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 #672
(Transcription services provided by PWOP Productions)

Aspect Oriented Programming at NDC
June 16, 2011

Our Sponsors

http://www.telerik.com

http://www.gcpowertools.com

Lawrence Ryan: .NET Rocks! episode #672, with guests Gael Fraiteur, Donald Belcham, Philip Laureano, and Krzysztof Kozmic, recorded live Thursday, June 9, 2011.

[Music]

Lawrence Ryan: This episode is brought to you by Telerik and by Franklins.Net - Training Developers to Work Smarter and now offering video training on Silverlight 4.0 with Billy Hollis and SharePoint 2010 with Sahil Malik, order online now at franklins.net. And now here are Carl and Richard.

Carl Franklin: Hey Oslo, welcome to .NET Rocks! There must be 40,000 people in this room. God.

Richard Campbell: It's a full room.

Carl Franklin: It's great.

Richard Campbell: Yeah.

Carl Franklin: What a great turnout. How many of you, by applause, listen to the show? Applaud if you listen to the show.. That's awesome.

Richard Campbell: That's why I love Norway.

Carl Franklin: We don't even get this response in New Jersey. Well, Richard and I are here today with an esteem panel to talk about Aspect-Oriented programming. So before we get into it, I just like to let the panel introduce themselves starting with on the end. Don, please.

Donald Belcham: I'm Don Belcham. I'm a consultant out of Canada. I do primarily C# development and I used to run the user group there. I dropped off the map a little bit now, but still doing a lot of code.

Richard Campbell: The user group in Canada?

Donald Belcham: Yeah, the user group __________ 02:01.

Carl Franklin: You run a lot of them?

Donald Belcham: No, no, one of them.

Carl Franklin: How many beaver pelts a month does it cost to be a part of the user group in Canada?

Donald Belcham: It's beaver pelts and maple syrup.

Krzysztof Kozmic: I'm Krzysztof Kozmic. I work for a company called Readify as a consultant. They are based in Australia, and I also contribute to an open source project called Castle DynamicProxy.

Carl Franklin: Yes.

Richard Campbell: Hmm, yeah. Very good.

Gael Fraiteur: My name is Gael Fraiteur. I'm the creator of PostSharp which I started working on six years ago and progressively it became much more and more time __________ 02:43. I created a company named SharpCrafters that continue development and sale of the product.

Richard Campbell: Okay.

Philip Laureano: My name is Philip Laureano. I am the author of LinFu, LinFu AOP, as well as the other half of the proxy library NHibernate which is LinFu DynamicProxy. Basically, I'm an IO geek and going to be working with Krzysztof in the same company producing...

Krzysztof Kozmic: Are you out of Hong Kong?

Philip Laureano: Moving to Australia.

Richard Campbell: Moving to Australia. We really got around the world representation on this panel, and Canada.

Carl Franklin: There's always the "and Canada" __________ 03:18. Well, I've talked about AOP on the show before but never really experienced it until I sat down with Gael at the New York code camp. We did a dnrTV where he showed PostSharp and my first reaction was why isn't this just in the .NET framework? Why isn't this just an option in Visual Studio for development? It seems like a no-brainer to me. For those who aren't familiar with AOP, why don't we just start there? Somebody define it for me.

Gael Fraiteur: So Aspect-Oriented Programming is a slightly different way to look at software application development. The first thing to identify that, there's a category of program that can only address correctly this AOP and it is older surface with tracing, exception handling, transaction handling, NotifyPropertyChange. These are all a class of problems that require us to make a bowl of great code. So Aspect-Oriented Programming look at inference as so called cost-cutting concerns and let them do code that implement just that and then apply this code to all the business code that needs to be transformed. So first we describe the transformation of the code, then we apply the transformation to the business code and because there are two phases we separate all the aspect code from the business code and we get nice separation of concerns and principally nice business code.

Carl Franklin: The way I saw it was you're hooking at a fundamental level things in the framework and then inserting some code at a very low level so it's just gone in one place and then it sort of takes it off the plate for everywhere else in your project. Just by decorating your object or your classes with some attributes, some code happens at compile time that you don't have to worry about. It takes it away. It takes it off the plate.

Richard Campbell: This is sort of constraining what you want to do with Aspect-Oriented Programming. Only cost-cutting concerns is the stuff you're going to do in it. What shouldn't you put in the aspect?

Donald Belcham: Probably the easiest thing not to put in is business logic. It's really, really hard to find business logic that's cost-cutting in my experience. It seems to be more infrastructure-base stuff so it might be logging to the prime example, but it's anything like that where you needed it in your app but it's not something where the business says, oh yeah, when we sell three of this you get a discount type of stuff.

Richard Campbell: Right.

Carl Franklin: Multilevel Undo was another one that...

Donald Belcham: Competitive code generation of things that you see all over the place and you can just generate it instead of manually coding it and things like that.

Richard Campbell: So there's sort of key things that behaviors just pop out and you say this probably should be an aspect, and I'm thinking we're now repeatedly cutting/pasting the same chunk of code into every method.

Philip Laureano: Well, and generally if you're cutting and pasting anything, that's -- they're badly counting on that.

Carl Franklin: Clipboard Inheritance we call that.

