.NET: Vision, Technology, Agenda

Juval Lowy

.NET isn't just a new technology. It's Microsoft's big vision ... and a tactic to keep the company together.

About the Author

Juval Lowy is the corporate software architect at KLA-Tencor, a Fortune 500 company based in San Jose, Calif. Juval designs and develops frameworks and components to be used across the corporation. He's the author of an upcoming book on COM+ and .NET (O'Reilly & Associates). Reach him by e-mail at idesign@componentware.net.

Recently, Microsoft unveiled its next-generation technology and vision—the .NET platform. The two share the same name (for now), which has created a lot of confusion. To understand the .NET technology, you must examine the problem the .NET vision addresses.

The problem: Most IT systems today are architecturally the same, and internally are more or less homogeneous. The problem is, these systems have a hard time interoperating. You can't use DCOM/RMI/CORBA to invoke methods on components on other systems because of the firewalls, transferring and interpreting the data is inefficient (no common format), and even XML is only helpful to a certain degree—you still must write code to parse, store, and execute. Transactions can't span and flow across systems, you can't fire and publish events, versioning is a nightmare, security and credentials are system-specific, and so on.

This is holding back the Internet. Cycle time for new services is measured in dozens of man-years with enormous risk—by the time you're done, you might be obsolete … and out of money.

This should all sound familiar by now; component technology in the pre-COM days faced similar challenges. With its binary compatibility, location transparency, interface definition, security, MTS, and now COM+, COM has evolved to solve the problems that have prevented components from interoperating easily.

The vision: The .NET vision is elegant and simple: Think of one of today's systems as just one component in a megasystem—the Internet. By applying the same abstract concepts from the components world to the Internet, you get the benefits of components, but on a grand scale. In .NET, HTML is analogous to TCP/IP, SOAP is analogous to RPC, systems are analogous to components, binary compatibility is analogous to systems interoperability, and your system is the Internet.

.NET will allow you to build not just systems, but *applications* that use multiple systems to implement your goals. You will treat whole systems as reusable components in your application.

The technology: To achieve the vision, Microsoft is developing .NET, the technology. This platform allows you to develop binary components that take advantage of the Web. The platform contains architecture solutions (such as SOAP, to call methods on components over HTTP), services (such as COM and Windows interoperability), and infrastructure (such as BizTalk).

.NET doesn't use COM as the supporting component technology to attain binary compatibility between components. The compatibility is achieved by having all .NET components share a managed runtime environment.

The platform allows you to develop binary components with ease—every class is a component, so simply declare a class and you have a binary component. You can also easily use a .NET component from a C++/VB COM client, or a COM object from a .NET client. However, .NET doesn't define new component services, and will take full advantage of COM+ services (transaction, JITA, events, queued components, and so on).

The agenda: .NET's Achilles' heel is critical mass, or adoption. Companies won't want to invest in developing two connectivity models, one hand-crafted and one .NET. It'd be like exposing your COM interfaces in a DLL as exported functions, because your client might not know what COM is.

Microsoft is probably the only company capable of driving adoption, thus bringing a second information technology revolution upon us—the .NET platform will be installed on every PC on the planet. Certainly breaking up Microsoft will at the least hinder the vision and the implied prosperity. This might be the *real* reason behind Microsoft's decision to expose .NET when it's still under development.

In any case, .NET is a fascinating new way for us to develop components rapidly, without many of the hurdles facing COM developers today. VCDJ