08/10/01

From COM to .NET

By Juval Lowy

.NET is a revolutionary new component technology from Microsoft, available with the next release of Visual Studio, called *Visual Studio.NET*. .NET offers exciting new application frameworks such as WinForms, ADO.NET, ASP.NET, and web services. As Microsoft's next generation component technology, .NET was designed from the ground up to simplify component development and deployment, while at the same time provide programming languages interoperability at an unprecedented scale.

However, often companies have considerable investments in existing product lines, and adopting a new technology such as .NET comes with a hefty price. This article discusses a few tips and strategies on how to prepare for the move to .NET, so that when the time comes to shift, the move will be with minimum cost and as smooth as possible. I assume that your company is already heavily involved with Microsoft technologies, and that you use COM as the substrate of your component development and connectivity.

Develop a Migration Plan

Not all your existing code and applications need to be affected by .NET. In fact, it is quite possible to develop a migration plan that will not only preserve your existing investments, but also take advantage of the new capabilities available with .NET. I consider the careful design of the migration path as the corner stone of a successful adoption of .NET.

First, you need to make the distinction between porting code and interoperating with existing code.

Porting

Porting is the act of moving code between different platforms. Porting is in general a delicate, time-consuming task, and contrary to common knowledge, is rarely done.
Often the changes required to the existing code to run properly on the new platform, go beyond mere syntax or API changes. The difficult aspect of porting is changes to the programming model. For example, in the case of porting a COM component, a C++ class or VB control to .NET, you will have to change the programming model from deterministic finalization (Release() in COM, destructor and Terminator in C++/VB) to garbage collection, and actively manage resources. A classic example is explicitly closing database connections, because you can no longer rely of the object going out of scope to automatically release the resource. If you do not do that, your application will not behave the way it was designed to, and will not meet its scalability and throughput requirements. The changes in the programming model will translate to design and architecture changes,

and you will end up with a re-write, instead of porting. A re-write is often more challenging that starting afresh with a clean slate.

Interoperability

Interoperability is the ability of code on two different platforms to work together. Because porting is so difficult and impractical, interoperability is what you should be looking at when trying to combine existing COM applications and .NET. In .NET, every COM component is painlessly viewed by .NET as a .NET component, and every .NET component is viewed by COM as a COM component. Microsoft went out of its way to assure this level of interoperability, including support for complex scenarios such as converting .NET events to COM events.

Classifying Components

To develop the migration plan, you must first classify your existing components. Some components may not ever be aware that you are using .NET. These are typically low-level components, accessed by other COM components, third party components you use, entry points to other applications, and so on. In Figure 1, these are the type A components.