Philip Laureano: No. But in general, when we talk about Aspect-Oriented Programming it's not just these pieces of code. It's also cost-cutting structures. One class of code, for example, is INotifyPropertyChange. Everybody has to implement it. Everybody hates implementing it. But at some level you have to be able to implement this and you can't really do this with code per se because it has to be done with some tool or maybe something that compile this. The other angle of this one is that Aspect-Oriented Programming exists because of compiler deficiency. There's no way to clean up all these concerns that are scattered all over the place so what happens is you have to come up with third party tools that are able to do this kind of clean-up.

Carl Franklin: Yeah. INotifyPropertyChange obviously happens with in-binding when you're bound to a class to a UI element let's say. A UI element needs to know when that value has changed so they can refresh it. So anything that you're using in Silverlight or WPF that's bound, a bound object, you're going to have that issue.

Richard Campbell: Folks, if you have any questions, by all means just raise your hand up. We'd love to hear from you. And rumor has it I have licenses for PostSharp to give away for good questions.

Carl Franklin: We have more products here other than PostSharp represented and we should let other people talk about their products.

Richard Campbell: Hey Phil, what are you working on that paper...?

Philip Laureano: I was working on two things. I worked on LinFu .NET with proxy, the thing NHibernate. I liked it so much before too. The other part was a LinFu AOP which did some pretty crazy stuff, which was runtime interception of basically method calls and being able to change any assembly even ones that you don't own so you could intercept exceptions, third party method calls. You could change and swap out method bodies at runtime, things like that. So that was the kind of thing I was working on a couple of years ago until I switched back to IoC.

Carl Franklin: That's a little dangerous, runtime interception. So, not just of your code but any code, any .NET managed code?

Philip Laureano: Pretty much, as long as I could access the assembly.

Carl Franklin: Yikes.

Richard Campbell: This is one of those interesting truths, generally speaking, the implementation of Aspect-Oriented Programming, you need some cool tricks at the IL level to make it happen.

Carl Franklin: You have a question? Speak loudly.

Male Audience: How do you the __________ 09:16?

Richard Campbell: So, how do you do the interception?

Male Audience: Yeah.

Philip Laureano: Well, the interception itself is at the library. There's a library called csoap that is basically a reimplementation of System. Reflection, and unlike System.Reflection it's a read/write type of Reflection. So you could look at the method body, you could parse individual instructions and then add your own instructions. So for LinFu, what it does is it takes the existing method body and wraps the method body in basically what is "If" statement. When the "If" statement says if you want to intercept it, call this other method outside which is probably going to be an interface that somebody is going to implement which is to be like an interceptor if you're working with a DynamicProxy. So when it calls that method, it is basically up to you to provide behavior or you could delegate it to what the old method was actually doing.

Carl Franklin: Has anyone ever used that for various purposes?

Donald Belcham: What are you thinking of?

Carl Franklin: Well, I'm thinking like you could write of an album of a virus that way or a worm or...

Philip Laureano: It's just another tool in the toolbox.

Carl Franklin: No, no, no. I'm not saying it's bad inherently. I'm saying that, you know, it's like a really cool thing. You could do evil things.

Richard Campbell: Absolutely. I just find it interesting that we tend to forget after 10+ years of developing in .NET that there's compiling and then there's compiling, that these assemblies, generally speaking, are relatively accessible and overwriteable, manipulable as necessary. Generally, we can do good things with them. I'm sure it should be self-informative too. Gael, have you run across some of these and done something horrible with your tools that made things worse?

Gael Fraiteur: I don't know.

Richard Campbell: Only good things come.

Gael Fraiteur: Originally yes. There is a project I've seen on CodePlex or is it on Google Code. Anyway, you can write validation of fields or properties or parameters using custom attributes and you would write a string inside. So a string is an expression being validated. So, the aspect generates C# code based on the string, calls the compiler, gets an assembly, integrates assembly inside the current project and at runtime the assembly is loaded dynamically and this validation code in a string is being executed. This is quite original, I think.

Richard Campbell: This is the almost compiler as a service effect.

Gael Fraiteur: Yes.

Richard Campbell: That they've done with an aspect.

Philip Laureano: Yes. So this is one of the craziest things that I've seen.

Carl Franklin: You could have a lot of fun on April Fools' Day. Think of that, man.

Richard Campbell: A question here.

Male Audience: Yes. I'm just curious about how do you derive Test Driven in the absent or in the missing aspects? What I'm trying to say is how do I verify that an aspect is missing from a method call? So, for example, if one of the aspects that exist is tracing or performance counting for that matter and I need that on a specific level of client project, how can I ensure that those aspects are being used?

Gael Fraiteur: So you have to test your aspects directly from the rest of your code. You would need to test that the conditions that your code in the aspect are actually of type. So in the specific of your question, how can I test that the aspect has been applied or not on the method, then you would need to reproduce the same kind of commission and then test it. So this specifically is quite complex, but is not more complex than testing with certain aspect if your code is hard-coded.

Male Audience: It's very easy for me to verify. What I'm doing at the moment is that I have one of the aspects that I'm using. It's that I'm, for performance monitoring reasons, I'm logging the call and counting that particular method called and it's very easy for me to do that because I'm just using some of my event handlers to just pull some of the aspects that's caused by someone else. But if I do that in an Aspect-Oriented way and attributes that method with something that would do the same, then it becomes very hard for me to test that with a unit test for example. That's one of the things that's preventing me going to...

