

# Re: Poly Couples

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.object/2006-07/msg00175.html>

---

- *From:* "topmind" <[topmind@xxxxxxxxxxxxxxxxxxx](mailto:topmind@xxxxxxxxxxxxxxxxxxx)>
  - *Date:* 8 Jul 2006 12:29:34 -0700
- 

Sasa wrote:

topmind wrote:

Fine. Oop is great for device drivers (like print drivers above), animals, and shapes. But can we pleaaaaaase get something that reflects what real people see in the real workplace? The vast majority of

Um, drivers are pretty real life example, just as any other "non custom business software" example mentioned in this thread (sorry for nitpicking).

In my fairly limited experience, we had following stuff:

1) Parallel versions of essentially same application.  
There were medium to large variations in look&feel (but not in the data being edited on the forms), the business part was the same, but some features were enabled in some versions.

Why did you have parallel versions?

Plus, the pattern of the parallel version differences is NOT tree-shaped in my observation. It is more or less like the "cartesian join of features" example that I gave to R. Martin.

2) Business validation and reporting the invalid data  
Essentially we had 4 levels of business rules and 4 different situations of presenting them, where situations do not map to the levels. I.e. in every possible situation some subset of rules (based on levels combinations and the current context) would be executed over the current state of data.

BTW. it was required that data is not always persisted, that is – when user is on the form, the control contents is not suppose to be stored. However the rules had to be executed over this "in memory" data. The

## Re: Poly Couples

rules also had to be executed over the persisted data.

I am not sure what you mean. Good tools would allow "in memory" copies of schemas and data tables such that I could query them just as if they were in a database. This used to be fairly common in 80's tools. Only ColdFusion supports this now that I know of (and those left over from the 80's, such as FoxPro.)

If you want to claim that OO is the consolation prize for lame database tools, then you may have a point.

3) The data is highly hierarchical, as presented in the UI. Logically, some actions on the higher level can affect some or all parts of the subtree.

What data is hierarchical? I would like to inspect the hierarchy.

4) Some simple DSL exists to enable non programmers, domain specialists, to manage validation rules, dynamics of the hierarchy (when should some elements appear/disappear depending on the user input).

5) There are ca. 700 input dialogs. Some of those appear more than once in different contexts (same form, for editing different data). Some are dynamic in the sense that they can appear zero, once or any number of times, depending on user input.

Why couldn't p/r do the same thing?

There is not OO model such as employees, orders, order items, ... Rather, there is a metamodel which describes the hierarchy. The leafs map to data either entered by user or computed. There is small engine which evaluates the metamodel and performs significant actions (transferring data to database, evaluating rules, showing/hiding UI elements depending on the user input). The "engine" is implemented in OO. It is not complex, but there were couple of occasions where polymorphism fit nicely.

Again, I would like to see an example of this hierarchy. There are not too many real biz things that fit nicely into a hierarchy. If you found a blue moon oncer, then so be it. Every dog has its day, even OO.

## Re: Poly Couples

6) Metamodel is stored in SQL, however, the data is stored in non relational database (essentially flat files).

Why?

It is the mid term goal that the database (for the data) gets changed. Admittedly, the original selection of database was poor, in addition the database dependent statements are all over the code (and not just the engine), and it will be fairly painful to replace it.

I rarely had to replace DB engines. Plus, one could put all the SQL in one place \*without\* OOP if you really want to take that odd approach. Thus, that is not really an OOP-vs-p/r issue.

I'm definitely OO biased (influence of my education), and came to this system when it was already shaped. It is my belief that OO is good for managing such complex stuff, much more appropriate and useful than something procedural.

I keep hearing such claims over and over from OO affectionados, but never see the actual code being better. Is it possible that you never mastered procedural/relational because lack of exposure or good training?

It doesn't amount only to polymorphism, but rather to the whole process of creating small, independent, reusable modules which are easy to understand, maintain and improve. When good designed, the changes usually do not ripple across the entire system,

Isn't that true of anything?

and I feel much less paranoic when editing the code. It is also easier for me to analyze the code. We worked in fairly high paralelised mode, due to strict interfaces and separation of concerns, we managed to work independently and integrate without significant problems.

Schemas can do something similar. Each section only worrys about what it puts or reads onto the DB and not other code modules (other than

## Re: Poly Couples

shared libraries).

I'm willing to give the benefit of the doubt that OO is not optimal, but I still fail to see the arguments coming from the hard OO critics.

Other than that the case for polymorphism (which to me means programming to interface and varying implementations) would be unit testing – you want to test a module and want the test to fail only if there is error in that module and not in any other on which it depends.

I would like to inspect such code.

As for swapping databases, I already mentioned in some other thread the contrived example – imagine writing the application for one bank, now imagine you want to sell it to some other bank. The business process is essentially the same, the existing UI is perfectly fine. However, they insist that you use their existing database which is from a different vendor, and the structure is different than the one you use originally.

Okay, but that is a known up-front need. Plus, it can be done with functions. Generally you would target about 4 vendors (SQL-Server, Oracle, DB-2, Postgre), and thus infrequently have to add new CASE items. You can't target every DB vendor because then you would have to test all jillion vendors. And if you did have that many, why not put the SQL in the database and then dispatch it that way? Special characters would mark variable insertion place-holders in the SQL.

table: SQLstatements

```
-----  
queryID  
DBvendorID  
descript  
SQLtext  
parameterDescript // (perhaps use a param table for this)
```

table: DBvendors

```
-----  
venderID  
venderDescript // example: Oracle, DB2, SQL-Server  
useAltID // vendor to use if statement not available for given  
vender
```

This way you could make some nice screens to bring up, report on, and inspect all the SQL statements for the different vendors. I've seen similar tools to allow DBA's to change SQL without having to fiddle

Re: Poly Couples

## Re: Poly Couples

with code, although it had nothing to do with vendors.

One can add a new DB vendor without compiling a single line of new code.

How's that for abstraction? The mother of all DB-vendor-wrappers is a DB! Out-poly that!

Well, this was somewhat unstructured (sorry), but I hope it does shed some light on my thinking process. Any feedback is appreciated.

Sasa

-T-

.