Anders Hejlsberg fue entrevistado en junio, en el reciente congreso de Microsoft Tech-Ed en Orlando acerca del pasado, presente y futuro del lenguaje C#.

Anders Hejlsberg es uno de los principales ingenieros distinguidos de Microsoft Corporation. Él es conocido por haber desarrollado el compilador de Borland Turbo Pascal, y ha sido el arquitecto principal de la tecnología Delphi de Borland. Hejlsberg dejó Borland, para unirse a Microsoft en el año 1996. Desde principio fue criticado por estar creando el lenguaje de programación C#. Originalmente con el nombre de desarrollo Cool, C# fue diseñado para acabar con Java.

En el keynote del Tech-Ed dado por Paul Flessner (MVP), supimos como Visual Studio, SQL Server y BizTalk Server quedan cada vez más estrechamente integrados. ¿Cuáles son las implicaciones de esa integración progresivamente estrecha entre herramientas y los lenguajes?

Pues las herramientas, es el copiloto para que sienta que consigue más y más habilidades. Mi interés particular durante el par de años que pasó ha sido realmente pensar profundamente en la gran incompatibilidad que tenemos entre los lenguajes de programación, C# en particular, y el área de la Base de Datos, como SQL, o respecto a eso, el mundo XML, como XQuery y esos lenguajes que existen allí.

En mi sesión del Tech Ed, pregunté, "¿cuanto de ustedes acceden a una base de datos en sus aplicaciones?". Se rieron y me miraron, y entonces todos levantan sus manos.

Así de que sustraje que cuando usted aprende a programar en C#, de hecho no sólo aprende a programar en C#. También aprende SQL. Y también aprende todos los APIs (las interfaces de programación de la aplicación) que están relacionados. Y aprende un estilo completo de escribir una aplicación distribuida…. Los dos mundos están de hecho sorprendentemente integrados

En el lado de herramientas, hemos progresado tremendamente consiguiendo una integración más profunda entre estos dos mundos. Pero reflexiono sobre el lado de lenguaje, también podríamos hacer una enorme cantidad de progreso.

Entonces, hay tantas cosas que podríamos hacer allí para que estos mundos estén más integrados. Y eso es lo que hemos estado pensando profundamente en este par de años que pasaron. Y pienso que usted ve algunos de los frutos de esta integración en la liberación de "Whidbey". Específicamente, una característica como generics (que estará en parte de Whidbey, o Visual Studio 2005) no es sólo una herramienta grande para programadores porque le da permiso de parametrizar sus tipos y tener mucho más código compartido, y obtiene mejor seguridad de tipo en tiempo de compilación. También fortalece el sistema de tipos, o lo hace más expresivo.

En términos de otros lenguajes, he notado que (el señor Indigo) Don Box y (el señor Avalon) Chris Anderson han estado escribiendo bastante últimamente acerca de Ruby y Python. ¿Dónde ve usted ha Microsoft jugando en ese ámbito?

Hacemos muchísimo trabajo para hacer de .NET una plataforma aun mejor para los lenguajes dinámicos.
Pienso que hay en cierto modo, un interesante resurgimiento de interés por lenguajes dinámicos. Cuando miro, clasifico de vista dos cosas diferentes allí. Y algunas veces pienso que las personas confunden algunas de las ventajas de los lenguajes dinámicos. Por ejemplo, algunas personas dicen que amo escribir en Python, mi lenguaje dinámico favorito, porque mi código es bien terser

y eso es mucho más fácil escribir, y de esa forma se vuelve mucho más rápido. Y algunas veces las personas dicen, pienso que está bien pagar el precio de tipos no comprobados para obtener ese código terser. ¿Entonces yo digo, es eso realmente necesario para tener tal strong typing en nombre de tersity? No pienso que sea, necesariamente.

Actualmente pienso que un mundo aun mejor es un mundo donde un lenguaje es terso, pero aún es strongly typed. Y pienso que es absolutamente posible hacer eso. Algo de eso, en su infancia, está en algunas de las capacidades de inferencias de tipos que tenemos en C# 2.0, donde, eficazmente, inferimos el tipo de la variable de la forma usted que lo usa. Y hay mucho más que podemos hacer allí.

¿Usted llegaría inclusive a crear otro nuevo lenguaje para poder obtener esto?

No pienso que haya alguna necesidad de crear otro lenguaje a estas alturas. Una vez que usted crea otro lenguaje, En el afán de solucionar un problema, usted adquiere otros nueve problemas. Pienso que muy naturalmente podemos expandir los lenguajes que tenemos de manera que lo puede ejecutar.