Gael Fraiteur: Yeah. There is what is __________ 14:03. What you could do is to look at it because there's so much that is there on the method for instance, or to look at the aspect simplified to see if has been applied but it's not standard. So I have many questions about, I get many questions about testing those aspects and it's definitely possible but it's a little different than the...

Donald Belcham: I find that testing the aspect itself, what the aspect should do is very easy. Testing the configuration and application of the aspect is way, way trickier. So what we've gone through on my current project is strictly convention-base. So we say this aspect is quite in this namespace, and that's how we test that it's there. If it's in that namespace, then we've got that piece of configuration and we're good to go. So we don't actually write an end-unit test. That's an example. It's very visual testing. It's the only way we've been able to do it in any consistent way.

Carl Franklin: And Krzysztof, we haven't heard from you about Castle.

Krzysztof Kozmic: Yes. Actually that's quite easy to do with DynamicProxy because what you get, you get an object which is a proxy and the aspects are applied by interceptors. Every proxy that DynamicProxy creates implement an interface so we can cache the proxy object to that DynamicProxy, proxy interface and from that interface there's a method which allows you to get all the interceptors on that proxy and then we can inspect the interceptor that implements that the aspect that you want to apply is in the collection for this proxy.

Richard Campbell: Interesting. A question over here.

Male Audience: Straight from my conception though is it's so different in applying the aspects that sort of hold all your code by type versus runtime.

Richard Campbell: Krzysztof.

Krzysztof Kozmic: So I guess the other time you can -- well, not every behavior can be applied to that compile time because you don't have all the information at compile time. A very good example of that is logging frameworks which there's no way that, for example, the authors of Rhino Mocks will know what interfaces you're going to mock and what behavior you're going to expect from the interfaces that you mock. Likewise with NHibernate, there's no way that the authors of NHibernate will know what classes you're going to get and therefore they are not able to recreate the proxies for Lazy Loading for your code. So only at runtime where NHibernate inspects your model. It has on the other side, it has the information to actually create the proxies for Lazy Loading your own model and for every application it's obviously different. So I guess if this is your code that you're writing this for, you know that the information you need is there, whereas, if you are like a certified IT provider and then everyone else is going to use your library, they cannot make assumptions about how they code and other people will write...

Carl Franklin: This portion of .NET Rocks! is brought by Telerik JustDecompile. Recent developments in the .NET world have opened up a niche for a free .NET decompiling tool. If you, like so many other developers, have been looking for an alternative .NET decompiler, you'll most certainly welcome to the launch of JustDecompile, a powerful tool which promises to stay free forever. Currently in beta, JustDecompile offers effortless .NET decompiling and assembly browsing, innovative code analysis and navigation, side-by-side assembly loading, auto-updating, and better decompiling accuracy. A product by leading .NET vendor Telerik, JustDecompile has an aggressive release schedule and a roadmap based on community feedback. You can visit the JustDecompile feature suggestion forum to let Telerik know what features you'd like to see added to JustDecompile or vote for one suggested by your peers. The official version launch is expected this summer of 2011. Go to telerik.com/dotnetdecompiling and remember to thank them for supporting .NET Rocks!

Yeah, I guess we covered the testability. There aren't any other barriers to testability that you can think of with AOP?

Philip Laureano: Just composition because as one of the people here ask us, I mean how do you actually know it's composed with a particular aspect. If you have Component A and you want to tie it to a specific aspect, how do you actually test it? It's actually coupled with that particular aspect.

Carl Franklin: Isn't that a matter of the implementation? Like I know that PostSharp uses an attribute so you can look for that attribute.

Philip Laureano: But at some level you stop to be able to test that it's actually occurring in whatever framework you decide to use. I mean, as a natural extension of that you have to make sure that everything is tested including the parts of the code that you add to the rest of the app.

Richard Campbell: Yeah.

Carl Franklin: And in this way you were saying, Don, that that is extremely difficult.

Donald Belcham: It can be, yeah.

Carl Franklin: Yeah.

Donald Belcham: But it's really tough. We found on our current project when you're applying multiple aspects to the same thing, so your composition, you build like a water composition in and we can just go in and flip the switches actually and say, yeah, put logging on or put the work management on and things like that. It's the testing of that, it goes multiple steps and multiple levels of composition there that seems to drive the most difficult for us in our codebase. But it's a simple thing. Yeah, we know we're going to want logging on here because that's easy to test. But the fact that we compose our stuff on the fly and one day we might say, yeah, we want this over here and the next day we might not, that composability is where we're starting. We personally, on my project, are starting to struggle with the testing.

Carl Franklin: What about test aspects? I mean, can you assume that if you test with a test aspect that it's simply maybe write a certain string to a log and then you have to go to the log and find the string. Can you then assume that the rest of the aspects are working?

Philip Laureano: Well, that's actually what the design of my contract does. Back in, I think it was in the 1980s, they came up with this idea of trying my act. I came up with this idea that you could wrap methods with certain assertions. In fact, the Aspect-Oriented Programming does solve that kind of thing because in practice you really wouldn't want to put all these assertions all over the place. Nowadays, it's just impractical when you could just have a separate project with unit testing. Assuming if you wanted to do something like say design my contract, then you would have to use Aspect-Oriented Programming to say which classes and which interfaces would have to be checked before and after the method call to make sure that it's working the way it's supposed to be working.

