

# Not everything is Domain Specific

---



[Andrew Eisenberg](#) 4 posts since

May 26, 2005

Hi,

First of all, I think LOP is a great idea in many respects. I believe that in 10-20 years programming will be fundamentally different than it is today, with DSLs sprouting up everywhere and everyone (including my grandmother) programming (and many times without even knowing it). There will still be plenty of jobs for us programmers, although we'll be meta-programmers.

However, I don't agree that every statement should be part of a DSL. The best example that I can see is a Collections DSL. Efficient use of collections are subtle and tricky. A DSL specifically for collections will need to be complicated to account for all these subtleties.

I don't see it being any simpler or more expressive than a general purpose language (like Java). And there would also be the problem of writing super-DSLs that connects the Collection DSL to every other DSL it needs to interact with (potential combinatorial explosion). Some things are best left in a base language that is commonly known and has instant recognition.

Where LOP (or similar methodologies) will take off is in truly domain specific areas like Business or accounting rules, GUI building, mathematical equations, etc. All are areas where the (meta-)programmer is unlikely to be the one best suited to doing the programming.

What I envision happening is something akin to macros in Scheme or lisp. Macros are powerful and can help create tidy little languages that increase expressiveness...if used well.

However, unbridled use of macros can cause many more problems than they solve. Also, macros are always rooted in some sort of general-purpose base language.

Not everything is Domain Specific

However, LOP doesn't offer macros in the same way as lisp. LOP offers *Display Macros* that do the same thing to the display as regular macros do to syntax.

--andrew



Morgan Mörtzell 1 posts since

Aug 3, 2005 1. **Re: Not everything is Domain Specific** Aug 4, 2005 12:15 AM

I think collections is a very good candidate for a DSL.

Take Java for example.

Before J2SE5.0 you had to use enumerator or iterator classes and call methods like `hasMoreElements()` and `getNext()`.

Then they added a **language feature** that completely obliterates the need for those *unnecessary* classes.

Now I simply type

```
for(Type t: my_collection)
```

So here is real life proof that class libraries really benefits from **language features** that are not possible to do with classes.

The fact that some things are unpractical to implement with the collections api in Java does not make it obsolete. I still use it where it is possible, because of the flexibility.

The same case may happen with a collections language. Some things may be unpractical due to sheer size or whatever.

I can see the use of being able to add my own language features.

In Perl you can write.

Not everything is Domain Specific

```
foreach (sort keys %myMap)
```

In Java you have to write something like.

```
List list = new LinkedList(myMap.keySet());  
Collections.sort(list);  
for(Type x: list)
```

There certainly is a lot of language features that would be handy to have with Java collection.

And just think about being able to add your own features as you need them.

There is a scary thing about a whole new language. But think about it. You can write your completely own collections library in Java. But then you will have to forget about using the for loop. You will have to resort to iterator class and stuff like that. A typical case where OOD messes up code that could have been nice and clean with sufficient language support.

And there are hundreds of APIs out there, and each one has to be **learned**. With language features to cover up the object goup behind, and with code completion and refactoring tools that have an understanding of the 'API', then a new 'API' will be really easy to learn.

So if one sees DSLs as modules, like Java APIs are today, but with custom language features to boost the API. Then this is just a better API.

I think what DSLs will do to APIs is similar to what OOD did to functional programming.

OOD added inheritance, overloading, polymorphism into the data structure that simplifies operations on that very data.

A DSL adds keywords and other structures that simplifies the usage of the API.

But I think the real trick will not be to design a specific DSL. The difficulty will be to make ten or a hundred languages interoperate.

Because there will be need to write programs using many languages. To inline elements from many different DSLs into one program and referencing common objects from any place.

That in itself may be a language. A sort of glue language.

Not everything is Domain Specific

--

And then in the very end, there will be so many languages that the programmers must have interpreters when talking to each other. And altavista will add Babel Fish translation for programmers. And God will weep in heaven...



[Andrew Eisenberg](#) 4 posts since

May 26, 2005 2. **Re: Not everything is Domain Specific** Aug 14, 2005 2:38 AM

Your example mentioned this as a way to remove unnecessary classes:

```
for(Type t: my_collection)
```

I find that such a statement only works about %60 (or so) of the time in my code. This is because it doesn't allow me to do something like this:

```
for(Iterator iter = my_collection.iterator(); iter.hasNext(); ) { Type t = (Type) iter.next(); if (t.isBad()) { iter.remove(); // this part is in
```

Of course, this looks ugly, but given Java's current extended for syntax it is impossible to do.

That's my point...using collections are tricky. The DSL could be extended to allow for a removal of an element, but what would that look like? Would it be any better than using standard syntax? If a DSL only works %60 of the time, is it worth using?

--andrew

Not everything is Domain Specific



[Rob Harwood](#) 896 posts since

Sep 10, 2002 3. **Re: Not everything is Domain Specific** Aug 29, 2005 5:50 PM

I think it would be worthwhile to look at Ruby or Smalltalk to see why a collection DSL is a good idea. For example, in Ruby you have blocks which are very flexible for working with collections. See [http://www.rubycentral.com/ref/ref\\_c\\_array.html#delete\\_if](http://www.rubycentral.com/ref/ref_c_array.html#delete_if) for the equivalent of your latter example in Ruby.



[Fernando Tareco](#) 3 posts since

Sep 21, 2005 4. **Re: Not everything is Domain Specific** Sep 21, 2005 3:10 PM

I entirelyly aggre with you when you say that DSLs will simplify the usages of APIs, apart from other advantages.

What I don't see as a real obstacle will be the DSLs interoperability issue: this may not be needed at all. One can simply share concepts between DSLs, we dont need to share DSLs.

I can give you a real life example: we develop mainframe stuff (CICS/COBOL/DB2 transactions, batches, etc) using a DSL oriented for that platform. Some of that stuff need to be published as WebServies in an integration platform. We have a second DSL relating exclusively web services generation and deployment. In this second DSL we refer some transactions defined in the mainframe DSL.

Finally I also dont see as a threat the proliferation of DSLs: those are to be used locally and have a major goal of productivity increase. So the learning curve has to be smaller than if you have to understand the entire system frameworks. The DSL role is to translate the high level concepts, more easilly understandable, to the organization specificities.