of large software systems?
5. **Assemblies**. Why are .NET assemblies easier to install and to deinstall than COM objects?
6. **ASP.NET**. What are the major differences between ASP.NET and the older ASP technology?
7. **Web services**. Describe several scenarios in which web services could be useful.
8. **Web search**. Visit the web sites www.microsoft.com/net, msdn.microsoft.com/net, www.gotdotnet.com and www.devhood.com and look at the information they offer about .NET. The web site www.go-mono.com describes the port of .NET to the Linux operating system.