Carl Franklin: So Don, even though it's tricky to test, you still find that it's worth using it?

Donald Belcham: Oh yeah, absolutely. I wouldn't work a project without it anymore. I don't think.

Carl Franklin: Really.

Donald Belcham: Yeah, we use it very extensively.

Richard Campbell: Well, it's interesting the way you just describe it but these are aspects you may switch in or out.

Donald Belcham: Yeah, depending on what to regulate the -- because they're non-functional requirements essentially that we're implementing in our aspects, it depends on how that part of the world lose inside the bar for our requirements gathering. So the support team comes to play. You know what? We don't really think we're going to care about logging around the events being fired at the UI. Deeper down in the code, yeah, but not up there. So we can pull it off for them and then they come back and we play and say yeah, we're looking at the logs, we needed that. So we started composing in that way and it's really, really easy for us to do that. It's also because we're using convention over configuration for attaching our stuff, it just happens. I mean, some people are scared about that kind of magic. But for our project we're in now, the guys that I brought on to the team, they just start writing code and it's all getting logged. There's no question to ask if you do the logging stuff. It's just assumed that it's working and things like that. So yeah, I wouldn't use it. The increase in velocity that we've seen out of it is spectacular.

Richard Campbell: How much do you think the cleanliness of your code helps things in that? None of this plumbing is visible. I know you told me a little bit about your project, that a bunch of different guys are working on this and there's some turnover there too. So people aren't particular about the people's code.

Donald Belcham: Yeah. The lack of visibility always seems to be the biggest problem we have because of the turnover and training new people. Usually what we find is that they don't need to know about these things all the time. The developer doesn't need to know what's being logged. It shouldn't be the thing that they're thinking about when they're building a new service layer component or a new UI component. It's not the logging you were looking for.

Carl Franklin: It's that what you're getting at? The Jedi mind trick.

Gael Fraiteur: So if you're using the big time framework like PostSharp, well, I guess you have this sensibility inside. You just make sure you can see which aspects are being applied.

Donald Belcham: So we're using runtime and not build-time stuff. We're actually using the stuff Krzysztof works on.

Carl Franklin: Okay.

Donald Belcham: You need Castle Windsor and using interceptors on it. The visibility is simply where are we configuring our aspect to be attached in the FLUID interface for Castle Windsor, whereas, if we were using PostSharp we could see on the class name there would be a little line under the name, you hover over it, it gives you a tool tip saying these aspects are being attached. So, the visibility question disappears very rapidly there. The way we're using it with the interceptors, you have to know to go look for the configuration, where it is, what it means, all that kind of thing.

Carl Franklin: Yeah. So let's make some -- oh, I'm sorry. We have a question right in front.

Male Audience: Yeah. So I have a question for the author. If you, for example, implement INode or by property change using that aspect, could that __________ 24:40?

Gael Fraiteur: Yes, yeah.

Male Audience: And then you inject the IL code into the __________ 24:51.

Gael Fraiteur: Yes, yes.

Male Audience: Will you still be able to debug __________ 24:57. My question is do you maintain the simple style?

Gael Fraiteur: Yes, exactly. Yes. So you can step into the aspect. So if you step into the center, you will get into the aspect and the aspect will call the property intercepted, it will step into the property. If you don't want to step into the aspect because it's becoming boring, then you use the custom match __________ 25:20 for instance and then you -- so you go directly to your code. So you can control that as usually as if it were a normal code.

Male Audience: Okay. But when you say step into the aspect, you mean stepping into IL code.

Gael Fraiteur: You step into the aspect code. Well, you see the code of the aspect. You don't see the MSI code or the SMB code.

Male Audience: Okay, okay.

Donald Belcham: It does as Gael said becomes boring very quickly when you have a lot of aspects and you're debugging and you're stepping through and it's like, "Oh, I'm going into the logging aspect, again." Step into the next class, I'm going into the logging aspect, again. So, that is one of the downfalls when you're debugging and stuff.

Carl Franklin: Okay. We have a question, way in the back. Could you please speak up?

Male Audience: __________ 26:06.

Carl Franklin: Okay.

Male Audience: You mentioned a __________ 26:11. Do you have any more examples of __________ 26:24? Would you consider security __________ 26:26?

Carl Franklin: Yeah. That's a good -- he says, you know, let's give some more examples of things that we can do with aspects. I'd like to hold on that question after one more because that's exactly the question I was going to post, that we take some time to sort of give the pitch for AOP in terms of features, you know, things that you all gleam from it, but we have one more question right on front.

Male Audience: __________ 26:47 some things that help us choose which one?

Carl Franklin: The question is how do we choose between compile time aspects and runtime aspects?

Gael Fraiteur: You attend my session tomorrow.

Carl Franklin: Hey!

Donald Belcham: I think it depends on the tooling you're using on the project. That's how I approach it. So if I'm already using something that supports aspects like an IoC Container, I'll just integrate that rather than introducing a new tool. But that's a personal choice I make with my clients because a lot of the times they're getting introduced to a lot of stuff at one time anyway so throwing another tool on top of the camel might break its back there.

