.NET Rocks!
The Internet Audio Talk Show for .NET Developers
With Carl Franklin and Richard Campbell
msdn
http://www.dotnetrocks.com

http://www.dotnetrocks.com

Carl Franklin and Richard Campbell interview experts to bring you insights into .NET technology and the state of software development. More than just a dry interview show, we have fun! Original Music! Prizes! Check out what you've been missing!

*Text Transcript of Show #298*
(Transcription services provided by PWOP Productions)

Pwop! Productions

# Gael Fraiteur on PostSharp
## December 13, 2007

***Our Sponsors***

Developer EXPRESS
http://www.devexpress.com

CoDe component developer magazine
http://www.code-magazine.com

RadControls FOR ASP.NET telerik
http://www.telerik.com/

**Geoff Maciolek:** The opinions and viewpoints expressed in .NET Rocks! are not necessarily those of its sponsors, or of Microsoft Corporation, its partners, or employees. .NET Rocks! is a production of Franklins.NET, which is solely responsible for its content. Franklins.NET - Training Developers to Work Smarter.

[Music]

**Lawrence Ryan:** Hey, Rock heads! Put down the torch and monkey wrench and listen up! It's time for another stellar episode of .NET Rocks! the Internet audio talk show for .NET developers, with Carl Franklin and Richard Campbell. This is Lawrence Ryan announcing show #298, with guest Gael Fratieur, recorded live, Monday, December 10, 2007. .NET Rocks! is brought to you by Franklins.NET - Training Developers to Work Smarter and now offering SharePoint 2007 video training with Sahil Malik on DVD, dnrTV style, order your copy now at www.franklins.net. Support is also provided by Telerik, combining the best in Windows Forms and ASP.NET controls with first class customer service, online at www.telerik.com, and by CoDe Magazine, the leading independent magazine for .NET developers, online at http://www.code-magazine.com. And now, the man who says, "I'd rather have a bottle in front of me than a frontal lobotomy," Carl Franklin.

**Carl Franklin:** Hey, thank you very much. Welcome back to .NET Rocks! This is Carl and Richard here with you again on this beautiful Thursday. Lovely, lovely New England weather we're having here. It actually sucks. How is your weather, Richard?

**Richard Campbell:** About the same.

**Carl Franklin:** Yeah. It's December, you know.

**Richard Campbell:** You think.

**Carl Franklin:** Kind of sucks, but then the rest of the year also sucks.

**Richard Campbell:** Christmas is coming and the kids are excited and we're spending all our money.

**Carl Franklin:** We just want to have some snow, that's all we wanted. None of this teasing us with a little snow and then it rains and then it goes away. Last, the snow... we had some snow that lasted five minutes. Seriously, five minutes it was all ice and sleep. All right, let's get to Better Know A Framework.

[Music]

**Richard Campbell:** So a fun one this time.

**Carl Franklin:** Yeah.

**Richard Campbell:** Okay, what is it.

**Carl Franklin:** This is the System.Resources.IResourceReader Interface. So you know resources, right?

**Richard Campbell:** Yeah.

**Carl Franklin:** Yeah, we all know what they are. They are these little blobs, they are text or different language text or images or things that just exist on these files out there, and you can usually read a user ResourceReader or ResourceWriter, just the once that defaults to read and write those. But if you have a Resource thought that was written using a customized writer in Reader and you want to read and write your own files in your own format, then you can implement the IResourceReader Interface which provides a base functionality to read data from a resource found. There's also an IResourceWriter. So there you go. You want your resource files in SQL Server, about to boom.

**Richard Campbell:** No problem.

**Carl Franklin:** Yeah.

**Richard Campbell:** So it's not that you are changing up the format of the resourced file, just being able to record it somewhere else.

**Carl Franklin:** Well, if the resource file was written using a customized ResourceWriter, then you want to use the Reader as well. So it just gives you an opportunity to control what happens when you use the Resource Reader and Writer. So there you go.

**Richard Campbell:** Excellent.

**Carl Franklin:** Have fun with that. Richard, have you got an email for us?

**Richard Campbell:** I do indeed. You know, not every email is love and this definitely one of them. "Hey, Carl and Richard. I just got down listening to your open-source DNR episode and though it was a bundle of laughs, I have to admit that I never heard more people get so cynical over development paradigm namely open-source in my life."

**Carl Franklin:** Oh, it wasn't that bad.

**Richard Campbell:** No, I didn't think so either, but give the guy's opinion here.

**Carl Franklin:** Okay.

**Richard Campbell:** "The whole show can best be summed up by the comment Richard made at 37.27, 'what are you saying when you say open-source, that I need more grief in my life?' You might as well have called show 296 the why we hate open-source show because it was basically what I was hearing from you." Which is funny because I put four people on the panel that were very pro open-source that most of them are making a living from open-source.

**Carl Franklin:** Yeah, well, I don't know. Maybe a little sensitive, maybe.

**Richard Campbell:** Maybe.

**Carl Franklin:** Just a little.

**Richard Campbell:** "I will admit that at 57 minutes, that one guy was talking about he was able to get FTP server running due to some abandoned open-source project, and for the briefest of moment, you all appeared to be talking about open-source at a positive light, and let's be honest, the majority of the show that played rough let's ragged on open-source."