¿En cuanto a Visual Studio, ve usted todos los lenguajes que son soportados ahora evolucionando en la paridad? ¿Continuará usted haciendo para Visual Basic todo lo que usted hace para C#?

Pienso que es muy improbable que nosotros evolucionemos conjuntamente. Las innovaciones ocurrirán en un espacio y ellas resultarán ser grandes para usuarios sobre aquí. Pero absolutamente siempre compartiremos toda la información que podemos y trataremos de tener paridad como podemos. Pero no pienso que sea o significativo o posible garantizar completa paridad.

Yo pienso que hay algunas diferencias en los perfiles de programadores VB (Basic Visual) y los tipos de características en los que ellos están interesados, y los programadores C#, quienes deben de estar algo más interesado en el bleeding edge de tecnologías del lenguaje.

Fui informado — y puedo estar equivocado — que la siguiente versión de C # (3.0) tomará prestado una parte de FoxPro.

No, no diría que específicamente tomar prestado de FoxPro. Como he dicho antes, el área donde ya colocamos algún trabajo base en Whidbey (Visual Studio 2005) es esta área grande y gran parte inexplorada de una integración más profunda del lenguaje y de datos. Y, por supuesto, FoxPro es un lenguaje que ha estado allí, pero también, como dBase y todos esos lenguajes, llamados lenguajes del 4GL - aunque realmente no sé lo que significaron esos "4" - pienso que muestran como la cercanía a los datos es útil para una cierta clase de aplicaciones.

Pero desafortunadamente, también pienso que esos lenguajes, por cualquier razón, nunca consiguieron ser aceptado como los lenguajes de programación representativos de la mayoría. Carecieron de algunas capacidades que los programadores, hablando generalmente, querían.

Lo que trato de hacer es intentar hacer esta cosa de los datos sin arrojar afuera al bebé con el agua de bañera. Y soy muy consciente de eso. Lo que fuera que hagamos para C# tiene que sentirse completamente natural para los programadores C#.

FoxPro llega a esto desde un ángulo diferente. Tienen su propia infraestructura de tiempo de ejecución y ellos no son hospedados en .NET en la misma forma como los otros lenguajes lo son. De tal manera que esa es la pelota que tienen que perseguir. C# está ya en lo que pensamos,

somos la infraestructura directamente de tiempo de ejecución pero está desprovisto de las capacidades de integración más profunda de datos. De tal manera que es donde estamos mirando.

Los equipos están actualmente uno al lado del otro. Conozco a todas las personas del equipo FoxPro. Así es que tenemos mucha conversación. Y tienen mucha experiencia en este ámbito. Pienso de nuestro acercamiento en C#, sin embargo, va a estar un tanto diferente. FoxPro está muy apretadamente unido a sus datos estando los datos en una base de datos. Cuando hablo de datos en esta conversación, significa datos en grande. Necesariamente no tiene que venir de una base de datos. Puede venir de un documento XML o precisamente puede ser algo que usted movió de un conjunto de objetos y ahora usted quiere hacer un arreglo y analizar operaciones en estas gráficas de objetos. Así en esta percepción, somos en cierto modo diferentes.

Soy curioso de saber cómo trabaja usted. ¿Trabaja usted en la misma forma como si usted fue un autor escribiendo una novela? ¿Perfila usted la "novela" primero, así es como usted sabe a dónde quiere ir? ¿O escribe usted capítulo por el capítulo?

Pienso que usted escoge un objetivo grande. Dejemos simplemente decir por el bien de la discusión que quiero hacer fácil el programar datos en C# tal como lo es en FoxPro.... O escoja cualquier objetivo a usted quiera. No estoy diciendo que es una meta particular. Y entonces usted comienza a aprender. Hace su primera aproximación del problema. Y entonces usted aprende bastante de él.

Usted comienza a construir lo qué usted piensa que es la solución. Y entonces de la solución que usted adquiere cómo hay realmente mas de una solución general para el problema ..... la solución que usted construyó es una instancia específica de la solución más general. Entonces, efectivamente, usted se dirige a un metanivel. Y eso típicamente son las extensiones del lenguaje. Sería completamente adverso, por ejemplo, para una extensión del lenguaje tan atado a nosotros — (uno) dicho eso, vamos a hacer integración de datos y eso va a ser SQL con SQL Server y eso es lo que vamos a atiborrar en el centro. Bien, ese sería el beso de muerte para C#. Porque entonces, ya no es un lenguaje de programación de propósito general.

