Boo

Home Up Boo Unit Testing Boo Tips

Introduction to Boo

Boo is a great alternative to Python - almost the same syntax, with static typing and runs full speed in .NET

It is a combination of Python and C# and runs in .NET. Its fast and looks like an interesting and **potentially important synthesis** - you can have static typing plus the benefits of a concise syntax etc. It has type inferencing which means you still get static typing without having to explicitly declare every single thing (though its not *always* possible for the compiler to infer these things, so in those cases you do need to explicitly state the type.).

P.S. It even "fixes" the requirement to use 'self' in the declaration of methods on classes. ;-)

The synthesis of static and dynamic?

When I wrote the phrase "potentially important synthesis**"** about a year ago I suspected but I didn't really fully know how I could blend static and dynamic in the same language. Recently when I ported a reasonably sized python project to Boo I discovered what I think is an important insight.

During my port from Python to the static typing of Boo, when I first started statically typing things, things went well till 95% of my code was complete and then I found myself fighting with the static typing system (instead of it helping me). This reminded me of every project I have ever used that had static typing - at first its great until at some point you start wrestling with the type system and forget about the actual application you are trying to build. How many times have you been there too? Here we go again, I thought...

Then I solved the static type fascism by declaring a few key things as duck type (a variant type available in Boo). This allowed me to move through the problem area instantly. It was like opening a release valve - allowing the pressure out of the static typed pressure cooker.

A sprinkling of dynamic duck typing in Boo lets the pressure out of a static typed system

So perhaps *this is the way* we are going to resolve the great debate of static vs. dynamic. The solution is to have a static typed system by default, with lots of type inferencing (yes, Boo does this) so that you don't need to waste your time declare the type of every single little thing. Then when you get a touch of "Godel incompleteness" in the static type system of your application... just introduce a little duck typing to **release the pressure**. You may pay a slight performance penalty at these "duck moments", but since most of your app is blazing along at statically compiled speeds, it doesn't matter (furthermore, the places where you used duck typing may not be the time critical ones, anyway).

Boo has perhaps solved the static vs. dynamic debate by allowing both paradigms, allowing the developer to use the strengths and avoiding the weaknesses of each paradigm.

Advanced Ideas

Boo uses an Extensible compilation pipeline: you can actually extend the language by implementing a visitor (you have to be comfortable with AST - abstract syntax trees though). You can even create your own keywords using 'syntatic macros' - I believe someone has added a 'with' keyword and someone else has implemented 'design by contract' this way. A small blog overview of Boo by Daniel Turini's can be found here. Lot of interesting stuff also at the Boo wiki pages.

Latest download - Relationship Manager for .NET 

I have implemented RM Relationship Manager for .NET using the Boo language (porting it from Python). See it at RM for .NET
View the Boo source code of Relationship Manager for .NET online (color coded) on my Boo wiki page.

Pending research

Issues to to with unit testing using Boo.

Here are my tips for Developing & Debugging visually in Boo

Andy Bulka
http://www.atug.com/andypatterns