Carl Franklin: Yeah.

Donald Belcham: But yeah, it also depends on what functionality you need. So the runtime type stuff like interceptors through an IoC Container can do some of the things you can do with the compile time tools. So if you have that need, then you're kind of force on your head.

Carl Franklin: Okay.

Gael Fraiteur: I think the key point was you need to use one, whether dynamic or real-time. It's that if you can use the thing on a dynamic runtime, well, do it. It's easier. The tools are more renowned. But the limitations are quite serious. So you can add only aspects between a consumer and a service, okay. So if you need to add aspects for instance to WCF -- no, WPF form or control, you cannot put that behind the proxy. So then you will need time and space. So for a complete discussion, please attend the session tomorrow.

Richard Campbell: Is there a performance implication between build-time versus runtime?

Philip Laureano: There's a definite performance implication. Just to quickly answer that question, the other side of this one is you have to ask yourself how dynamic you need to be. If you go build-time, that assumes that you already know what you've already been doing at the point where you build the application and you have the aspect in place. There are scenarios of course where you're not going to be able to know what you're going to be doing until the program is running. So there are different approaches to doing that. There are different libraries that actually allow you to put hooks in instead of action putting the behavior itself in. So that's something to consider because if you throw build-time, it's okay if you already know what you're going to do. But if you want to go dynamic and if you're not afraid of a little magic, then go dynamic and just go runtime.

Carl Franklin: Reflection, is this is an issue with dynamic stuff? Does AOP usually rely on Reflection? I mean, not always but it can. Can it?

Krzysztof Kozmic: Well, it relies on Reflection first time you request a proxy for a given type because the way DynamicProxy works is you request the proxy for something and DynamicProxy calls a cache of proxy classes that it generates. So it looks into the cache and it looks and says, okay, I don't have a proxy for that, then I know what type I want to proxy, then it goes and using Reflection, and using the Reflection I need it goes and it generates the proxy class. So there's a solid performance hit at the first time you request the proxy for a given type.

Carl Franklin: Okay.

Krzysztof Kozmic: At the second time, there's no Reflection happening because all it does is it looks into the cache, okay, I have the type. Now I can create an instance of like interceptors and give you the optic back.

Carl Franklin: Okay, okay. But it still might be worth it.

Donald Belcham: I've always found the performance hit it is the idiot programmer writing the aspect.

Carl Franklin: That's a very good point.

Donald Belcham: Yeah. It's always the guy writing the aspect like me who thought things they've got to write and not realizing the implications of applying that aspect in certain parts of that code. So we did logging where we want the entire parameter list and all the sub-objects and everything in print, all the values coming in for the collections and things like that.

Richard Campbell: What could go wrong?

Carl Franklin: Yeah.

Donald Belcham: We applied it to our repository layer for NHibernate and had this actually in Lazy Load and we couldn't figure out why we're getting these new his all of a sudden, performance hits, against the database. It was because as soon as it loads it up, Lazy Loaded, we walk through it and it says, oh yeah, I need this object, go to the database and get it. Oh yeah, I need this object. So we just negated Lazy Loading completely.

Carl Franklin: Yeah.

Donald Belcham: Not understanding what the ramifications of putting the aspect on to that part of the application.

Richard Campbell: Yeah. It's simply influencing the way that code is going to behave.

Donald Belcham: Yeah.

Richard Campbell: And as the guy who has lots of performance tuning, I would be concern about how I'm collecting that instrumentation impacting my measurements and having the observer effect.

Carl Franklin: Yeah.

Richard Campbell: I change my results because I'm measuring it.

Donald Belcham: Yeah.

Carl Franklin: Okay. So logging is probably the first feature that we think of when we think of Aspect-Oriented Programming because it's such a no-brainer in tracing. But some other things that -- Gael, we'll start with you because I know a little bit about PostSharp. One of the features that really blew me away was this multiple levels of Undo. Tell us just a little bit about how that works and the kind of trace that you can do with it.

Gael Fraiteur: So one of the ways you can implement Undo/Redo is to record a change of every field. When you change a field of a business object, you record before and after. You put it in and double link list. With Undo, you call -- so you set the field to the old value and you set the field to the new value. That's super easy. It just takes a couple hundreds of lines of code to do this double link list and so on. But the problem is that you have to record all the changes to fields, and with Aspect-Oriented Programming you can make an aspect that has 100 lines of code or less and it will instrument everything and creates a note in this list for every change of a field and it can save thousands of lines of code if you're developing a graphical user interface based on __________ 33:29 model, it can save lives.

Carl Franklin: Yeah.

Richard Campbell: I could see Undo code overwhelming the business code. It's just so much.

Carl Franklin: Yeah, if you're writing it yourself.

Richard Campbell: If you're writing it yourself.

Carl Franklin: Yes. It's just something that I wouldn't even think of writing because it's ridiculous amount of code.

Gael Fraiteur: And then you have Undo/Redo and then you have NotifyPropertyChange. But what went into the post object? You make and undo, but then you have notify that the object has been modified. So it becomes quite tricky to make it manually especially if you figure out in the middle of the development that your past 1,000 implementations have an issue because you did not think about NotifyPropertyChange.

Carl Franklin: Yeah.

Gael Fraiteur: So this is one particular application.