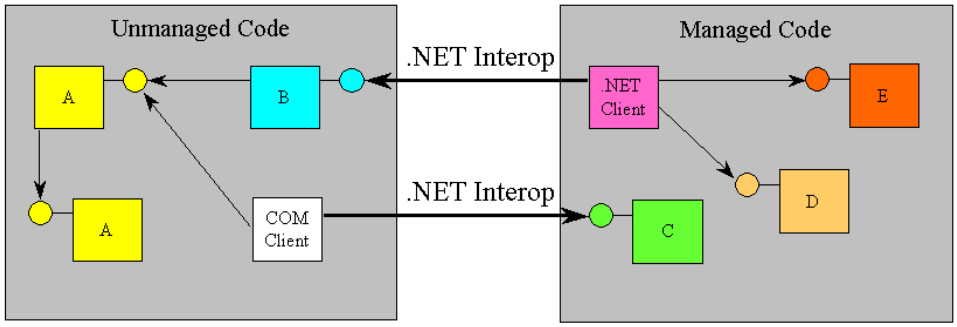

Figure 1: Component types in your migration plan

The other kind of existing components are COM components you will access using .NET interoperation (type B in Figure 1). See the rest of the article regarding design considerations of these components. Moving into the managed part of your application, identify three types of components. There are components that will require providing services to COM objects (type C), usually in the form of subscribing to events but maybe also to provide services. There is some design consideration with those components as well, outlined later on in this article. You will of course have components that you develop natively in .NET (type D). These components will usually be the main draw of .NET for you, and there are likely to take advantage of .NET new capabilities, such as web services, ASP.NET, windows forms and so on. Finally, you may still want to port existing components to .NET (type E). You may want to port you core business logic or trade secret, to ensure that the heart of your application is in on the new platform. Again, be mindful that porting is not a task for the faint of heart, and it will require considerable understanding of both the way the current components work, and of .NET new programming model.

Once you have classified your components, you execute the migration plan like any other project. The important benefit of the migration plan is that it allows you to break down the effort involved in moving to .NET, estimate the effort (and the benefits) involved, and minimize the risk in adopting .NET.

Design for .NET on the COM side

COM components accessed by .NET clients (type B) have to be imported to .NET. The import process creates .NET metadata representing the COM interfaces, in the form of wrapper classes and interfaces. However, the import process cannot convert all the possible COM custom interface signatures. Certain features (like arrays, or server-allocated memory) have no adequate conversion. To maintain high fidelity between the original COM interface definitions and the .NET wrapper classes, make sure that your type B components all use OLE-Automation compliant interfaces. Also, do not use S_FALSE as a method returned value. Any HRESULT indicating error should throw an exception on the .NET side, but S_FALSE is ambiguous.

Design for COM on the .NET side

.NET components that providing services to COM clients (type C) have to be exported to COM as COM objects and interfaces. The export process must comply with COM limitations (such as mo method overloading or static methods). As a result, avoid in your type C components the following:

- Method overloading
- Public methods that are not part of interfaces
- Public static methods and fields
- Parameterized contractors
- Multiple inheritance of interfaces in interface definition

Again, it's not that you cannot do all of the above; it's just that the resulting exported COM definitions will look mangled and messy.

Use COM+

With all its innovations and advanced concepts, .NET is only a *component technology* - it provides you with the means to rapidly build binary components. .NET does not have component services. .NET relies on COM+ to provide it with component services such as instance management, transactions, activity based synchronization, granular role-based security, disconnected asynchronous queued components, and loosely-coupled events. A .NET component that uses COM+ is called a serviced component. .NET refers to COM+ as Enterprise Services.

Using COM+ allows developers to focus on the business logic instead of component connectivity and plumbing. Developers that use COM+ enjoy a productivity boost, faster time to market and a robust application. In addition, using COM+ implies certain architectural constrains and standards, and .NET serviced components have to comply with those requirements, much the same way as traditional COM components. As a

result, the likelihood of a good match between your existing COM+ components and .NET serviced component is high. Remember: A COM+ based architecture means components focus on business logic, and rely on COM+ for all the complicated plumbing and component services. If you choose to implement your business logic in .NET using COM+, it should plug in seamlessly into the existing COM+ architecture.

Use COM+ 1.5 Web Services Support

Web services support is the most exciting new feature of the .NET platform. Web services allow a middle tier component in one web site to invoke methods on another middle tier component at another web site, with the same ease as if the two components are on the same site and machine. But .NET web services come with a price – companies have to invest in learning new languages such as C#, and cope with a new programming model and new class libraries. To preserve existing investment in COM components and development expertise, while providing a migration path to the .NET world, COM+ 1.5 (the next release of COM+, available with Windows XP) can expose any COM+ component that complies with web services design guidelines as a web service. All you need to do is specify the virtual directory of the web service, and COM+ provides the necessary adaptors as a component service. Each component in a COM+ application is exposed as a separate web service, identified by the component prog-ID under the virtual directory. COM+ installs the web services with IIS, and generates the proper web service configuration and information files. The only requirement is that IIS and .NET be installed on your machine to enable web services activation mode for your application.