

Exclusive Interview with Anders Hejlsberg: Getting Reacquainted with the Father of C#

In our premier issue, back in October 2002, we ran a full-length interview with Anders Hejlsberg, the Distinguished Engineer at Microsoft responsible for the creation of the C# programming language. Here, more than two years later, we present a follow-up interview conducted by *.NETDJ*'s editor-in-chief, Derek Ferguson, at Microsoft's Tech Ed 2004 conference in San Diego, California.

.NETDJ: Between now and the last time we spoke, Borland has entered the .NET space. As an ex-Borland employee who is now one of the most revered .NET icons, what are your thoughts on this?

AH: I'm very excited that they are doing that! It is the right thing for the Delphi community and .NET is the logical next place for all Windows development tools to go. I think it is a win-win situation.

.NETDJ: I didn't realize the last time we spoke that you started off at Microsoft building the Windows Foundation Classes for J++. I never had a chance to look at WFC during its day. If I looked at it today, would I see bits I'd recognize from the .NET Framework?



AH: Some of the ideas from WFC were carried forward into Windows Forms, so you would see stuff there.

.NETDJ: Where, if anywhere, do you see a need for "a language specializing in... <blank>" today? For example, I think there is a real need for a new entry-level programming language. What do you think?

AH: I think there is a healthy cross-pollination that occurs between all the programming languages. While you might see C# pioneer in one direction and VB pioneer another, ultimately there is a lot of crossing over of ideas over time. Certainly, anything we do on the C# team we let the VB folks participate in and vice versa.

.NETDJ: I recently spoke with Soma Somasegar, Microsoft's Corporate VP of the Developer Division, and he mentioned that you were one of the people looking at how XML usage can be done better from a programmatic standpoint. Can you tell us a little about this work?

AH: Well, I wasn't involved in X#, but two of the people who were on the X# team are now members of the C# team. Some of X#'s ideas are being carried forward in C#. These are not going to look the same way as they did in X#. However, I think that the core concepts will carry forward. I can't make

any commitments, though.

Actually, the main thrust of X# was trying to deal with relational data. It also tried to deal with XML, but it did not deal with loosely typed XML, which is what most of the XML in the world is.

My feeling is that if XML is strongly typed, then it can very conveniently be represented as classes. So what we really need is syntax for creating "object literals" - objects with nested objects - all in a single expression. We are kind of calling these "object initializers."

I would not expect to see XML literals embedded in the C# programming language, however. There are two ways to think about integrating concepts. On one hand, you can just embed one language in another. Honestly, however, this approach just calls attention immediately to all of the differences between the two languages. Take Embedded SQL, for example. You still have to learn two languages, but you also have to learn how those two languages interact.

On the other hand, if you could just take the conceptual things that are in two languages and merge them into one, you could get a much more productive environment. This is the approach to data I want to take in future versions of C#.

.NETDJ: The last time we spoke (at OOPSLA 2002), you had just announced that anonymous methods would be added to C#, but you couldn't give a timeline. Now we know that that will occur in Visual Studio 2005. What has happened between now and then to take this idea from a vision to reality?

AH: The stuff we talked about at OOPSLA was the first glimpse we gave of the new C# language features that will be in Visual Studio 2005. There are many that are well-known: generics, anonymous methods, iterators, and partial classes. There are also many more - nullable types, which we just talked about today - and a whole bunch of smaller things that are not as well-known.

Nullable types basically represent the ability to have value types that can be set to null, which is something that people very much ask for. I recently asked a group of programmers, "How many of you access data in your applications?" Virtually everyone raised his hand.

People do a lot of database access, and yet there are some pretty notable impediments to interactions between databases and programming languages. One of these mismatches is the absence of nullable types from programming languages. SQL databases have always had nullable types, but programming languages have never had nullable types. One of the things that you are sort of seeing here is the beginning of us trying to think really deeply about this problem. Nullable types are a specific problem that we are solving as a part of .NET 2.0. In .NET 1.0, value types can't be null, but in 2.0 they can.

[\(continued on page 2\)](#)

System Nullable

.NETDJ: How have you actually implemented this?

AH: Specifically, we have a type implemented called "System.Nullable<T>." It combines a type with a flag that indicates whether or not a variable's current value is null. It is itself a Value type. This saves you from having to box and allocate all of your - for example - integers on the Heap. Nullable types allow you to actually store your values and their null-or-not-null flags together as a nice little package.