**Carl Franklin:** I totally disagree.

**Richard Campbell:** Yeah, well...

**Carl Franklin:** We were holding it up as what it is. I mean we didn't have it jammed one way or the other.

**Richard Campbell:** No, of course we don't. Honestly, when you have four people on the panel that are using open-source in a serious way, you've got to ask those critical questions.

**Carl Franklin:** That's right.

**Richard Campbell:** I think from our point of view, we have to represent the audience who's wondering why they would use open-source. It's not automatically love.

**Carl Franklin:** Right.

**Richard Campbell:** You have to just make some clear distinctions as to why things are good, and I thought we made a few great points there.

**Carl Franklin:** But thanks for that email. We do read them all, good and bad.

**Richard Campbell:** You bet.

**Carl Franklin:** So, you got an email, send it to dotnetrocks@franklins.net and let's see, how about the SharePoint video, Sahil Malik's DVD that we're selling now at franklins.net? It's 9+ hours of dnrTV style training in SharePoint and lots of topics, a lot more than we covered on dnrTV. It is also a 1280 x 1024, so it's a little bit bigger and nicer. The audio is obviously nicer and there's a sample video that you can check out at franklins.net and also there's a detailed table of contents. So you can get a real idea for what you're getting into. We also have site licenses available, check that out. Also, the guys that Infusion are still looking for a few good developers, they're really just having a field day with our listeners. We got so many more listeners that have moved to New York because of these guys, but they're offering you to move you to New York City, to work in an exiting field with some very creative people, to live rent free in an apartment in Manhattan for a year and do the tour. If you're interested in that, go to shrinkster.com/kh6, and yes, shrinkster.com is now back online. Check it over and it's going very smoothly, so please enjoy. Well, let's bring on our guest today. Gael Fratieur is the founder and the project leader of PostSharp. He is a Microsoft Certified Solution Developer for Microsoft .NET, and has an advanced knowledge of Oracle Server, Microsoft SQL Server, TIBCO middleware, and UNIX platforms. Besides maintaining PostSharp, he is currently helping a Web 2.0 startup building high-availability and scalable solutions. So Richard, you should have a lot to talk about.

**Richard Campbell:** You bet.

**Carl Franklin:** Before that, he worked on back-office systems of a mobile operator. After completing a MSC in mathematical engineering in Belgium in 2001, he followed his Czech princess up to Prague, where they have lived so far. They got married and have two kids. Welcome Gael.

**Gael Fratieur:** Thank you.

**Carl Franklin:** Prague is beautiful, isn't it?

**Gael Fratieur:** It is, actually.

**Richard Campbell:** Yeah, Remi Caron and I and a couple of other friends went there, and Stephen Forte was one of them. We went there right after the STC conference in the Spring. No, it was in the Fall. It's a little blur already.

**Carl Franklin:** Right, right.

**Richard Campbell:** We didn't see you anywhere Gael?

**Gael Fratieur:** There's a big American company in infrastructure development. A big .NET shop in Prague, its Monster, that has hundreds of

.NET programmers. Maybe the biggest .NET shop in Prague is Monster.

**Richard Campbell:** We met several of the guys from Monster in Prague and talked about the wonders of developing software.

**Carl Franklin:** Let's get right into PostSharp. Tell us about it.

**Gael Fratieur:** PostSharp is a tool for developers and it enable to make more of less code actually and the idea is when you want, for instance, a method to be transactional, don't use side lines of codes to do it. It is easier to put just a person agreeable to the method and say this is a transaction, or more precisely this method is a transaction boundary which requires a new transaction. So PostSharp is a tool that allows you to annotate your code and in the background, doing compilation. It will add new behaviors to your code. So it will in fact, edit the assembly of the compilation transparently.

**Carl Franklin:** Okay.

**Richard Campbell:** Funny you would describe transactions that way. That reminds me of the way we did it in COM where we just use to setup properties and say this particular component requires a new transaction.

**Gael Fratieur:** Exactly. So the difference is that with PostSharp, you can do it with any method. You don't have to derive it from the Enterprise Company and Class, or you don't have to derive it from whistling object like it was a framework. This kind of transformation is called aspect-oriented programming because naturally, transactions are one aspect of programming.

**Richard Campbell:** The whole idea is to keep that sort of infrastructure code out of your main code and add it in a different way.

**Gael Fratieur:** Yes, yes, exactly.

**Carl Franklin:** Attributes are a really good way to do that.

**Gael Fratieur:** The idea is to separate functional codes solving business programs like in management and product management and so on. So to separate this functional code from nonfunctional code which is log-in transaction management, for instance caching and so on.

**Carl Franklin:** Yeah. So the idea, as Richard said, is to keep your code cleaner so that when you're working on the solutions, you don't have a bunch of plumbing code. Everybody has to write plumbing code to some degree, I mean, the framework made it easier and there's certainly a lot lesser of it now, but you still have to do that sort of maintenance code a lot and I guess what you're looking at here, is when you open up your project and you look at your code, you're focus on what is this thing doing from the business point of view not necessarily all that gook clogging it up.

**Gael Fratieur:** That's it. If you take a typical business method, you will have the business function in that, and then you will have four lines for log-in, and then you will have five lines for transaction management, and so on. So if you go down the lines, there is maybe 30% of functional code and 70% of nonfunctional code. It is in worst scenario of course, these rates.

**Richard Campbell:** Well, in recognizing every line of code you write is code you need to maintain, code you need to debug. I'm going jumping on the transaction issue again because how many times have I been trying to solve a problem where really what it was is we haven't really properly assign a transactional boundary. So the code was totally functional but weren't following the transactional rules and one part wasn't rolling back properly.