Así es que ese es el nivel de generalidad que usted tiene que buscar para garantizar su instancia de una solución que usted construye, si no es así, entonces usted lo puede botar y puede construir otro y otro. De lo contrario, lo que acaba usted de hacer envejece el lenguaje. Cuando han acopiado tanta corteza, parecen viejos y cansados.

¿He estado interesado en algunos de los varios proyectos de investigación de Microsoft alrededor de los varios "Sharp" como X#, F#, espec#. Las cosas que Microsoft aprende en esos proyectos van a cambian lo que usted está haciendo? ¿Cómo?

Oh, claro que sí. Tomemos los generics. Pienso que ese es un ejemplo maravilloso. Generics se puso en marcha como un proyecto de investigación hecho por Don Syme y Andrew Kennedy en la investigación de Microsoft en Cambridge(UK). Estos dos investigadores eficazmente tomaron el lenguaje C# y la plataforma .NET y añadieron parametrización de tipos encima de él. Lo modificaron. Escribieron dos documentos de identificación acerca de eso y ganaron experiencia con ella.

Su trabajo fue muy creíble y muy aplicable, desde que estuviesen trabajando con los mismos bits. Entonces, todo ese trabajo se compenetró gradualmente en el código de producción (Visual Studio 2005). Terminaron por echar una mano y trabajar en las implementaciones de producción de estas cosas. Creo que el prototipo (de su trabajo) fue llamado Gyro cuando fue creado. Esto

estaba construido en la base del código de Rotor. Fue, para mí, un ejemplo maravilloso de transferencia de tecnología de investigación en un producto. Pienso que Don y Andrew son investigadores geniales e ingenieros muy pragmáticos. Esa es una combinación grande. Deseo que hubiera más de eso.

Don está ahora trabajando en esta cosa llamado F#, lo cual toma una apariencia más profunda en los lenguajes de programación funcionales en la plataforma .NET. Y ciertamente muchas de las cosas que estamos haciendo con C#, seguimos adelante, aprendiendo de — y ya aprender de — lenguajes de programación funcionales.

Es algo interesante. De muchas formas, veo mi trabajo como forma segura de mantenimiento en lo que ocurre en la investigación y academia y la construcción de puentes para las tecnologías del mundo real. ¡Porque cuando usted mira lenguajes de programación funcionales, como Haskell o ML o lo que sea, es análogo! Para muchos investigadores, la sintaxis es simplemente una encarnación necesaria de semántica. Realmente, en lo que están interesados es en la semántica. Y quién cuida qué notación. Bien, desafortunadamente entre programadores, la sintaxis tiene importancia. ¿Usted puede ver eso en las guerras religiosas ya sea si debería ser VB (Basic Visual) o C# - cuál, eficazmente, se reduce qué sintaxis usted prefiere?

Así en ese sentido, soy un ingeniero en el corazón. Pero estoy aficionadamente interesado en investigación. Entonces, si, F# y Spec# son cosas que miramos y evaluamos todo el tiempo. Nos encontramos con los investigadores. Nuestra reunión de diseño C#, lo cual es una serie de reuniones que han sido candentes por seis años en el mismo cuarto, en las mismas ranuras de tiempo, hacemos que los investigadores vengan regularmente y que den presentaciones en lo que ellos están haciendo y viceversa. Esas son todas las tecnologías que miramos. C-Omega (antiguamente conocida como ' Xen ' y ' X# ') es otro. Algunos de los miembros de ese proyecto de investigación son ahora miembros del equipo de diseño C#.

¿Cuáles son los conceptos generales y las tendencias del lenguaje de programación que le interesan ahora, en el esquema más magnífico?

Generalmente hablando, es interesante pensar en más estilos declarativos de programación vs. los estilos imperativos. Muchas cosas de las que hablé son instancias actualmente específicas de eso. Los lenguajes de programación funcionales y las consultas son actualmente un estilo más declarativo de programación. Usted en la clase declara lo que quisiera haber hecho, pero no exactamente cómo quiera hacerlo. Y de muchas formas, nosotros hemos educado las generaciones de programadores para pensar no sólo acerca de “lo que”, sino acerca de “lo como”, y para explícitamente declarar “lo como” en sus programas.

De muchas formas, los programadores tienen que gradualmente desaprender eso y aprender a confiar en que cuando declaran "lo que", la máquina es lo suficientemente lista para hacer " lo cómo" la forma que la quieren hacer, o la forma más eficiente.

---

by: http://www.neuronaltraining.net/blog