Above this - in the Base Class Library (BCL) - we have added a bunch of things in the language itself. Just as we have support for strings in the language, now we have supporting features in 2.0 that allow you to do interesting things with nullable types. For example, we have syntax for nullable integers. We have a bunch of conversions from the null literal to nullable types. And, of course, there are conversions between nullable and nonnullable types.

This is all covered in the specification for C# 2.0, which we recently updated on the C# homepage. We put up the complete and (hopefully) final specification for 2.0, whereas we only previously documented the top features.

.NETDJ: The last time we spoke, you had just returned from a sabbatical. I'm guessing that the march to 2.0 hasn't allowed for any of these since then. Is there another long break in your future? If so, what will you do?

AH: No, no - I'm about to go on vacation for a while, but no sabbaticals now.

.NETDJ: What is a typical day-in-the-life of a distinguished engineer at Microsoft? Do you actually have deadlines and reviews and all that sort of stuff?

AH: It depends on which day it is. I get to work at home on Tuesdays and Thursdays. On Mondays, Wednesdays, and Fridays, however, my typical day is lots of meetings. The C# design meetings have run continuously in the same room on these days from 1 p.m. to 3 p.m. for 5 years now. In these meetings, we design current and future releases of C#. Some people have moved out of these meetings and some people have come in, but I've been in all of these ever since the beginning.

I also meet with all of the other groups. I have a function, for example, as an architect for the BCL, so I might meet with them to keep them abreast of everything that is going on with C#.

On Tuesdays and Thursdays - when I work at home - this is my time to read and write code and specifications and do the more research-oriented aspects of my job.

.NETDJ: I know that the J2EE folks are adding parallel functionality for some of the new features that will be in .NET 2.0, such as generics, as well as some features that are already in .NET 1.1, like attributes. I know nothing about how they compare and contrast, though. Could you enlighten us as to how the .NET approach is, to your way of thinking, better?

AH: Well, so... the Java 1.5 release will have generics and - I'm not sure if the design is finalized - it appears to be the case that their generics implementation is based on the premise that the VM cannot

be modified; code must run on an unmodified VM. The compilers will erase the "genericity" at runtime and substitute the root object types of your object parameters. Unfortunately, this limits how far you can go. You don't get "genericity" over value types, so you can't have a list of integers, for example - only lists of reference types.

.NETDJ: Why is this important?

AH: You wouldn't get any efficiency from using generics with primitive types. When you take things out of a list of a certain type, for example, the compiler needs to insert type casts. Nonetheless, they still incur the overhead that they always did at runtime, so you don't get any execution efficiencies from using generics in J2EE.

The other thing is more subtle, but more important. When you erase type information at compile time, you don't have faithful reflection at runtime. So, you wind up with fewer features available. The industry is relying more and more on dynamic code generation, so it is more important that we have faithful type representations at runtime. In .NET, you can go to any object at runtime and ask it what its type is. For example, with generics in .NET 2.0, I can actually go ahead and understand that I am holding a list of integers. In Java's "erased world," I only know that I have a list - I have no idea what is in the list.

It also looks like Java 1.5 will have a bunch of features that we have always had: a foreach-kind-of-thing (they can't add new keywords) - Enums, extensible metadata (what we call attributes), and a few more. There is some cross-pollination going on and that's great! Certainly, there was cross-pollination going on in the opposite direction when we started .NET.

.NETDJ: Where does the ECMA standardization process stand for C# - current features and new?

AH: At this point, we have submitted all 2.0 specifications to ECMA. The last one we submitted was nullable types about a month or two ago. There are a few more ECMA meetings before this fall when ECMA is set to vote on C# 2.0. As we did with C# 1.0, we have submitted every specification that covers the language in complete detail. Of course, we only have one vote so I can't say with certainty when this will become a standard, but it is moving forward nicely.

[\(continued on page 3\)](#)

Features added to C# in the post-2.0 timeframe

.NETDJ: Are there any other features out there that might be added to C# in the post-2.0 timeframe? Anything you can tell us?

AH: Well, I mentioned a little bit about the space we want to explore more, which is the tighter integration between programming languages and databases. We are now including nullable types, which will allow you to talk to your databases a little bit more effectively. There are many other things I'd like to look at, like the mismatch between relational types and language objects, and the lack of

query and set objects in C#. You can query in the database - but once the data comes down, you can't further subquery without going back to the database.