**Gael Fratieur:** There is a different term for this. It is property concerns. We use this term in infrastructure engineering to say that some concerns like log-in transaction management and so on, to look at all functional concerns, and in infrastructure engineering, we try to separate concerns which is the principal of separation of concerns and it is difficult to separate concerns for these processing ones. So solve this for this purpose, we have aspect-oriented framework like PostSharp or you have other frameworks link mostly the same but differently they're all pros and cons.

**Richard Campbell:** I want to drill in to a bunch of different aspects and so forth, but before we go down that path. Let's talk a little bit about how PostSharp came into being. It's freely downloadable, right? I don't have to buy it.

**Gael Fratieur:** It is free, it is open-source, it is released under three different licenses. The first one is the most regular license which you can use even for commercial purposes. You don't have to make your own "open-source" if you use PostSharp. There's no propagation of the license like in GPL, but if you want to use GPL components which is quite rare in .NET system but you can because PostSharp is also published under GPL.

**Richard Campbell:** Oh I see. So you actually have an interesting solution. You just have multiple licenses.

**Carl Franklin:** We're just talking about multiple licenses on the open-source panel.

**Richard Campbell:** Right.

**Carl Franklin:** Let's start with some of the basic AOP, aspect-oriented programming features, Dependency Injection being one of them.

**Gael Fratieur:** Dependency Injection is not directly related to AOP but you can simplify your work using AOP. I have a kind of test code on my computer where you can just declare fields with a cache and attribute and you say this is Dependency, and then the framework I can initialize the thing automatically and you don't have to use reflection, you don't have to use the shell scripters.

**Richard Campbell:** Yeah.

**Gael Fratieur:** So with AOP, you can do things like the Google dependency injection framework. I don't remember the name of this framework.

**Carl Franklin:** Interesting, right.

**Gael Fratieur:** You can do something very, very similar because PostSharp can do for you the code that will actually inject the dependency from the framework, but I'm not sure it was your question actually.

**Carl Franklin:** You know what, maybe I should rephrase the question a little bit different. So I guess PostSharp Laos is the AOP framework as you said, and let's talk about some of the features of PostSharp Laos.

**Gael Fratieur:** Okay. I think the most interesting feature and what makes PostSharp Laos very different is simplicity. It has very smooth learning curves. PostSharp Laos doesn't use only academic concepts of object-oriented programming. It used the vocabulary of every developer. For instance, an aspect is normal system attribute. There's nothing you have to learn, you just have to make a new system actually and to implement some methods that will be invoked at runtime. So for instance, the most common system attribute is on method boundary affect, and it defines four methods which is on entry, on exit, on exception, and on success. All you have to do is to make your own system attribute, your rise from on method boundary I'll take attribute.

**Richard Campbell:** Interesting.

**Gael Fratieur:** Then, you can override on entry and you're code on entry, so even handler, for instance trading some message or defining the beginning of a transaction scope.

**Carl Franklin:** That makes a lot of sense. Attributes are one of those things that I think doesn't come up in everyday development unless you actually think about how to use them effectively and you have a broader strategy. It is not just something that I think most developers will just go ahead and use. It is interesting that you're using them with the object-oriented features of inheritance and stuff that kind of makes a lot of sense.

**Gael Fratieur:** I think the servers are use to think, to get some attributes of pure annotations without any behavioral part.

**Carl Franklin:** Right.

**Gael Fratieur:** For instance when you use a system attribute for experimentalization. It is purely information for the analyzer. The difference with a PostSharp Laos is that this attributes really changes your code. If you take the code before and after with compilation and you use for instance the reflector to look at your code, you will see that the methods of your recent activity had been injected in your business methods.

**Carl Franklin:** Yeah.

**Gael Fratieur:** I think from the point of view of the first time user, it is very smooth to say that they just to code on some different attributes. You don't need an XML file on the side. You just install PostSharp on the platform, you just use it at the language field so they treat it as new and you get some attributes and big and that's all. I think this is truly the biggest difference here too. The reason for that is that it works as plain C# plain, VB Net or J# and so on because I don't need a new syntax.

**Carl Franklin:** Yeah. It's not language dependent in other words.

**Gael Fratieur:** Yes. It is not language dependent because other implementations use language extensions and so on, and this is not in the case of PostSharp. You really use .NET object-oriented programming generally and this has plenty of adventure of course.

**Richard Campbell:** Now, it looks to me Gael, like there are two products here. There is the PostSharp core and then there's PostSharp Laos and I can't use Laos without core.

**Gael Fratieur:** Yes, technically Laos is a plug-in to PostSharp core.

**Richard Campbell:** Okay.

**Gael Fratieur:** Because I wanted to separate two concerns. It is high level use of the platform for the end-user in the management of get some attributes and so on. Low level is working with .NET assemblies with the MSI representation and so on. Laos is relatively small, it has really a few total lines of codes that internalized on PostSharp core which is much bigger and PostSharp core can be use to make other solutions based on post compilation transformation of assemblies. So I know an object relation on network using PostSharp as a back-end.

**Richard Campbell:** The ability to inject yourself into the compilation process so you have it up for you to do your transformations.

**Gael Fratieur:** Yes exactly, you…

**Richard Campbell:** All that stuff should not core.

**Carl Franklin:** It's interesting to me that this all happen post compilation because that's not the way you typically think of toolkits that you want to use in a development environment.

