

... class libraries ... aren't...



urazlin 26 posts since

Sep 4, 2002

I cannot agree with this. What about struts configuration file? Isn't it a DSL? What about Maverick config file? It also has a config file format that can be seen as a DSL...

Look at this, for me, this is a program written in some DSL. Meanwhile IDEA supports code completion and error highlights for this language...

-

-

And what is more, Maverick has a very interesting feature - it allows apply an XSL transformation to configuration file. So, you can build your own DSL. Here is a part of my application written with Maverick

-

-

... class libraries ... aren't...

and with xsl I can easily build meta information about my language composed of <edit, <create, <table. If I needed some kind of strict validation, I could use DTD for my configuration files... This my language has nothing in common with maverick config format except it is also XML 😊

It looks for me that for class libraries ability to develop with custom DSL could be easily achieved with this:

- for class library provider: make your library accept XML config
- for application developer: write your application in your own DSL (xml-based)
- for application developer: define transformation of DSL to format of class library config

I'm not a developer of maverick, and I'm not trying to push it anywhere 😊 It appeared that a couple of weeks ago I made a small presentation about our language for building applications with mav and also thought of languages closer to application domain, than to dumb computer instructions 😊



[urazlin](#) 26 posts since

Sep 4, 2002 1. **Re: ... class libraries ... aren't languages ...** Nov 21, 2004 9:54 PM

Anyway, thanks for the article it is very interesting 😊



[Florian Hehlen](#) 295 posts since

Aug 21, 2002 2. **Re: ... class libraries ... aren't languages ...** Nov 22, 2004 11:08 AM

I agree.

... class libraries ... aren't...

A lot of what LOP/MPS wants to formalize is already out there existing in all sorts of forms. But the IntelliJ proposition is to formalize all these approaches into a set methodology.

The example you give with Maverick is valid for Maverick only. LOP/MPS provides a strategy for implementing that same kind of approach where ever needed.

I don't think that Sergey has the pretension of saying that his idea is completely new and novel... I think the approach on it is though. Look at the references for the article... they go back 10 and even 20 years!!!

Florian Hehlen



[Ben Pickering](#) 72 posts since

Aug 21, 2002 3. **Re: ... class libraries ... aren't languages ...** Nov 22, 2004 4:49 PM

For me, one of the main things that makes this interesting is the fact that JetBrains have such a good track record of implementing code editors.

What I mean is, these concepts have been around for years, sure, but an implementation from the guys who brought us IDEA is really exciting.

Even if this isn't a globally adopted new paradigm, as a code-assistance feature for IDEA users, it has tremendous potential. EAP ASAP please! 😊



[Rob Harwood](#) 896 posts since

Sep 10, 2002 4. **Re: ... class libraries ... aren't languages ...** Nov 22, 2004 7:54 PM

You should leave the context in when you quote. Here's what the article says:

====

... class libraries ... aren't...

Domain-specific languages exist today in the form of class libraries, except they aren't languages, have none of the advantages of languages, and have all the limitations of classes and methods. Specifically, classes and methods are immediately tied to a specific runtime behavior which can't be modified or extended, because that behavior is defined by the concepts of class and method. Because they are not languages, class libraries are rarely supported intelligently by the environment (compiler and editor, for example).

====

So, when we say that class libraries aren't languages, what we mean is that the classes and methods defined don't have the flexibility and support that a full language feature would have. They are trying to emulate a real language with a collection of classes and methods.



[Brian Slesinsky](#) 39 posts since

Aug 28, 2002 5. **Re: ... class libraries ... aren't languages ...** Nov 23, 2004 8:19 AM

Specifically,
classes and methods are immediately tied to a
specific runtime behavior which can't be modified or
extended, because that behavior is defined by the
concepts of class and method.

I don't understand this. Interfaces don't specify **any** behavior, and classes can certainly be extendable if they're designed that way.

Because they are not

languages, class libraries are rarely supported
intelligently by the environment (compiler and
editor, for example).

... class libraries ... aren't...