Carl Franklin: What are some of the features that we might not think of when we think of Aspect-Oriented Programming, Krzysztof?

Krzysztof Kozmic: Well, so I guess it will be when you see the same code or a very similar code over and over again that you put in front of your method or at the end of every method or the things that you want to apply around those methods like for example you need work management or exception handling. A good example of that would be for example WPF proxies. So if you have a WPF proxy and if you call a method on the proxy, for example, the communication object may have been closed and you may want to recycle the object. If you know that the call didn't get through then you may want to try to call again. So you can do it very transparently with a DynamicProxy because you can try to issue the call and you know that for example the communication object is closed so you cannot make a call, you can transparently in a way that the call that is using the proxy doesn't know you can recycle the proxy, create a new communication object, open a channel and you can issue the call without the call that makes the call originally knowing it.

Carl Franklin: I'm getting memories of COM+ Enterprise Services. Is there any sort of Aspect-Oriented Programming going on in enterprise services in COM+? Am I just old?

Richard Campbell: You said COM.

Carl Franklin: I said COM.

Richard Campbell: Don't say the C word.

Gael Fraiteur: I think it's very close to the concept of AOP through interceptor because what COM does is it's adding the aspects, or add the interface of the object. So when you require COM objects to require an implement an integer of an interface but you didn't get the object, you got a proxy of the object that was in the COM+ server. So COM+ implemented Aspect-Oriented Programming in some way and we got the same thing in...

Richard Campbell: Some complicated distributed transactions with it essentially. You said it was injecting ahead of the connecting communication, the transactional information as well.

Gael Fraiteur: So this was very close to AOP through proxies because we got -- well, the client code talk to a proxy and then there was serialization and with object working a different process.

Richard Campbell: Don and Krzysztof both mentioned unit of work management. Maybe we need to run down what is that. What does it mean to have a unit of work management?

Donald Belcham: So a unit of work is a pattern that allows you to manage database actions really. So you collect them into many different actions into a list and then you perform them all at once and it gives transaction, type transactioning around the execution of all those steps.

Richard Campbell: Okay.

Donald Belcham: So we use NHibernate on my current project and that's all in NHibernate all handled through the section object and Entity Framework has one. Most of the good ORMs will have some implementation of unit of work. So you need to manage it. You need to say start the unit of work and start accepting stuff to it and then execute unit of work and then whatever happens at that point, whether it fails or succeeds, you do your final step in your transactioning. That code tends to infiltrate all over your application if you don't centralize it somewhere. I remember working way back when I was straight out of school, I code and we pass transaction objects around its parameters to try and solve this problem.

Carl Franklin: Ugh.

Donald Belcham: Yeah. I didn't know what I was doing then. And you want to centralize it into some other place. So having that ability to use an aspect to say everything that's going to be executed up to this point in time has a unit of work available to it and I'll start it up for you and I'll execute the code and then the code executes and then you say at the end, okay, I'm now finish so let me actually run through and flash the unit of work. Go and do whatever work after.

Richard Campbell: It also strikes me that doing that as an aspect means that the guy who know most about NHibernate's behavior and the kind of transactions you end up doing and the rules and/or SLAs around transactional behavior can take responsibility for one shot and everybody who does code from now on that affects NHibernate does the transaction write.

Donald Belcham: Yeah, yeah. Someone else can impose the ruling and everybody else just does their work and the ruling is wrapped around for you.

Richard Campbell: Okay.

Philip Laureano: So in other words, the concern splits so that it's no longer about 10 people. It's just that one guy.

Richard Campbell: This one guy who is responsible for how the transaction suck.

Krzysztof Kozmic: So one guy can mess them all up too.

Philip Laureano: I mean the other angle to this is not just cost-cutting concerns in the code itself. It's also cost-cutting concerns within the people that are working with the code itself.

Gael Fraiteur: This turns out to be the most important benefit of AOP. When I talk with long time customers like two or three years of use, they don't site that we have less lines of code. They cite the benefit that a junior developer comes to the project and he just sees the code that matters or the change we have to do. He doesn't need to have a deep understanding in caching, in NHibernate, any of those.

Carl Franklin: Hey, I just want to give a shout out real quick to our friends at Data Dynamics who make ActiveReports.NET among other awesome things. ActiveReports.NET is great because it allows you to just build your reports with the Easy Editor, embed them right in your application, provide PDF and HTML output, give your end-users a Report Editor, royalty free of course, a great Access report upsizing Wizard and all this for a price that isn't going to break the bank. ActiveReports.NET from Data Dynamics, go check it out now at datadynamics.com.

Donald Belcham: I have a guy on my project who's been there six months, seven months on the project and just a couple of weeks ago finally stumbled across our unit of work implementation. He had no idea what we were doing with it all, and he was right. Everything he wrote code-wise work great, but he never had to know about it. It was a piece of his mind that was just off over there that I don't need to use.

Philip Laureano: Caching aspect?

Donald Belcham: Yeah. Caching aspects are nice.

Philip Laureano: Another one is thread synchronization.

Donald Belcham: Yeah.

Carl Franklin: Thread synchronization.

Richard Campbell: Current field. You're on. Tell me all about it.

Carl Franklin: Frighten me.

Philip Laureano: Well, it's if you were to do it in C#, it's just a lock statement around a method call.

