Microsoft® TechEd 2001

Chris Frizelle reports from Microsoft's annual European extravaganza

TechEd is now perhaps the largest and most important Microsoft event at which the company seeks to update customers with its vision, goals and progress. The US-based TechEd is normally held slightly before the European one, but the content of the two events is very similar. I spent a week in sunny Barcelona, Spain, with over 7,500 delegates and about 175 journalists, soaking up all the latest developments in Microsoft-land (as well as plenty of delicious Mediterranean seafood: oh, isn't life hard...!).

Software development issues formed a very significant part of the content this year: perhaps even more so than usual. The first keynote was given by none other than Anders Hejlsberg: the original developer of Turbo Pascal and Delphi, before he moved from Borland to Microsoft. Anders is an exceptionally talented software developer and his skills are very much in evidence at Microsoft.

As one would expect, the emphasis of TechEd was very much on .NET and associated technologies. Folk like to present slides that aim to demonstrate that 'the whole history of technology has been leading up to this moment', with cute illustrations. A key phrase from TechEd this year is perhaps 'the programmable web', meaning that there is intelligence at both the client and server ends of the wire. A rather surprising quoted example was Napster: folk all over the world making their music available through some client-side software, with the help of Napster's server.

Strangely enough, this reminded me of my trip out to Heathrow. The minicab driver had got himself an always-on broadband cable connection to the internet and was using a file-sharing program, similar in concept to Napster, to download gigabytes and gigabytes of pirated software, music, videos and whatever, from many thousands of contributors around the world. The key aspect of this technology is that the stuff is not located on one web server, but on thousands and thousands of PCs around the world: therefore making all this material available cannot realistically be policed (without shutting down the net).

As you, dear reader, are a software developer (perhaps with your own software products to sell), you will realise that there is going to be a rather increased interest in truly effective software security systems. Cue Dave Jewell's review of Titanium 2.0 on page 57.

Web Services

Anyway, I digress a little. What Microsoft wants folk to be doing with systems where there is intelligence at both server and client is using Web Services: the ability to loosely couple applications using SOAP (which is based, of course, on XML) to transfer information between server and client. The advantage of SOAP is that it is text-based and thus travels easily through firewalls (unlike DCOM and CORBA).

The protocols that Web Services use are internet communications (eg HTTP), XML for the data format, XSD for the data type system, SOAP for service interactions, WSDL for service descriptions, DISCO for service discovery and UDDI for a service directory.

The loose coupling arises from XML being language-neutral and platform-neutral: the Web Services server (which is providing some information, or content) and the Web Services client (which is *consuming* the service) do not need to understand each others' internal implementations, nor do they need to be able to do anything more compatible than generate and understand XML.

There is the possibility of new business models: companies can provide precisely the content they are specialists in.

One example already in place is OAG, which provides information on flights: who flies where, when, flight status and a whole lot more. Check oag.com for more details. Being a forward-thinking company, it has been looking at providing its information in new ways. Part of this has involved the development of a Web Service to provide flight status information and an Office XP Smart Tag to consume it. The idea is that if you type a flight number into Word, for example, the Smart Tag will recognise it

About Perspectives

The 'occasional' Perspectives column aims to give readers a wider perspective than just Delphi and Kylix, allowing us to keep an ear open to other development technologies, with input from a variety of contributors. Do you have an idea for something we should cover? Email me at chrisf@itecuk.com.
Chris Frizelle, Editor

as a flight number and offer you the ability to connect to the OAG Web Service and retrieve flight status information, which it pastes directly into your Word document. See www.oag.com/ smarttag for more information.

You can imagine the same technology being used for currency conversions, sporting results and a whole lot more. Specific information which at the moment we access using a web browser but could be delivered to just about any application.

And Web Services are easy to implement. Delphi 6 supports them: check Bob Swart's *Under Construction* column on page 18 of this very issue. David Intersimone has an article on the Borland community website at

```
http://community.borland.com/
   article/0,1410,27501,00.html
```

Tips from early adopters include the usual advice to plan well, but crucially not to overdo it: a Web Service probably works best when it does just one thing well and fast.

