

# Re: Poly Couples

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.object/2006-07/msg00184.html>

---

- *From:* Sasa <sasa555@xxxxxxxxxx>
  - *Date:* Sun, 09 Jul 2006 09:01:35 +0200
- 

topmind wrote:

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) Paralel 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 parellel versions?

Because it was requested from us :-)  
Essentially – different versions for different clients. For example, there was full blown version with relatively classic UI and there was stripped down version with fancy looking UI. I think total number of versions was >10.

Plus, the pattern of the parellel version differences is NOT

## Re: Poly Couples

tree-shaped in my observation. It is more or less like the "cartesian join of features" example that I gave to R. Martin.

Whatever. It comes down to having some common logic and some variations. For example – the forms have different look & feel. But the communication mechanism with the database remains the same. If you use some mediator between the view and the model where view is abstracted with interfaces, you can easily switch implementations.

### 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 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.)

I never said OO is the only way to resolve it. To me is the most elegant to deal with such differences. By using interfaces, I can remove details from the evaluation logic which really shouldn't care where the data resides. Some simple factory mechanism would deal with that.

BTW. this allows me unit testing of the evaluation logic, because I can provide "test" data source which feeds the evaluation engine with test data.

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

I made no such claim. To me OO is nice general purpose mainstream tool for dealing with complexity and developing agile code.

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

## Re: Poly Couples

subtree.

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

Maybe I wasn't clear enough. The requirements wanted that the data is presented as the hierarchy. Consider something like windows explorer – tree on the left, input dialogs on the right. If you are on the level 2 and edit some stuff, some subset of the subhierarchy could change (entire tree items could appear/disappear, some subcontrols on one dialog in subhierarchy could be disabled or disappear)

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?

What's a p/r?

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

You mean in your experience/opinion, or there is some definite formal proof of it?

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?

I never asked, but everyone agrees it was a bad decision.

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.

In that sense, nothing is – you can do almost anything with any type of system.

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

It is hard to prove. Imagine complex system which took collective effort of couple of man years (only developers time included). How can you prove that it has (or has not) the best possible choice of languages, or that its architecture and design are optimal?

The best I can do is, observe in every day work where does it hurt me to work with the code, and where does it feel simple and intuitive. Nicely designed OO code works for me and my colleagues. That doesn't mean that other languages (such as sql, prolog or lisp) are excluded. But I like to deal with complexity mostly by

## Re: Poly Couples

thinking in objects.

mastered procedural/relational because lack of exposure or good training?

It is possible. My experience is most certainly small, so I give the benefit of the doubt to any claims, including the ones where OO gets completely downgraded.

Could you answer the same question for yourself?

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?

Most probably. My preferred dialect is OO.

and I feel much less paranoid when editing the code. It is also easier for me to analyze the code. We worked in fairly high parallelised 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 worries about what it puts or reads onto the DB and not other code modules (other than 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.

## Re: Poly Couples

I'm no unit testing expert. Keeping that in mind, consider following example:

I1 <- TestableModule -> I2

Since testable module depends on interfaces, by providing simple test implementations, one can test the module in the isolation. Tests will fail only if there is an error in TestableModule or in the test itself. The implementations used in production are not included in the test.

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

Not always, and it can be secondary. Consider that you deal with single client. So you start with your application, code it, test it and deploy it. And then the company tries to sell application to other potential clients. And one client says "This is great, but we require that you use our existing database".

My general point is that you usually don't know where the change will be. By breaking the system into small, independent, cohesive components (which to me means decoupling), one has the system which is easier to change. Sure, it will not cover all possible change scenarios, but IMHO it still helps a lot.

The finer the granule of the module division, the better the chance that modules will get reused – both in the changed version of the same system, as well as in some other system.

Dealing with future changes is not the only motivator. IMHO it is easier to develop and work with small, independent modules, they are easier to develop, integrate, analyze, debug and test.

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)

## Re: Poly Couples

table: DBvendors

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

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 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.

What if structure of the database varies, and you have business calculations (which is common for all systems) in SQL?

To put it more systematically – consider following constraints:

1. You ship the application to multiple clients.
2. The databases systems (both vendors, and the structure of database) are different. The clients don't allow you to use your own database, or to change the structure of the existing one, which you must use.
3. The business logic is mostly common. Some subtle variations can exist.

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

You might be right, wrong, or it comes down to cultural differences.

I somehow don't fancy the idea of having a lot of code snippets stored in the database. I prefer OO with statically typed language for the code (but of course, it depends on the nature of the problem). With appropriate IDE, I can work easily with the code, and have features such as refactoring, debugging and compile time checking. It doesn't resolve all possible problems, but it helps dealing with some of them.

Sasa

.