**Richard Campbell:** Right. Normally, it's just code into your methods.

**Carl Franklin:** You make a reference to it, it gets compiled and it runs. How does debugging work or does it?

**Gael Fratieur:** In the beginning, it works quite transparently. If you use Laos and that into a method that has an aspect, you will get at the beginning of the method and if you go against it, you too you will get into the aspect, okay, and then we'll step back out in the method. Go to the end of the method and when you do step into, you will be in the on success method and so on. So at the beginning, it works transparently, there is no change from the user point of view and this is one of the differentiating point of those systems, that this works quite well.

**Carl Franklin:** How does that work though, I mean, is there some special magic happening like to hook the debugger and then execute the code, and for that matter, what's the benefit of post compilation injection?

**Gael Fratieur:** To have the debugger working, there's definitely no magic in that. It is really only your writings involve for the debugger to know that EDP files are, I mean, you write it and I just say that some part of code is non-user and that is all I have to do if it is weak, really there is no trick in that.

**Carl Franklin:** Well, that's good.

**Gael Fratieur:** One of the good things in PostSharp is that it creates a plain .NET assembly result and a track result. There is no special feature enabled if you… you don't need to load complex systems, sub-systems up there in time. You kinds use again the tools because it is just plain assembly which is a little modified.

**Carl Franklin:** Okay.

**Richard Campbell:** Fundamentally, you know you're available aspects are your hooks. In these event points where you're jumping in and running your code, are you able to actually grab those events, big toes connections after the code is compiled and you're intercepting mid-compilation.

**Gael Fratieur:** What I do is that in the core is disassembling all methods and injecting your instructions so there is lot of trades to understand the method, I have for instance to change if we turn instruction to a go to and it is not so easy to up that handler to a method and so on. This is done by PostSharp core. So based on this analysis, I inject instructions into code at multiple occasions.

**Richard Campbell:** You mentioned cache methods and I love to go down that path because I think that's a very interesting area, but just a sort of throw it out there, what's the overhead of adding this code? How much performance is it costing me to insert it this way as opposed to coding it yourself directly into the method.

**Gael Fratieur:** There are two kinds of performance point of view. The first one is compile time performance, of course it is expensive.

**Richard Campbell:** So that it takes longer to compile the application.

**Gael Fratieur:** Yes.

**Richard Campbell:** That's a one time thing.

**Carl Franklin:** Yup.

**Gael Fratieur:** I have a community member telling that it takes twice a time to process a big system.

**Richard Campbell:** Okay.

**Gael Fratieur:** So you have to pay something in performance at compile time. The second problem is at runtime. If you use strictly PostSharp core and that you could just have insertions that you inject in such in console with that straight line, you can make excellent code. You can easily choose exactly what you know and you can make hand tuned code, but if you want to use PostSharp Laos so it is easy and so on, then it is lower because every code for the aspect method, it is needed to create a new object. It is required also to box all parameters with the method, so in array and so on.

**Richard Campbell:** Right.

**Gael Fratieur:** So you cannot choose PostSharp to log an application which does not range in numerical computation. This would not be adequate because of the heat effect, it would be much too high. If you want to look at, of course a level of genericity, you won't see the difference, and if you compose PostSharp to competing solutions, you will see that there is a lesser over this PostSharp because I don't use remoting proxies and so on.

**Richard Campbell:** I think that a fairly key point here is that you are incorporating your code into the same assembly. You're getting rid of the other process calls by your approach.

**Gael Fratieur:** Yes it is, or in processes or in the same track the GIT compiler can in line eventually the aspect, and also I did not spend much time in performance optimization in the first release, but in the next release, the post compiler will be a little smarter and will not generate all the stuff if you only a little of this.

**Carl Franklin:** Tell me about the lifetime of an aspect. What is the life cycle like?

**Gael Fratieur:** This is also very specific to PostSharp. So aspects have two lives, it has one life at compile time and one life at runtime, and the real difference in PostSharp is that the aspects are instantiated at compile time so your constrictor is invoked at compile time inside the post compiler and then it is share a life inside target assembly, you know, there's a binary formatter and at runtime it is this share a life and it can be enrolled. This is really a great difference because at compile time, you can already do some job. For instance, if you have to compute something difficult at compile time, you can do it at compile time if the result of the computation depends only on the metadata of the assembly name of the method, for instance number of arguments and so on. Typically, when you have to generate a tree to look to the log in of the system, this strength is quite complete to generate. You can do it at compile time. You can freeze it and then so in an instance here of the aspect and on runtime you will see it available. This allows to make quite interesting things. You can also at compile times, validate the whole system attribute that had been used. I think it is something new in .NET that system attribute can have a validation method that the compiler, hey, you cannot use that method on static, you cannot use the system attribute on static method. You can use it only on instant methods.