Of course, if you (or your clients) want to make money at this game, then security is important. Secure sockets (SSL) can be used to encrypt the transfer of information between the Web Service server and client. And do bear in mind what might happen if your service becomes really popular: stress-test the software.

Visual Studio .NET

And, of course, Microsoft would like you to believe that the best tool to use to create Web Services with is the forthcoming Visual Studio .NET. Certainly, as Anders pointed out, the .NET framework has XML at its very core, rather than bolted on as an afterthought.

Beta 2 of Visual Studio .NET was given out free to the over 7,500 delegates at TechEd Europe (and a similar number at TechEd US), so Microsoft is making a strong effort to test this product very thoroughly. If you have used Beta 1, beware: there are some significant changes. One interesting non-technical change is that it is now possible to distribute applications created using Beta 2 (you need to request a distribution licence from Microsoft, via their website).

The .NET framework makes development with Microsoft tools both easier and more robust. Easier, because Microsoft has seen the light and dispensed with COM (do I hear enthusiastic cheers all around the world?), introduced real components (not those horrible VBX things), plus type safety, garbage collection and exception handling. It will be much harder for developers to screw up. Much of this we have already in Delphi, of course, but I must admit the garbage collection looks tasty.

As you will probably know, Borland is working away on .NET compatibility for Delphi, although it is unclear whether this will mean a new version of the Delphi IDE that will use the .NET framework and produce Intermediate Language (IL) files, or a plug-in for Visual Studio .NET, or even both. We must wait and see.

What is very clear is that we in the Delphi world have a huge head start over developers who have been weaned on Microsoft tools: the concepts in .NET are ones we are already very familiar with. The stuff that others will 'ooh' and 'aah' over, we've been doing for years. So let's capitalise on our advantage, embrace change, and jump in. Oh, and we could all make some useful dosh out of it all, too (not that you are mercenary enough to be interested in such things!).

The C# Language

Anders is, of course, the chief designer of C#: the language which almost all of the .NET framework is written in, and which Microsoft wants to be the language of choice in the future. (It has to be said, though, that the company appears to have realised that many Visual Basic developers would have to be physically dragged, kicking, screaming and biting, away from their beloved development tool, so they are putting more effort into making the transition from VB 6.0 to VB.NET rather less painful than indicated by Beta 1.)

I went to an interesting C# overview and spent most of the time thinking 'Ah, I see, yes, just like Delphi...' I kid you not, the thing is basically an amalgam of Delphi ideas and Java syntax, tweaked a bit to avoid charges of plagiarism, of course. For experienced Delphi developers, getting used to the syntax differences will be no hardship. But why would we want to?

Components are very much at the centre of .NET and C#, along with properties, methods and events. And will developers who have been using Visual C++ or Visual Basic be fully up to speed with developing real components for .NET? No, of course they won't. Who are the most likely candidates for this important task, then? Right first time: Delphi developers. Folks, there is a world of opportunity out there.

Sure, when the .NET flavour of Delphi is ready it's quite likely that we'll be able to create .NET components in our favourite language as easy as falling off a log, but while we are waiting we could find that there's a nice little niche for us to slip into while those VC++ guys are still sat scratching their heads. Folk like Developer Express, best known for their innovative and high quality Delphi components, have already shown the way by creating some .NET components.

There are some nice ideas I'd like to see in Delphi, though. One is embeddable XML comments. Just type /// in the code editor and Visual Studio .NET embeds an XML outline for commenting (say, for a procedure or function). Fill in the gaps and then, at compile-time, the system extracts the comments (checking them, and adding more too), combines them with an XSL stylesheet and bingo: nicely formatted code documentation. It's not a new concept, but the way it's built into the system with some extra swings and bells is very good, in my opinion.

ASP.NET

I'm sure that those of you who develop web applications will have spent some time at least demonstrating to clients why applications built in Delphi are so much better than those written in ASP server-side script (usually VBScript). Yes, we've all visited websites which have become over-dependent on those horrid ASP pages which take forever and a day to do their thing.

But the day may come when you and I are indeed developing websites using ASP out of choice, not with our arms twisted behind our backs. But we'll be using Delphi as the language and seeing excellent performance. How is this?