It's true that class libraries could be better supported, and I hope JetBrains works on this. But I don't think it should be necessary to turn a library into a language to get better IDE support for it.



Florian Hehlen 295 posts since

Aug 21, 2002 6. **Re: ... class libraries ... aren't languages ...** Nov 23, 2004 10:22 AM

I think this language vs. library discussion is not the issue here!!!

LOP, IMHO, is not meant to replace things like Struts, Maverick, or any other framework libraries. LOP is not even about frameworks. The fact that you could write new framework type libraries with LOP and that some scriptable frameworks have what resembles LOP is just that... a fact!

Where LOP is very interesting is in an environment like mine: Finance. There is no Apache project that covers the kind of application I help implement. Further-more many financial institutions, because of a long history, and because of a need for secrecy have a very special inhouse business language, business rules, algorithms, etc.

And before anyone say that there are financial libraries out there let me assure you that most financial institutions can't use them, or certainly not straight out-of-the-box; because there are no standards on how to calculate interest rates, do accounting, and billing.

So, I see LOP here as crucial to help formalize the business language that a company uses into a programming language:

-It formalize the real language used by analysts into a programming language.

-It can provide over-site by a wider group of people because code should become inherently more readable for Non-techies.

... class libraries ... aren't...

-It will provide a more flexible tool than a traditional libraries for re-use.

-It might even be a great compliment to an existing library framework. Many library frameworks come with more functionality than is needed or wanted. a LOP wrapper around a library could reduce learning curve and miss-use by people's who's primary job is not to understand what this framework does.

A final point... If I may. I do not think that we should think of LOP as a complete replacement for a standard code-base. At least not in it's first few incarnations. It will more than likely be something that can be used for specific parts of an application. And that will probably vary from one project to an other. In my case I see it aa having great potential in translating business specification/algorithms into code. Which is the place where for me there is the most miss-understanding and time spent fixing code.

Florian Hehlen



[Peter Booth](#) 2 posts since

Oct 16, 2002 7. **Finance is a great example** Mar 19, 2005 4:47 AM

The meaty interesting parts of finance are the

domain specific ones. There are dozens perhaps hundreds of mathematical ways to compute the risk of a portfolio of assets - a huge family of competing approaches to modeling markets. Yet so much financial development seems to be the mundane, tedious business of:

read an array of product ids , positions from a file

if the product id is a valid reuters id then assume the product is non-European. If the id is valid for both Canada

and US issue for the same company deal with this. If the

product is a future then multiply the position by its

... class libraries ... aren't...

multiplier (unless its a Nikkei future traded on CBOT

where we want to divide by ...) If the product id ends in an "=" assume the product is a currency an dstrip teh trailing "=", unless we are in New Zealand ...

New Zealand ...

LOP seems to share a lot with the 'Domain Driven Design described' in Evans book except its addressing the issue from the outside not the inside. Now I speculate that less

than 10% of Java development is really OO devlopment and less than 10% of that is doing what Evan's describes. SO if we can't use current technology optimally is teh solution to change the technology. Sergey is clearly a very smart guy. Is LOP something that the other 97% of us will be able to do effectively?

Anything that can help us isolate the domain specific logic (and please don't anyone say XML rules engines) is a great thing.



[Rob Harwood](#) 896 posts since

Sep 10, 2002 8. **Re: ... class libraries ... aren't languages ...** Nov 23, 2004 2:33 PM

>> Specifically,

>> classes and methods are immediately tied to a

>> specific runtime behavior which can't be modified or

>> extended, because that behavior is defined by the

>> concepts of class and method.

I don't understand this. Interfaces don't specify **any** behavior, and classes can certainly be extendable if they're designed that way.

Let's talk specifically about Java, for the purpose of demonstration.

... class libraries ... aren't...

In Java, an interface specifies no behaviour. What if I want it to specify behavior? This would not be a terribly good thing to do in this particular case, but it does show that you are limited to using 'interface' the way it is defined by the Java spec (unless you start doing meta-programming, which is what MPS is all about).