Richard Campbell: Right.

Philip Laureano: Yeah. So instead of explicitly lock it, you would say I want this aspect that before and after the method call it locks and unlocks it.

Carl Franklin: Depending on how you synchronize.

Philip Laureano: Where it becomes hard is when you want to read or write your locks.

Carl Franklin: Yeah.

Philip Laureano: How many people here use the read or write locks?

Carl Franklin: How many people here use locks?

Richard Campbell: Lots of hands with locks, but I only saw one for read or write locks.

Philip Laureano: Why? Because it's so hard to use.

Richard Campbell: Yes.

Philip Laureano: And so there's an attribute so you can say...

Carl Franklin: Well, first of all, a read or write lock is something that is locked for writing but not for reading. So you say this data structure or whatever is locked, nobody can write to it but anybody can read it.

Gael Fraiteur: And so when you can do aspects, you make say a custom attribute. This needs a reader lock because it just reads data and it needs a consistent image of the object and this is a writer lock, and you don't have to play with the read or write lock and then the next version of read or write lock you have to change everything in your code. So just say you're in lockers, use this when it treats and when it is right use this, and when it is on property you can use other lock type.

Carl Franklin: But we really want people to use the new async features of C#.

Gael Fraiteur: Yeah, but it doesn't understand the problem of locking. So we will have more problems with locking, thanks to the async keyboard because now it's so hard to do different threads and then people don't do it.

Carl Franklin: Yeah.

Gael Fraiteur: So with async there will be more threads, more problems with locking.

Carl Franklin: More light works because we got to clean up.

Richard Campbell: Yeah. But you know, interesting point. Light the unit of work that the one guy who really understands thread locking can write the aspect and it's just implemented as an attribute after that.

Carl Franklin: I think it's just one of those things that has to be so finely-tuned for every method, every little piece of code. I don't know. I don't like generalizing threading. Another one that we used to do was the context bound object which was essentially any time this object access, lock it. So it's essentially only accessible by one thread, period, at a time any time you touch it. That's safe, but pretty damn slow too.

Gael Fraiteur: Mine is not safe because if you use it in the user interface, so one thread, the worker thread changes the object, it has a lock, it go to NotifyPropertyChange, WPF will read the object, it has a lock and you have a deadlock.

Carl Franklin: Right.

Gael Fraiteur: So you need read or write locks.

Philip Laureano: For the record, context-bound object is the view of AOP. Nobody uses it I get.

Carl Franklin: Well, that's what I'm saying. It's just threading. It's just a horrible problem.

Donald Belcham: I think you've raised one of the big concerns that people have. Whenever I went to a client and bring up AOP, it's that so much can go wrong if we use this I go saying. It seems to me people think of AOP at a very high level for attaching it to things, but you can attach it very granularly if you use the right tools in the right way as well. So you don't have to attach it to a class. You could just attach an aspect to a property or a method or something like that. So looking at it at a high level freaks people a lot of time. But when you hit it down you say you want only this one little bit I need to do this on, and you work at that level if necessary...

Richard Campbell: Like a library call essentially. When I need this for the plumbing, insert it here.

Carl Franklin: Yeah.

Krzysztof Kozmic: And actually AOP is not something that you necessarily may need a specific library for. I guess the people need to know that they're using AOP but they don't realize it. Because if you are using ASP.NET MVC and you got the action filters, hey, guess what? It's an aspect.

Gael Fraiteur: It's the same with custom behaviors in WCF.

Carl Franklin: Right.

Richard Campbell: Sir.

Male Audience: I take it that you have written projects with AOP once in a while. How come this needs to decouple aspects? Is that common practice or do you walk yourself down to one specific library? Did you understand the...?

Gael Fraiteur: So, let me rephrase that. So the question is would you ever use like for example PostSharp and DynamicProxy in a single project?

Male Audience: No. For example, if you want to switch out PostSharp, how come it needs to not use PostSharp directly but you decouple it through like interfaces and then use that instead so that you're no longer dependent on specific aspects?

Carl Franklin: Is that wise? Because aspects may behave differently.

Donald Belcham: I've never done it. To me it has always been you make a decision on what tool you're using and it's going to be pretty rare for that decision to get overturned. So I guess let it rip.

Carl Franklin: Yup.

Male Audience: But I believe you aspects on like conceptual level so you need to recycle what's called concepts so to speak then you can have those implementations. That's what you're saying.

Gael Fraiteur: Yes.

Male Audience: I'm just asking you if it's common practice to do it or not.

Donald Belcham: It would be very easy to it with interceptor-based aspects because they essentially follow the same pattern. The IO stuff is a lot more structured. You're not as responsible for creating structure like the decorator pattern structures essentially. So it would be a little bit harder. but there's no reason you couldn't.

Philip Laureano: Well, that's if the interfaces for the implementation of those aspects from one library to another are different which they probably are.

Gael Fraiteur: In PostSharp and in similar frameworks, the default way to say that an aspect must be __________ 47:48 a custom attribute in the default is that the custom attribute is the aspect. So, if you have this and you want to switch to another Aspect Framework, you need to create a new class that inherits from a different class. You can dig a little more and say this custom attribute is derived from System.Attribute. It's absolutely decoupled from the Aspect Framework and then you have an assembly wide aspect that would enumerate all the methods that are decorated with __________ 48:21 add the aspect and in this case, your code is fully decoupled from the Aspect Framework. There is only one point of coupling. It is that the Aspect Framework looks at all the attributes. The problem is the feature-to-feature matching of PostSharp with the other real time frameworks, but if you have equivalent features, which is a big if, you can do it this way.