Well, ASP.NET is very different to the ASP we know (and hate): it uses any of the .NET languages (so that will hopefully include Delphi at some point) and compiles the code, so you get good performance (according to *PC Magazine*, 3 to 4 times as fast as old ASP).

Another advantage is the separation of the code from the content: so your code is not jumbled up with the designer's pretty HTML, but kept in clean isolation. And XML is of course right at the heart, along with better integration with web forms, easier deployment, an event-driven architecture and more.

And you can build Web Services with ASP.NET too: they get their own file extension of .ASMX (normal ASP.NET pages are .ASPX to avoid confusion with the old ASP). A nice touch is that you can update ASP.NET applications even while the existing copy is running: just upload the new files and the app will seamlessly recognise the new version and switch to it when appropriate. No more restarting the web server just to allow you to update your web apps, as is needed right now with ISAPI. This will make a huge difference.

ASP.NET is actually hosted in an ISAPI application on the web server (ASPNET_ISAPI) and also provides goodies like state saving, authentication, error recovery and more.

Microsoft Solutions Framework

One session I went to was quite revealing. To be honest, I went for the speaker more than the subject: Rafal Lukawiecki, of UK company Project Boticelli Ltd, is excellent at getting complex stuff across and a real treat to listen to. He was describing how he and Microsoft have used the Microsoft Solutions Framework (MSF) to ensure largish development projects don't fail. Dry stuff, you might think, but actually fascinating.

He had the usual horror statistics (from the Standish Group): only 26% of development projects succeed, 28% fail, 46% are challenged (had problems but did not totally fail), the average cost overrun is 189%, 94% of projects have to be re-started, the average time overrun is 222%, the average functionality delivered is 61%... Aaargh, enough! We get the message.

MSF was used by Microsoft on Windows 2000 and is currently being used with Windows XP. Being a framework it is flexible: it's not a rigid methodology. The minimum size of project is said to be 3 months with a team of 3 people.

One of the most useful aspects at Microsoft has been the use of a daily build cycle, especially in the testing phase. They do a fresh, complete product build at 6pm every day, incorporating all the changes and fixes made that day. Overnight, the new build is installed and stress-tested, to come up with an amended list of stuff to fix for the key 9:30am team meeting, where bugs are assigned to developers for fixing.

You might say that this will produce delays, because some things take more than a day to fix. The response to this is that if something is taking more than a day it's serious and therefore merits a proper team-wide review: it could reflect some problem that is more deep-seated.

Other aspects which seem to work very well relate to the team. Specifically, recognising the various roles: development, program management, testing, logistics, product management and user education. If a daily build cycle is used from as early in the project as possible, then there is something to show the client, and get feedback on, from a very early stage. So the likelihood of the client waiting for months and months, getting the beta and saying 'that's not what I wanted!' is a lot less. And the interesting point is that physical development is just one sixth of the team (though it may be more, or less, numerically of course).

Food for thought. Check www.microsoft.com/msf for some more information.

Conclusions

Time for me to wrap up. Although there was less 'new' stuff this year, I certainly saw ideas from previous years which had become real products, which in turn were being used in real life. Things are moving on. For us in the Delphi camp, I believe that we need to recognise the huge head start we have over old-style developers used to the old-style Microsoft tools: we already know so much about the ideas that Anders and others have put into the heart of .NET, so let's run with our advantage and not get left behind.

And, if you're listening in Scotts Valley, some news on 'Delphi for .NET' (or whatever it might be called) would be very timely, please! What we need from Borland is '.NET far better than Microsoft can do it'. Delphi blew Visual Basic out of the water when it came out: we need Borland to give us the same advantage with .NET development. A tall order, but there are still plenty of very talented people in the Delphi team. Although not previously a Borland approach, perhaps a public beta might on this occasion be a worthwhile plan. It would show the world that there is an alternative, and it would convince Delphi developers to stick with Delphi and not abandon it for C#, which could be a very attractive proposition for many, in all honesty.

Chris Frizelle (chrisf@itecuk.com) is Editor of *The Delphi Magazine*.