Back to classes and methods. Here are some typical ways to use classes and methods in Java:

```
C obj = new C(some, args);
```

```
LHS = C.staticMethod(some, args);
```

```
LHS = obj.someMethod(some, args);
```

The last one is most common. What does it specify? It starts at class C, looking for 'someMethod' and scanning up the class hierarchy if it can't find it, then calls that method, passing (some, args) in on the stack, and does some book-keeping to update 'this'. That is what it does, and there's nothing you or I can do about it.

What if, as in Smalltalk and Ruby, I don't know ahead of time whether 'someMethod' exists or not? In fact, I might want obj to be able to accept **any** message, and do something dynamically with it. For example, maybe I want obj to be like a dictionary like this:

```
obj = new C();
```

```
obj.name("Joe");
```

```
name = obj.name(); // Returns "Joe"
```

```
address = obj.address(); // Error, address hasn't been added
```

```
obj.address("123 Apple St.");
```

... class libraries ... aren't...

```
address = obj.address(); // OK, returns "123 Apple St."
```

This is not possible in Java, but it is in Smalltalk or Ruby (and probably other languages).

As soon as we use a class or method in Java, we are limited to what classes and methods can do in Java. This is what is meant by "Specifically, classes and methods are immediately tied to a specific runtime behavior which can't be modified or extended, because that behavior is defined by the concepts of class and method."

Any class library in Java is **unable** to use the trick I just showed, even if that trick would make the code more readable, understandable, or easier to work with. Class libraries are not able to introduce new language features (unless metaprogramming is possible in the language, which is what MPS is all about).

Because they are not languages, class libraries are rarely supported intelligently by the environment (compiler and editor, for example).

It's true that class libraries could be better supported, and I hope JetBrains works on this. But I don't think it should be necessary to turn a library into a language to get better IDE support for it.

So, you don't mind waiting 20 years for JB to add support for your home-grown class library to IDEA? Wouldn't you prefer if you could add that support yourself? Since class libraries are just classes and methods, you can only use IDEA's generic class and method support for your class libraries now. If you instead make a DSL using MPS instead of a class library,

... class libraries ... aren't...

your language features become full-fledged citizens in the system and can have specialized support from the editor, compiler, etc.

The vision is that all or most professional programmers will create and use languages as an everyday, ho-hum task, just as creating and using classes today is an everyday task. So, waiting for JetBrains or some other vendor to add support for all these languages is simply not an option.



[Brian Slesinsky](#) 39 posts since

Aug 28, 2002 9. **Re: ... class libraries ... aren't languages ...** Nov 23, 2004 9:03 PM

In Java, an interface specifies no behaviour. What if I want it to specify behavior?

As you admit, that's a bad idea. You need a better example.

What if, as in Smalltalk and Ruby, I don't know ahead of time whether 'someMethod' exists or not? In fact, I might want obj to be able to accept **any** message, and do something dynamically with it.

I've done this in python. It's a neat trick, but things like this can often be emulated using design patterns when they're really necessary, which isn't very often.

So, you don't mind waiting 20 years for JB to add support for your home-grown class library to IDEA?

... class libraries ... aren't...

Wouldn't you prefer if you could add that support yourself?

Well, no. I'm not willing to spend time writing IDEA plugins, so either JetBrains does it or I'll live without. It's not something I really need anyway.

If you instead make a DSL using MPS instead of a class library, your language features become full-fledged citizens in the system and can have specialized support from the editor, compiler, etc.

But this doesn't happen for free. By the time you're finished developing the IntelliJ plugin, I could be done writing the code in a more standard way.

waiting for JetBrains or some other vendor to add support for all these languages is simply not an option.

Don't you think that's a bit of an exaggeration? We still have the option of writing the code in Java, which has worked well so far.



[Rob Harwood](#) 896 posts since

Sep 10, 2002 10. **Re: ... class libraries ... aren't languages ...** Nov 24, 2004 4:46 PM

... class libraries ... aren't...

