Blog copy – untitled!

Background

So, my basic task on arrival at SL was to get production-ready with C# and .NET. I’d taken a look at C# .NET, back when it was originally released, but my job (and the business that sprang from it) kept me working in C++ world for many years to follow. A few years ago, I worked on a migration project, taking a legacy C++ code base and deriving a modern-looking WPF-based replacement. During that time, I picked up much of the newer C# language features, and started to get to grips with WPF. Unfortunately, I was tempted away into the world of C++ contracting, and didn’t complete my C# .NET learning. My own experience writing in .NET has been limited to a few small applications.

It was originally suggested that I attempt to build a financial trading application, was briefly shown some existing software, and largely left to it…

The first thing that occurred to me was that I had no idea what I was doing with a financial trading application! Getting time off those who could have acted as product owners seemed unlikely. Mark also announced his intention to do brown bags about this subject matter, and so perhaps, I thought, I could pick up the world of financial trading after having conquered .NET? Surely the key to picking up the state of the art of .NET was actually getting stuck into a project, and thus I decided to find a problem that was already well-defined, and so would not require guidance from someone else. In effect, this enabled me to act as product owner for my own training project. Had I tried to do that in the unfamiliar territory of financial trading, I’d have had conceptual problems related to the business domain that prevented me from using (immersing in?) the technology.

So, looking for some familiar territory, I reflected on my past. It’s almost all C++, and that had been the choice of the various organisations I’d worked in, primarily for the same reason – speed. Real-time signal processing, bit-level stream processing, dealing with hardware and tight timing windows - you need to use a language like C++, for machine-level access to the hardware, and for the deterministic behaviour of unmanaged code.

C++ nerds (like me) like C++ because it’s really “close to the metal” – there’s no safety net, you’re responsible for everything, and everything boils down to

WPF and MVVM

* Bit of a story to tell on MVVm. Finding the “good stuff” using search engines isn’t straightforward. The first thing to do is understand the pattern, its motivation and its benefits. The second thing to do is understand how to take the WPF framework and implement the “missing bits” to achieve MVVm – Josh Smith’s MVVM Foundation slots in here and is an example of the RIGHT way of doing MVVm in WPF. (The guy from Infragistics (insert name) reviews the wrong ways people have come to implement MVVm, like deriving viewmodels from DependencyObject argh!, and this’d be on my list of three of four good vids on MVVM to watch.)

The problem domain

Given my background in real-time processing, and the emphasis on performance that comes from that, I decided I’d pick a problem that required some serious CPU cycles to solve. Something that crunched numbers and would lend itself to recursive methods – I went with a numeric puzzle that (almost) everyone will be familiar with, based on C4’s Countdown. The rules are well-known, and the algorithmic complexity can be increased at will (by increasing the size and range of the input values.) Brute force attempts at working out how to solve the “Numbers Game” would certainly provide a context for working with algorithms and data structures in C# - and because of the familiarity with the problem domain, I immediately had many ideas as to where this learning project could be taken in future phases.