**Carl Franklin:** Okay.

Hey, this is Carl. I just want to take a minute out of the show to tell you about Telerik's Q2 2000 Tools update which can be summed up this way: Blazing fast performance for ASP.NET, WPF like visual effects for Windows Forms, and codeless reporting. The AJAX-based content editor is now 76% faster and much more intuitive. The grid also received the performance boost, plus PDF export, frozen columns, and they've even added a new awesome scheduling component. What I find even more intriguing is Telerik's Windows Forms Suite. It is unbelievable that it offers WPF-like visual effects like scaling, rotation, object motion, transparencies and so on, without WPF. As a result, you could have grids, tree views, ribbons, and more with a previously impossible level of interactivity and appeal. Telerik has recently added CAB support, which makes the component setup a perfect fit for large enterprise applications. Lastly, with Telerik reporting, you can create advanced business reports in Windows, web, or PDF format using pretty much design time only. Wizards, expression builders and converters help you with the design, styling and integration. You'll also be amazed to see some unique features like CSS-like styling and conditional formatting. See what all the fuss is about. Download a trial at telerik.com, and don't forget to thank them for sponsoring .NET Rocks!

So what are some of the aspects in Laos?

**Gael Fratieur:** So we talked already about the on-method boundary aspect defining on entry, on exit, on success, and on exception. These aspects more decide the method to which it is applied. That means that you cannot, for instance, apply these aspects to a method in the framework because you cannot modify the assembly of the framework. But what if you want to do it? You can but you think of a different aspect which is on method in location and this aspect has single method called an invocation, and all you get in the event argument object is a delegate to the method being called and the array of arguments. So actually you can cache and you may control to any assembly. So typically, if you want to look all code to the namespace system threading, because you know, nobody can cope perfectly with that namespace just with one line of code you can look all codes. Third aspect type is method implementation and it is quite a surprising aspect that using this method implementation, you can design, for instance, store procedures, database store procedures as external methods in C#.

**Carl Franklin:** That's interesting.

**Gael Fratieur:** You will say that the handler for this external method is my aspects.

**Richard Campbell:** I can see the power of that in I have this existing application that I don't want to touch but I want to override some particular method and add a new capability to it.

**Gael Fratieur:** Yeah. So in that case, you just say to add on your aspect and you don't have to edit every class. You just say I want to apply this aspect to all the methods having this name or in this namespace. So far, we told about method little aspects that also same level aspects and five level aspects. Same level aspects are also intercept load and store operations on seal.

**Carl Franklin:** Now, these are not properties. These are fields.

**Gael Fratieur:** It is a field because from the CLR point of view, there is no property in methods.

**Carl Franklin:** Sure.

**Richard Campbell:** Right.

**Gael Fratieur:** You can, for instance, add validators to fields or you can make your class transactional. You can raise an object which is itself transactional and you can make in memory transactions. For instance, the book of Ralph Westfall, he is doing the in-memory transaction system for .NET. It is based on PostSharp. You define a field, you think it is a finite, but actually you store it in a dictionary.

**Richard Campbell:** That's cool. Ralph Westfall, we have never had him on the show before.

**Carl Franklin:** He is an RD.

**Richard Campbell:** He was one of the guys who is really into object spaces, that when they install, ingested.

**Gael Fratieur:** So the idea, in case of same level aspect, the idea is that a field is just semantic. It is just a storage location and you can do get a load operation from that. With PostSharp, you can change implementation of the semantics. You can have your own gut feeling, your own load field. It's the idea.

**Carl Franklin:** I'm thinking purely from the listeners' point of view. Somebody is saying, well, that's cool, but where is the practical application of this? So give me a scenario in which this is a desirable solution.

**Gael Fratieur:** In a business application, sometimes you have two copies of one object. You have the committed copy. So the copy of the object which is equal to the committed representation in the database, and you have the working copy which is valid inside the current transaction or inside the current UI session.

**Carl Franklin:** Right.

**Gael Fratieur:** What you do when you click on the safe is that you copy the working copy to the committed copy and when you want to implement this as manually, you have to code much insertion by hand and with PostSharp because it is quite easy just to say when I do commit… so the object is copied, automatically the… just to say that the field is not a field but is actually two fields, and it is hidden from the business programmer. I remember when making a graphical client, I have really to fight to encode such a behavior to be able to have, in the user session, a different object than the one in cache for instance. So with transaction isolation, it is truly useful and I think to use it quite often in business applications.

**Richard Campbell:** A lot of times I look at AOP and think this is… two groups of people talked about this, one say that AOP is only useful for testing and debugging, like making it dirt simple to instrument the system threading libraries so I can find out why my mound of threading code is killing.

**Carl Franklin:** Without having to rewrite the code.

**Richard Campbell:** Right, without going through the whole thing, or just general log-in and so forth. The other group of people are the ones who say this is about me as a consultant stepping into an existing application and avoiding touching the code directly but being able to slip in and around, learn how it works, and make changes from a distance essentially.

**Carl Franklin:** Well, then of course, there's just a whole logical decoupling thing, right? I mean, that's really what the power of using AOP in everyday code is all about. Isn't it?

**Gael Fratieur:** I think so, but we should make a finishing not to overuse or to misuse AOP.

**Richard Campbell:** Right.

**Gael Fratieur:** It cannot solve the same programs or sometimes I read in PostSharp for users trying to solve using AOP same way in which to solve using normal object-oriented programming always isn't handing is wrong. AOP is not for every concern as most of things cannot be solve using object-oriented programming. I recommend AOP for a nonfunctional requirement. Some different people recommend AOP also for functional requirements and I… so really invasive in its aspects. Using PostSharp, you are forced to design aspects so that aspects don't know their target code. They don't know look how variables… so pay attention not to misuse PostSharp, not to misuse AOP, and respect also engineering principles, respect separation of concerns, and in the Java eco-system it is much, much longer concept and they have a lot of practice with it and users come with feel about that it's more difficult to program with aspect because aspect has to know the target code. They have to know the implementation of target method and so on. Target method has to know they will be affected. I think this is a very bad design, and when you plan to use AOP, I recommend to design your base code and your AOP codes so that they don't know about each other. It's really important not to get completely mad with a small change.