The other thing inhibiting proper integration between data and languages is the general lack of type checking in the present generation of database APIs. The compiler won't know anything about whether or not the data in your database will work with your code until runtime. This is certainly a direction that we are keenly interested in exploring and we already have some good ideas.

.NETDJ: When last we spoke, you said that generics and other new language features for C# might not make it into the Compact Framework. Correct me if I'm wrong, but they actually all made it in, didn't they?

AH: I do believe that they have, but you might want to check. All of the other features are just language features, so they are - by definition - in the code.

.NETDJ: What originally drew you to Pascal? Are there any implementations of Pascal for .NET of which you are aware?

AH: I wanted to write an ALGOL compiler because I liked ALGOL, but my partner said there was a new thing called Pascal, so we should go and check it out. It is a simpler language both to teach and to implement, so we went with that!

As far as Pascal for .NET, Delphi is backwards-compatible all the way back to Turbo Pascal, and it can create .NET code quite nicely! Delphi is what Turbo Pascal became. In fact, the 16-bit Delphi was the actual Turbo Pascal code base!

.NETDJ: How did you originally get involved in computer technology?

AH: I started originally doing computers in High School, and that's where I learned ALGOL as my first programming language. When I started at the technical university in Denmark in 1979, they had this event for all the freshmen where we'd go away to a kind of camp with the seniors. So, there I was playing cards with a senior and he lost a bunch of money, so I had to get to know him. It turned out that he was interested in starting a company to sell kit computers.

We wound up founding a company to sell Z-80-based computer kits and I wrote a bunch of different software for this thing. One of the things I wrote after writing assemblers and games and such was what essentially amounted to Turbo Pascal. It was a little 12k Pascal compiler that was burnt into ROM. Ironically, we were recommending that folks yank out the Microsoft ROM for BASIC that came with the kit and put in our Pascal ROM instead!

.NETDJ: Did that company become Borland, then?

AH: No. My Pascal implementation grew and grew and eventually became a full implementation, and got moved to CP/M. We hooked up with the guys who founded Borland, because they were also based in Denmark (though they incorporated in the United States) and they were writing their software using

a different implementation of Pascal. We told them about our software and why it was better and they said "Hey, why don't we do a joint venture where we will do the marketing and you do the development?"

This turned out to be a very good thing! I remember that, at that time, software development tools were generally about \$500 a copy. So when they said "why don't we sell it for \$50," we thought they were nuts!

It was, however, a great success. Pascal wound up being the only thing that I did at Borland, which soon moved their headquarters to the United States. It quickly became tedious to commute across the Atlantic, so I moved to the U.S. in 1987.

.NETDJ: Last time we spoke, I asked you for your favorite programming language and you said you didn't have one. So, this time I will ask you - of all the programming languages you've seen -- does any appear to you to just be a complete abomination? (My personal choice would be anything of the TCL/LISP family, for example).

AH: It's hard to say. I don't know. Batch files and the programming that is going on in those definitely need some help! Another language that is difficult to learn, but very powerful, is XSLT.

I see lots of little languages go by in various projects and I tend to stress internally at Microsoft that we all need to get on the .NET Framework so that we all share the same power of the API, regardless of our language choices.

One of the trends I think is interesting is the integration that is happening with programming models like ASP and XAML, which are mixtures of declarative and programming code that give you an amalgam of two different programming disciplines.

What I'm shooting for with the next generation of Microsoft platforms is to use XML for declarative tasks and C# for programming tasks. I'm less of a believer in using XML to "new up" objects in C#.

[\(continued on page 4\)](#)

Who are your technology idols?

.NETDJ: Who are your technology idols?

AH: I look at it like, "If I've seen so far, it's because I've been on the shoulders of giants." There have been so many people who have done so many things. Coming from the world of C, I thank Bjarne Stroustrup for C++, James Gosling for Java, and Kernighan and Ritchie for C. I'm just happy to be able to carve out a little corner of that universe.

.NETDJ: How do you perceive the role of C# in the .NET universe? Is it the preeminent language of .NET, the C-family language for .NET, or something else?