Philip Laureano: The other angle to this one is that some frameworks are specifically designed to leave no trace at all. It's this new thing that would decouple. My friends on Twitter, what they're doing is instead of doing this monolithic framework that we're used to seeing, they would actually implement INotifyPropertyChange but you would have no reference to the framework at all. If you actually look at it, it looks you wrote it yourself. Other things that they do is like, for example, if you're using NHibernate you need everything virtual and you don't want to sit through your hundreds of domain classes and make it all virtual, then what they do is they just go through the assembly and make it virtual and they don't leave a trace. So that's the other angle to this one. It's better off not to leave a trace at all and that's more useful than being reliant on any given project.

Richard Campbell: Question at the back.

Male Audience: Hello. It seems to me that __________ 49:50 sequencing of intercepts because all of the use cases we could see, __________ 49:56. The bigger aspect would be to call __________ 50:08.

Carl Franklin: The question is, is there an issue with the sequencing of aspects and how they are applied that can have differences in behavior? Is that a challenge or is that easy to deal with?

Philip Laureano: It's definitely a challenge because you have to make sure that each aspect is isolated from the other. One of the big things is side effects because if you can introduce an aspect and it does something to clear a state then obviously you're going to have some problems if another aspect comes along and change the same mistake.

Donald Belcham: Gael, in PostSharp, can you specifically order an execution per aspect?

Gael Fraiteur: You must. In PostSharp, if you don't do that PostSharp will see that there is an ambiguity in the order of fabrication and you will get a warning. So the warning is I don't know in which order this aspect will be executed. So it shows one order and the next time you will build the application you will get another. So to get rid of the warning you have to specify the order or you have to say the aspects are commutable. It means the order doesn't matter.

Richard Campbell: Okay. So it's declarative. You have to choose.

Gael Fraiteur: It's declarative and it's implicit ordering. You don't have necessarily to know the other aspects. So you can have aspect libraries that don't know about each other and you can still express dependencies.

Richard Campbell: Sure.

Gael Fraiteur: One of the most difficult points to get to write is composition of aspects and still do code that works if you act a method interception aspect and then a property interception aspect and then an exception aspect, this composition still works. That's the major challenge and that's what I did in PostSharp 2.0 actually.

Carl Franklin: Okay. Yeah?

Male Audience: Do you have any data on the compile time overhead using PostSharp in large enterprise projects?

Carl Franklin: The compile time what?

Male Audience: Overhead.

Carl Franklin: Overhead, any data on the compile time overhead of using PostSharp in large projects.

Gael Fraiteur: So it seems that the minimal cost you have to pay is a little second per project, maybe 800 milliseconds or 700 but take one second to be sure. So, with version 2.0, it's getting to know with version 2.0, think about the build time and now is getting better, it depends on the size of the project. So if you have a huge number of small projects, it's going to cost you a lot. Some teams say that build time is very important because of __________ 53:15. Some teams say that the benefit is so high that if you sit one minute or two minutes, it's not a difference.

Richard Campbell: Right. I'm thinking hours I would care about, minutes not so much, but okay, not much impact. Question here.

Male Audience: __________ 53:31 need to communicate anything from the aspect to the business logic __________ 53:34.

Carl Franklin: The question is how do you handle communication between the aspect and the business logic with accessing objects in the business tier.

Donald Belcham: We do some pretty funky stuff around -- starting working in an instance of an object that our containers contains or has knowledge of both and then it gets back out as a dependency into our objects that need it. So, that object is already visualized once passed as dependency to the next class that needs it and then once that execution path is rolled back up, then we have that object available out of the unit of work for the aspect to be able to say roll back or whatever is necessary at that point in time. The indirection can be very confusing at that level though because you're talking about this and more of this object gets float around out there that gets pulled into different things at different times and it's not really, really clear. Then if you throw -- we're doing Win Forms app and if you throw all the threading in there it starts to get in and making things thread-safe and starts to become even more difficult to visualize what's happening. But we do manage it not through static or singleton or anything. We manage it through a regular class which I guess is being served up as a singleton, not a Windsor in our case, but that's how we're doing it.

Carl Franklin: Yes?

Male Audience: Is it the same with error handling or __________ 55:15?

Carl Franklin: Is it the same with error handling?

Male Audience: Yeah, if an aspect throws an exception.

Donald Belcham: Oh, if an aspect....

Carl Franklin: Oh, if an aspect throws an exception.

Donald Belcham: Those are fun to debug.

Male Audience: Is it going to __________ 55:32?

Donald Belcham: In my experience it depends on what thread it happened in and things like that. Usually, I try and be safe in my aspects and have try-catches all over the place and then inexplicitly handle them the way I want to handle aspect errors and handle that differently than I want my regular code to be handled in error cases, but yeah, if you don't do that, it can be a nightmare. It can really be a nightmare.

Carl Franklin: Ladies and gentlemen, I'm afraid that's all the time we have right now. Could I have one more hand for our panel? 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.