**Richard Campbell:** All right. So I'm thinking about a scenario, actually this came up just recently in the field. I was talking to a group of folks about recovering from a cluster fail-over in SQL server. There's a reality that if you have a database server fail and it's in a cluster, there is still about a minute or so where the database won't serve transactions while it gets itself back in order, and you have to code for that if you really want to seam this fail-over, you have to retry transactions and one of the discussions we had around during that was is this something I could do as an aspect to add into the functionality, say I want to… the transactions failed, we think it's a cluster fail-over. I want us to retry the transaction.

**Gael Fratieur:** Yes, absolutely. It's a good use case of AOP.

**Richard Campbell:** What I like about the AOP approach is this throughout their entire code. It is cross-cutting like you describe before. It is everywhere and you don't want to drop in to each method that maybe accessed in the database and have to add that same glob over and over again to retry. So the aspect in that scenario then would be an on exception aspect?

**Gael Fratieur:** If you wish to retry, it would be an invocation aspect because you really need to call them into the game.

**Richard Campbell:** Right.

**Gael Fratieur:** On method invocation aspect.

**Richard Campbell:** That's interesting because when you're handling a failure but on exception is only when you're not going to retry.

**Gael Fratieur:** Yes, because unexception means that it adds a try cache and you get the chance to do some thinking that cache had broke.

**Richard Campbell:** Okay.

**Gael Fratieur:** Okay, but you don't have a four loop on the double dot.

**Richard Campbell:** Right, a do until kind of loop.

**Gael Fratieur:** Yeah.

**Richard Campbell:** To team up with the overriding invocation, I can actually put a loop around this.

**Carl Franklin:** I imagine that would be good for debugging real-time stuff. I've had my hand in it eversince I started programming.

**Richard Campbell:** Like Gael said, the whole monitoring system threading, I thought of you immediately Mr. Franklin, because just being able to have a real-time log of what's going on when you're multi-threaded…

**Carl Franklin:** Yeah, I mean, I've written that stuff but it usually involves code.

**Gael Fratieur:** What is good also is that you can add your aspect in one minute and remove it immediately.

**Carl Franklin:** Right.

**Richard Campbell:** Lovely for diagnostic. I mean in a consulting role, to drop in to somebody else's code and then quickly stick these features in, do recompile, watch it run, and now you understand what's going on.

**Carl Franklin:** You know the trace output is really good for that but it sprinkles this little trace and calls all over your code and when you're trying to look at a code that's tight in real-time, you really don't want stuff getting in the… oh no, I don't have to worry about that. You sort of like parsing it in your brain as you're debugging in your mind just reading the code, and I can see a clear advantage to getting rid of that stuff clear. I mean you might not think it is such a big deal, I just skip over it, but every line of code, now you have to evaluate is this significant.

**Gael Fratieur:** Also, I think diagnostic is a good start point because you really probably remove it from the release so there is no product twist with that, and when you start a new technology, you very often don't want to take... to commit a risky project. You need first a first project to take confidence in the technology and that's why diagnostic is a good start point to AOP.

**Richard Campbell:** It sounds to me like people started out just with log-in in AOP and gradually finds other opportunities, other cases, and I guess the area that I really want to push on was the places where AOP should go into production, which actually a better way to code those features. The best ones I've seen so far is transactional boundaries, but I know you mentioned caching at least once.

**Gael Fratieur:** Caching yes, because caching is a simple method. You have to compute the caching key based on the method parameters, then you have to make cache look up and if nothing cache, you have to do the business function and at the end of it, then cache.

**Richard Campbell:** Repopulate the cache.

**Gael Fratieur:** Yes. That's it, you have an implementation, but then that new rise of codes plenty of times and you can just make one aspect of that, you can implement the general way to encode the caching key based on method parameters and you implement cache look up and cache populate and that's it.

**Richard Campbell:** It's not a lot of lines of code but it decorates everything you do over and over and over again.

**Gael Fratieur:** Exactly. It's maybe one hundred times straight lines of code.

**Richard Campbell:** All right. I'll tell you a little story that's related to this that Shaun Walker, the .NET new fame related to me, and it was around ASP.NET caching which I have been living and breathing lately. He figured out working with Kent Alstad, that he actually has to test for the existence of cache items twice. So when you have a very high velocity, applications running, so these multiple people are requesting the same cache objects at the same time. While that's going on, you've got the same pattern going where you're saying is the cache over populated? No. Well, it's time to go populate it, but I don't want other people populating it the same time so write a sync clock or I'll put a lock in place to stop other people from entering that code so they now want... execution is actually loading the cache item. Meantime, a bunch of others are trying to do the same thing, so they are all sitting at the lock, and then when the first one finishes and the lock ends, all the ones that are waiting now execute the code as well, but they execute it one at a time because they just reinstitute the lock. So the important part was after the lock, now check to see if the cache item is populated again. If it is, then great, just go get the cache items. So all those blocked ones, will just use the existing cache items.

**Carl Franklin:** I thought the cache object had some sort of locking built in to it because you don't have to lock it like you do the application object in all the ASP.

