

The Inside Track From Here to There October 2000

In the last edition of *The Inside Track*, I introduced Microsoft's .NET, Great Plains Platform Services and myself. This time, I'd like to talk about where the technology infrastructure at Great Plains has been, where it's going and how we'll all get there.

I don't have to tell you that working with technology is a tricky business. It's a moving target. On the other hand, technology doesn't just appear from nowhere. People have been thinking about and working on the underpinnings of the technologies we use today for quite a while. For me, understanding the roots of technology helps to calm the panic I feel when faced with a mountain of new stuff to learn. Saying that doesn't make the mountain smaller, the devil is in the details as they say. Still it gives me hope that I can understand what the heck is going on. Given that a little background can go a long way, let's look back at the technology in Great Plains' products.

Way back in the beginning there was Great Plains Accounting (GPA). The first 3 versions of GPA ran under the UCSD (University of California, San Diego) P-System. The P-System was its own virtual operating system (OS). This meant that a program written for the P-System would run on any computer that supported the P-System. Hey, portable code! While the P-System had its advantages, it was a whole OS and made the user learn a new set of operating commands. OK for programmers (I love learning a new OS don't you?), but yuck for the user. Starting with version 4, we moved GPA to native compilers on DOS and Mac. The application code was portable, enabled by a portability library we built at Great Plains. By using a different portability layer under the application code, we could allow GPA to interact with its underlying operating environment more directly. Still, GPA was essentially a text-based product. The Mac with its graphical user interface was gaining favor fast; Microsoft even "announced" Windows in 1983 (really, checkout <http://www.microsoft.com/Museum/default.asp>) and finally shipped a useful version (3.0) in 1990. By the late 1980's the handwriting was on the wall, Great Plains needed a GUI based product. Building on the idea of a portable layer, a small group in the research department began to build an Even Better (EB) system. EB became what we know as Dexterity.

With Dexterity, Great Plains had a portability layer that it owned. In fact, we owned just about everything! We built the development environment including debugger, profiler and source code control. Well OK, part of source code control. We owned the controls, the buttons, checkboxes, text boxes and such. Well, that's partly true too since we largely preserved the native look and feel of each OS, Windows and Mac. I could go on, but the point is that while Dexterity did a lot, we did have to pick and choose just what we put our effort into. I'm sure each of you has a pet feature you'd like to see in Dexterity, right? As Great Plains grows, we feel the need to provide well-rounded tools that have all the features a diverse group of developers require in a complex world.

So it's that technology mountain again. Even corporations can see it and have to deal with it. When Great Plains started out, we built our products on top of the P-System to provide portability because we couldn't afford to build it ourselves. Today, our next generation products will again be built on top of another company's products. You see where I'm goingNET is that technology platform. But of course this is nothing really new either, we built Dexterity using C/C++ compilers from Microsoft on Windows and Metrowerks and Apple on the Mac. We used database technology from Pervasive and Microsoft. And we've incorporated a number of other technologies along the way.

Why .NET? In a word, Microsoft! Think big company with a huge research budget, lots of very bright people, extensive knowledge of compilers, browsers, web servers, security issues, databases and on and on. On the other hand, when it comes to software for managing the financial side of a company, think Microsoft Money! NOT. That's where Great Plains shines of course. Easy, Microsoft provides the technology and Great Plains provides the application. Done deal. NOT again! Anytime any significant piece of software is written, there are always multiple levels of abstraction. That's where Platform Services comes in.

To use an obvious but not simple example, take security. .NET provides a security mechanism based on Roles. But how those roles are defined and used will be quite specific to a financial management application. Platform Services will build a security model that masks some of the complexity where it is not needed, entirely hides security where that is possible and exposes it where it is needed in a way that makes sense. Microsoft builds a generic security mechanism into .NET, but Great Plains makes it useful within the context of our industry.

Let's further refine the security example. Within the Great Plains application, including your enhancements and modifications, a lot of security information can be implied by the reporting relationships defined in a Human Resources module or in the structure of the Chart of Accounts. Based on this information, both user-specified and implied, it may be possible to automatically filter the data a user sees. By pushing the technology into the Great Plains platform, we can make applications more robust and developers, both within Great Plains' four walls and within the virtual organization a lot more productive.

For a nice overview of Microsoft's .NET technologies, check out:

<http://msdn.microsoft.com/msdnmag/issues/0900/WebPlatform/WebPlatform.asp>

The technology mountain is enormous but Platform Services will shave it down to size by providing an abstraction layer for all of our benefits. Same thing Great Plains has always done.

See you next time,

Karl Gunderson
Technical Evangelist