Subject: Re: (re-)introducing Dependency Injection
Posted by DannyBoyPoker on Sun, 07 Apr 2013 01:01:27 GMT
View Forum Message <> Reply to Message

First you say 'The idea that you may want to swap out one menu system for another is pointless.' And then you say 'Finally, you ask the question "how to accomplish this?" How to accomplish what exactly?' It's not really that I don't want to use the Radicore menu system, but rather, that I may want, generally, to pick and choose from various components that all work from a common database. Here's the pitch:

Each module builds upon the others to provide additional functionality. These intelligent modules can even detect their peers and may offer additional functionality depending on what other modules are present. I have spent countless hours developing modules that perform specific functions for a web site on their own but because they all interconnect, the more modules you have, the more functionality each may provide.

Now, in such a case, User Management, and security, could be separate modules. You say: 'Radicore has a single built-in menu system which cannot be changed for another. If you don't want to use the Radicore menu system then don't use Radicore.' And, to this, I ask, what is 'core', in Radicore? I'm fine with it, if the basic functionality provided is fairly substantial, that's a good thing. And it is. But building on top of it, would mean, could mean various things, expanding the user interactive stuff, or the general information stuff, or business management stuff. eCommerce stuff.

And, the point is to discuss the problem a dependency. And, the the Dependency Injection pattern is only applicable in software systems consisting of separated components. The idea is that outer components inject dependencies into smaller components. Along with definitions of dependency, cohesion and coupling, we might try to find agreement, on what are the attributes to make something a component. I don't think it's true, that this doesn't matter to you, you are offering, in Radicore, components that in the general case have a limited scope of interest. And, these component instances cannot access properties of other components, unless you're providing a way to do so. And, your components are reusable and might be used in different systems in a way that the using system does not have to change code on the imported component to make it work.

If I say something like that in Radicore, maybe there are not enough components, then this only to ask for more Radicore, in Radicore. The idea is to not make it hard to write reusable software and test these in isolation, right? You give reasons why you don't use DI and that is fine (actually you do, to a great extent, I would have said, when I actually look at the code), as the point is not that couldn't possibly be a cleaner solution for dissolving dependencies. If you identify your dependencies, then refactor to components, as you have, largely, then you can proceed to keep your components dumb, and the DI pattern is just something that supposedly makes it easy.