**Richard Campbell:** No, you don't but this is not the issue here. The issue here is that the executing code currently believes the cache item is empty and it's going to go and try to repopulate it.

**Carl Franklin:** So when you actually execute the line of code to add or remove or check, that is a locked process, but in between that line of code and five lines down, that's not locked.

**Richard Campbell:** Right. So the funny part of the story where I think the assertory is very powerful, so Shaun figures this out and in his open-source project he laced in everywhere, this clever pattern that actually does all that cache management correctly and puts it out there, this is open-source, and then for the next three months keeps getting people removing the extra 'if' because it is obviously superfluous because they don't understand the code and then checking it back in going, "yeah, I moved that extra if, it was everywhere, I don't know why. I spent the day cleaning it up." I'm wondering if it had been an aspect so it hadn't been in the body code if it wouldn't have been the same problem.

**Carl Franklin:** Oh, do you really have to put comments? "Don't remove this code!"

**Richard Campbell:** This is what they're doing and people are removing it anyway because it's not intuitive. Until you're actually that deeply into the problem, you don't get why this is there so I think there's a real powerful part about isolating it, hiding it.

**Carl Franklin:** Good point.

**Richard Campbell:** What do you think Gael? Does that actually work in an aspect oriented model? Would be able to protect that bit of functionality?

**Gael Fratieur:** The developer should just think, oh I need to cache a portion of this method and just put an attribute and they would not care about the implementation of that cache attribute. They would care about telling the dependencies using the attributes for instance, and so on. They would care about that but not about the implementation.

**Richard Campbell:** Maybe we need to speak a little more about the implementation because I know though that I quite understand exactly how I would add this code in. So I've got to write my generic caching handler and then what's the process look like that connects that app to all of these methods?

**Gael Fratieur:** So in this case, you could use, for instance, on meta invocation. It will be the easier, not the most fun but the easier and so you add a new cluster in your project. You name the class cache attribute and you derive it from meta invocation attribute. You'll probably write the method on invocation and there you get a parameter event argument and on event argument there is a proper key named delegate and you get the delegate to the meta being actually called, and you have second method and the event arguments and it is get arguments. There you get the original arguments, and what you would do there is first, in the on invocation method you hold, first compute the caching key based on the method name and based on the input arguments. You would check the cache, for instance ASP.NET cache, and if it is not in cache and you hold the delegate, where is the parameters you got?

**Richard Campbell:** Right. You just go populate.

**Gael Fratieur:** Then you would put it back in the cache and return to the collar the result.

**Carl Franklin:** Is this attribute basically gets called for every call and you have to say "oh if it's this, if it's a call to the cache object, then I want to hook it" or do you associate that with the cache call somehow beforehand.

**Gael Fratieur:** No. It could be on the level of the user method in this case. You...

**Carl Franklin:** I get it, so whatever the user method is that's calling the cache object, that's where you decorate the attribute.

**Gael Fratieur:** Yes, and if you need...

**Carl Franklin:** Do you tell it that this attribute applies to Cache.Insert for example? Where does that association get made, in the attribute itself?

**Gael Fratieur:** Just infusion between what?

**Carl Franklin:** Because cache did not insert it, just a framework call. It's not something that you have the source.

**Gael Fratieur:** Yeah. This framework call you called the ASP.NET cache if you use this or the carefully underpriced language you find.

**Carl Franklin:** So it's an attribute on the actual call to the method itself.

**Gael Fratieur:** It's an attribute to the user method and in this attribute, you will call the cache.

**Carl Franklin:** I still don't understand how they get associated with the call to Cache.Insert let's say.

**Gael Fratieur:** Okay, how it gets associated with the method, the user method.

**Carl Franklin:** Right.

**Gael Fratieur:** Yeah.

**Carl Franklin:** No, not the user method. With the method that... like Richard is trying to say anytime in my code, somebody calls Cache.Insert, I want this code in the custom match execute. How is that association made?

**Gael Fratieur:** Okay. No, this would not work.

**Richard Campbell:** Yeah, but in this scenario, I'm adorning the methods that would normally go to the database is to get the data.

**Carl Franklin:** Okay.

**Richard Campbell:** Saying, instead I want an opportunity to populate, I check for it, the addition of the cache item.

**Carl Franklin:** Okay.

**Richard Campbell:** Although if I understand properly, using that invocation technique, I could intercept the cache, any reference of the cache object, if I really wanted to. I don't know that we do it, I wanted to do but there's definitely a way to go about that.

**Carl Franklin:** Okay.

**Richard Campbell:** I'm feeling little heavy with power right now.

**Carl Franklin:** Yeah.

**Richard Campbell:** I can get into and overwrite anything. I'm just thinking the power of, say you're playing around with cryptography and there's some new amazing cryptographic method out there that sometimes you want to really be able to use that and I've existing codes that's using the cryptographic libraries and it works just fine. I don't want to break it, but I want to add this new capability so the fact that I could intercept calls to that cryptographic library and then most of the time I would just pass it through on the interception as I say, "Go ahead, run the original one," but when they requested this exotic new method, I would be able to jump off and use that other encryption method and then pull it back in

**Carl Franklin:** Well, you know, this interception isn't anything new, but what's new is that now you don't have to change your... you don't have to make derived classes.

**Richard Campbell:** Right.

**Carl Franklin:** You can stay within the framework of the objects that you're using. To do this with object turning with inheritance you basically derived from these based classes and then you just make calls to your derived class, but that can be a problem as we all know, when certain methods expect certain types and casting and all of that stuff, sort of muddy to water to that.