This seems to me to just be the belief that "I can't imagine any advantages, therefore there are none. Therefore, I don't need it." Does that summarize your opinion? Some people take a more conservative, what-and-see approach, which is fine. Is that what you are basically saying? Again, that's a valid point of view; I just want to know if that is where you're coming from.

If it is *not* your point of view, then the rest of the message is an appeal to your imagination:

Can you really see no advantages? What about the short-cutting boolean operator? Imagine if there was no such thing in Java. Are you really saying that you would be happy with that situation and that if you had the power to change it, you wouldn't take that opportunity? These examples I've given are really the tip of the iceberg, toy examples for demonstration, something I can type in less than 10 minutes. I urge you to try to see the bigger picture. Maybe check out some of the other articles, interviews, and books that are referenced in the LOP article. There's the Intentional Programming video that's fairly interesting, if a little old.

Ok, with that in mind, I'll address your comments:

As you admit, that's a bad idea. You need a better example.

You're missing the point that Java has a fixed semantic that restricts what you can do with it.

I've done this in python. It's a neat trick, but things like this can often be emulated using design patterns when they're really necessary, which isn't very often.

As the short-cutting boolean example from a previous post shows, the design pattern will technically work, but will be verbose, easy to misuse, and have no IDE or compiler support. So, when you *do* need it, you will have to compromise on a lot of things that affect your (and the team's) productivity.

... class libraries ... aren't...

Well, no. I'm not willing to spend time writing IDEA plugins, so either JetBrains does it or I'll live without. It's not something I really need anyway.

What if adding this support was easy? Would that change your opinion? One of the goals of MPS is to make this very easy to add.

But this doesn't happen for free. By the time you're finished developing the IntelliJ plugin, I could be done writing the code in a more standard way.

Some features *will* be free, like finding usages, renaming, code completion, and others. You won't get that free support (except in a generic class-method way) from a Java IDE. The other support can be added as needed, incrementally, and will be easy to add.

Don't you think that's a bit of an exaggeration? We still have the option of writing the code in Java, which has worked well so far.

How well has it worked, really? Why does it take months or years to write relatively conceptually-simple GUI or web applications?



[urazlin](#) 26 posts since

Sep 4, 2002 11. **Re: ... class libraries ... aren't languages ...** Nov 23, 2004 11:55 AM

I'm really sorry if the subject looks like an attempt to misinterpret ideas presented in the article. The article is interesting to read and presents very interesting and ambitious ideas. Somehow the subject is limited by 70 chars and I cut it out. I'm sorry again.

But, actually I do not see a crucial difference between statement with or w/o context. I'm sorry maybe I just don't get it 😊

... class libraries ... aren't...

What I wanted to say, that with maven you develop applications using language defined here (<http://mav.sourceforge.net/maverick-manual.html#N10326>), and using XSL transformation of configuration you can even build ad hoc language tailored for your particular application domain, e.g. finance. Right? Do you try to do the same thing: provide a means to build a DSL and ability to develop applications using this DSL?

With struts you develop applications using language defined here (http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd). AFAIK, this language is difficult to extend because Struts do not provide any solution to this like XSL transform of config. But Struts has another interesting feature: many tools provide various kinds of visualization of programs developed with this language like this one (<http://www.alienfactory.co.uk/strutsgui/images/example-largeScreenShot.gif>, I never used this tool but a picture is kinda awesome). Do you write about this kind of visualization of programs developed with DSL?

I'm really looking forward to see what you could do in a generalized way to build such kind of tools 😊 It would be great if I could model my extensions to Maverick configuration language with your tool and have some kind of better support for my language composed of <table and <edit than I do have now, maybe some kind of visualization... Will it be possible? Am I on track?



Rob Harwood 896 posts since

Sep 10, 2002 12. **Re: ... class libraries ... aren't languages ...** Nov 23, 2004 3:01 PM

But, actually I do not see a crucial difference between statement with or w/o context. I'm sorry maybe I just don't get it 😊

What the quote is saying is that DSLs and class libraries serve the same role, but class libraries don't have all the power that a full-fledged language could have. So, CLs are languages in the sense of the role they play, but not in the sense of the features of a full language. Paraphrased, "Class libraries are sort of like languages, but not really."