AH: I think it is the C-family language for .NET. I think that if you are a C/C++/Java programmer today you will definitely come into .NET feeling very comfortable with C#. If you are programming in VB6, VB.NET will feel more familiar. Either way you go, both are completely integrated. You can now have projects that use both languages. With Visual Studio 2005, you can even link together VB and C# code into a single assembly, so external consumers can't even tell!

.NETDJ: So, what is the function of language in .NET, then?

AH: I think the thing that is really important to understand is that, in the world of .NET, language is really just a question of syntax. The unique value proposition for .NET is that all languages ride on a common runtime and therefore the learning curve...well, 10 or 20 years ago learning the syntax of a language was 50% of the learning curve, and 50% was its APIs; indeed it might have even been that the language was more. Now, the API is the overwhelming learning curve, 95% or better. So, having a common API regardless of your choice of language is an enormous benefit!

Under Windows, the current API is the .NET Framework. Going forward, it will be WinFX, which is even better! Syntax is a lifestyle choice. VB or C#, we all sort of tend to visualize things in the language in which we are most comfortable. Is English better than French? I don't think so. I don't think there will ever be one end-of-the-line programming language. New languages are how we make progress. New languages arrive and fix problems from old languages, and so on and so forth.

.NETDJ: I was recently asked by someone why .NET needs both Events and Delegates. Honestly, I couldn't answer them. Could you explain this?

AH: This is like asking why we need classes and fields. They are different things. Delegates are types, and Events are members, so we have to have both Delegates and Events. The tie here is that an Event is of Delegate type, but encapsulates a particular instance of that Delegate.

The way to think of it is that you can think of Events as being properties of a Delegate type, so you kind of see that distinction. Now we do add some syntactical elements in C# so that, rather than getting and setting Events directly, we ask you to use the -= and += operations. Other than this, Events are like Properties of Delegate type.

.NETDJ: What is the one feature unique to C# that you think developers coming from other platforms most tend to overlook?

AH: I think it is perhaps the type system unification, which is subtle but important. It is the notion that everything in C# is an object and you can start with this premise. I was talking to a professor who is teaching C# and he was claiming that he thinks C# is easier to teach than Java because it has a unified type system. He can get into the idea of value types versus reference types - which is pretty difficult material - later in a C# course than in a Java course.

In Java, you have to introduce primitives, references and wrappers, and so-on-and-so-forth immediately, because Java doesn't have a unified type system. I personally think that this makes a lot

of sense.

.NETDJ: Since last we spoke, Global Outsourcing has become a much hotter topic in development circles. What are your thoughts on this?

AH: I can't say that I have any well-formulated thoughts. It seems more of a political issue than a technical issue.

.NETDJ: Last time we spoke, Mono was in its infancy. Since then, they have actually issued their first full release. Do you have any thoughts on this platform?

AH: I've talked to Miguel and looked at Mono a little bit. I think it is nice to see that our standardization efforts are bearing fruit and that the CLI is being implemented on other platforms. I think it is important to a lot of customers to know that alternate implementations are available, even if they don't use them. It will keep us on our toes, too! Competition is healthy.

.NETDJ: Before we go, could you tell me a little bit about your book?

AH: It is *The C# Programming Language* - the complete technical specification of the C# programming language. It is based on all of the ISO and ECMA specifications. The book, unlike the standards, has a gentler introduction and also has a whole section on C# 2.0 that describes the new features.

.NETDJ: I can certainly see the usefulness in a clear explanation of a new feature like Generics. Don't you think the whole concept is too complex to ever become generally used by .NET application developers?

AH: Oh no, quite the opposite. I expect that Generics and the new `List<LT>` and `Dictionary<K,V>` generic types will become the bread-and-butter of any .NET application! Honestly, they do precisely what `ArrayList` and `HashTable` do, except that they give you strongly typed collections with all the convenience methods a developer is likely to need.

Previously, there has sort of been this tension between strong typing and convenience when writing code with lists and arrays. You could have an array of `Customer` type, for example, but an array can't grow itself or sort, etc. So then people use `ArrayLists`, which have all the convenience methods, but don't retain any type information. With generics, you can have your cake and eat it too! You can have a strongly typed `ArrayList` - a list of type `Customer`, for example.

I believe that this will become an incredibly used type. It will get used much more often than `ArrayLists`, for example. The best part is that you won't need to understand Generics at all in order to use Generics-based collections. In order to author them, probably yes - but using them is just like using an array!