**Richard Campbell:** Even more relevantly when you have classes that are locked down and you just know how to hook it that way.

**Carl Franklin:** That's right, there's no way to do it. Yeah.

**Richard Campbell:** So hooking it at the IL level just opens all that up.

**Carl Franklin:** Of course you can hook sealed classes, can you not?

**Gael Fratieur:** Sealed classes, yes, yes, because actually what it does when the code is not in your assembly, for instance, it is a framework method, it hooks of course. It does not modify the method. It hooks the core.

**Richard Campbell:** Right, that's the magic here, is your hooking the connection between the methods. You don't actually hook in the methods themselves.

**Carl Franklin:** Yup, I get it.

**Richard Campbell:** Only on request.

**Gael Fratieur:** Yes, but I would pay attention to behavior to methods that you are going to define on your own because you hesitate here also to keep the capability. It is something to add log-in to all namespace but to change the result of a method because you just want something new is very dangerous because when you have...

**Richard Campbell:** So what you're telling me is my cryptographic examples are really bad idea.

**Gael Fratieur:** It's really a bad idea.

**Richard Campbell:** I'm okay with that. Don't ever do this at home.

**Carl Franklin:** Which is struggling to find an example.

**Richard Campbell:** I agree with you Gael that the first thing I would do this capability to instrument so that I can understand what's going on.

**Gael Fratieur:** Yeah.

**Carl Franklin:** Very cool, very, very cool.

**Gael Fratieur:** Then you can add behavior but don't change semantics of your code. Don't make a method at multiplying sizes. You can do it but it's quite stupid, but you can make to augment the transition if you want.

**Carl Franklin:** Let's talk briefly about PostSharp4EntLib.

**Gael Fratieur:** Yeah. PostSharp4EbtLib is a kind of glue to use PostSharp with the policy injection application block from the enterprise library.

**Carl Franklin:** Okay.

**Gael Fratieur:** The incision policy application block is an AOP framework also. It is ship with enterprise library. It has a lot of tactics using the other application blocks as free application library like log-in, transactions, exception argument, but it comes with one technology for reading and it is remoting proxies, that means that it will work only if your code is derived from the object marked by your rest object. This is quite an imitation because that means you cannot use a twist with existing tools. You have to change for every class to make it inherit this, and if you don't want to derive your code from much of direct object, you have to expose interfaces explicitly. That's one program, but you cannot use a twist and private metals, static metals and so on. On the other hand, and it is a benefit of this approach, you can do it at runtime, it means that when the application is deployed, you can change log-in policy or the exception handling which with PostSharp you can use this application block with any metals of your program, even static, even private but if you don't at compile time... so it will be softer a little, it will work with all your methods but statically done at compile time. It won't work, you won't be able to change it at runtime.

**Richard Campbell:** Well, this gets back to the original issue we were talking about being in process, out of process to invoke these aspects, but as soon as you are in remote proxies, you are incurring such a significant overhead for your work that you're really undermining performance in a big way.

**Carl Franklin:** I see that it is a list of planned features in the show notes that you send us, support for mono, the compact framework assembly redirections, more aspects. Can you talk about any of those, sir?

**Gael Fratieur:** The support for mono actually is DOM, post mono, mono doesn't completely support postgres.

**Carl Franklin:** What's that?

**Gael Fratieur:** Simple sort compact framework, of course, itself will not run on the compact framework because there is no compile written on this platform but will target compact framework. That means that you can make applications for the compact framework.

**Carl Franklin:** Cool.

**Gael Fratieur:** So possibly, when your direction is more technically detailed, currently you cannot transform an assembly using the direction between different versions of the .NET framework, for instance from version 1 to version 2, and it is a limitation when you use third party components still linked for old frameworks. I also plan to spend more time in performance improvement because the first version was focused on functionality. The second one will make smarter code generation, it won't generate cache handler if you don't implement on exception little friends. Currently it does.

**Carl Franklin:** Okay.

**Gael Fratieur:** Finally, there will be more at hot aspects and the profit of signal from the community...

**Carl Franklin:** What's the website?

**Gael Fratieur:** Postsharp.org

**Carl Franklin:** Okay.

**Gael Fratieur:** Users frequently asked, for instance property level aspect, and it's quite easy to do based on PostSharp core, just waiting for the next to leave or for a different injection or this.

**Carl Franklin:** Okay. That's great. So, any last minute things that you want to mention or shout out sir, before we sign off.

**Gael Fratieur:** I'm looking to this. I think this is sort of everything

**Carl Franklin:** Gael, thank you very much for joining us on the show today.

**Gael Fratieur:** Thank you.

**Carl Franklin:** Its great stuff and I hope you'll get a lot of our friends testing it out and checking it out for you.

**Gael Fratieur:** Okay thank you.

**Carl Franklin:** All right, and we'll see you next time on .NET Rocks!

[Music]

**Carl Franklin:** .NET Rocks! is recorded and produced by PWOP Productions, providing professional audio, audio mastering, video, post production, and podcasting services, online at www.pwop.com. .NET Rocks! is a production of Franklins.NET, training developers to work smarter and offering custom onsite classes in Microsoft development technology with expert developers, online at www.franklins.net. For more .NET Rocks! episodes and to subscribe to the podcast feeds, go to our website at www.dotnetrocks.com.