... class libraries ... aren't...

E.g. How could you implement a short-cutting boolean operator in Java using classes and methods?

```
boolean b = new And(  
  
    new Atom(){boolean eval(){return false;}},  
  
    new Atom(){boolean eval(){return booleanWithSideEffect();}}).eval();
```

Or, you could use a full-fledged language feature:

```
boolean b = false && booleanWithSideEffect();
```

The first one is ugly, uses extra memory, etc. It also has no editor support. 'And' is simply a class. 'eval' is simply a method. The IDE doesn't know that they have a special meaning; they are just classes and methods to the IDE, just like ArrayList and size().

The second one, on the other hand, **does** have IDE support (at least in IDEA). The IDE can inspect the condition, see that it is always false, and show a hint to the programmer like 'boolean b is always false.' It can provide an action to remove the booleanWithSideEffect() call, since it will never be called. It could supply special refactorings for booleans, such as applying DeMorgan's rules or something. It can show boolean operators in a special highlighting. It could even show boolean operators as circuit diagrams instead of text, so you can visualize the expression (this would require MPS or a similar system). You could **not** get this level of support with mere classes or methods, because all classes and methods are the same in the eyes of the IDE.



[Miles Parker](#) 14 posts since

Jan 20, 2005 13. **One minor point.** Jan 25, 2005 8:19 AM

... class libraries ... aren't...

I have looked at the Java tools and most of them take xml files to build the code. And guess what, I hate to write code in xml. For example, JSP looks really ugly to me. And while Water was a neat concept it is really practically unreadable. So it is very important to me to be able to do readable code ala C or ALGOL style.

I actually haven't seen a generic tool (beyond something like Velocity or JavaCC which does things at a very low level) that accomplishes anything like what the IntelliJ has in mind. I think it sounds really awesome and I can't wait to try it. Exactly what we have been looking for.



[Mitsu Hadeishi](#) 7 posts since

Jun 25, 2005 14. **Re: One minor point.** Jun 26, 2005 2:55 AM

I agree. I have been thinking about meta-programming for many years now; I've even spent some time jotting down ideas for building a metaprogramming system, but I haven't actually gotten around to implementing it (too busy!), and I was hoping someone would someday build something like this. Of course, it would be even better if it were open source, but I've been wanting something like this for so long I'm going to try it even though it is a commercial project.

It's interesting to see parallel thinking here --- most of the concepts in this effort are things I had concluded were necessary for a decent metaprogramming system --- i.e., having a base language, the concept of a target language, translators, various domain-specific languages, and so forth. Of course, initially people are going to use this to build little scripting-like languages and the like, but I actually am convinced this will revolutionize programming in ways far more sweeping than I think most people suspect.

To me, the advent of meta-programming has the potential to have as great an impact on coding as high-level languages had relative to assembly language. I don't make that claim lightly.

The power of meta-programming or let's say "language-oriented programming" is that one can continue to build more and more powerful abstractions on top of other abstractions.

Even more importantly, LOP allows you to write specifications for programs that aren't

... class libraries ... aren't...

as tightly coupled to execution order. I.e., I want to be able to say "I want a spaceship to enter from the right side at this velocity which can bounce off of objects with property A, B, and C, but can go through objects with property D." Building a set of abstractions to allow one to specify variations of that is conceptually straightforward but nearly impossible to accomplish with standard languages. Instead one has to write a whole bunch of code scattered over a variety of parts of the program: initializing the object, linking it to other parts of the program, adding a variety of methods and interfaces, etc.; with LOP one can have a single expression do all of that at once, hiding all those details from the programmer --- yet the resulting program can easily be as fast and small as a hand-coded program.

Meta-programming also enables other things, like aspect-oriented programming, which can do optimizations and correctness checks that go far beyond the type checking we rely on today.

I see a great potential for LOP --- of course, it will take some time to learn to use it well. But I think as we develop good LOP practices and design patterns we'll be on the way to the next revolution in computing.