

## Re: Finally which ORM tool?

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-10/msg01938.html>

---

- *From:* Jon Skeet [C# MVP] <[skeet@xxxxxxxx](mailto:skeet@xxxxxxxx)>
  - *Date:* Sun, 14 Oct 2007 20:00:23 +0100
- 

Frans Bouma [C# MVP] <[perseus.usenetNOSPAM@xxxxxxxx](mailto:perseus.usenetNOSPAM@xxxxxxxx)> wrote:

You won't know which fields have changed unless you query the database whatever you do – sessions don't make the situation any worse here.

If you track changes inside the entity, you don't need to. If you don't have a session-oriented design, you have to track changes elsewhere, so you don't have to put the burden onto the developer to do things for the framework, the framework can decide what to do by itself.

I think we were talking about a different type of change tracking. However, I seem to remember that Hibernate performs bytecode trickery to keep changes within the object itself as well.

I suspect (although I haven't tried it) that using the ORM system on the client would let you effectively have a "fake" session that you can attach the objects to in the same way as using "context objects".

That won't help you. the session on the service doesn't know which fields are changed, which entities are new etc.

To be honest, I can't remember enough of the details to remember how/if Hibernate handles knowing about new entities.

Graph / entity management, one of the services provided by a good o/r mapper, is something often ignored by people who start with o/r mappers but it's one of those things which makes an o/r mapper framework be more than just a save/load layer for data.

Sure – but graph management in terms of uniqueness is more pervasive

## Re: Finally which ORM tool?

when it's part of a session system which is *\*also\** pervasive. IMO, anyway.

So why people still find it necessary to design an o/r mapper around a session is beyond me. Sure, it's more work to do it session-less, but the point of a framework isn't the joy it gives to the author of it, but the joy it gives to the user of it.

Well, given how many frameworks do use it, including organisations which certainly have enough time and money to avoid using sessions gratuitously, I suspect there are advantages which either you don't appreciate the importance of or are unaware of.

Name one advantage of session-oriented design over non-session oriented design for the USER of the framework. There are none. Sure, for the developer of the framework, it's easier, much easier, as every meta-data element and other data element to work with is centrally available and what to do is actually told to you.

You're still sounding *\*very\** black and white, and I simply don't believe that there isn't a single advantage (to the user) of keeping track of things in sessions. I have to say that when I was using Hibernate it seemed a very natural way of working, partly because it was very similar to using a transaction. Keeping a session open for the length of a request just seemed to fit as an easy way of doing something. I never experienced the "hell" of attach/detach, and having opened the session and used that for everything else, I didn't need to learn any "extras" to get uniqueness.

Certainly when I've used Hibernate I haven't found the session stuff to be a pain, and it's been nice to be able to treat it as a context for that request.

2-way databinding in webapps, passing entities across service boundaries... 2 examples where you have to tell the framework what to do while it's the job of the framework to find out what to do.

Ah, databinding – I've never been a fan of that to start with, to be honest. Too much black magic which only works if you happen to want to work in *\*exactly\** the way that was anticipated. At least, that's been my experience of it in every scenario I've seen. It does work pretty well for displaying stuff – but making it 2-way just has *\*so\** many problems. With any luck WPF improves matters somewhat as far as thick

## Re: Finally which ORM tool?

clients are concerned...

Sure if you don't mind to do the extra work, it's not a problem. Also, with hibernate based frameworks, you don't get a lot of entity management anyway. I mean:  
myOrder.Customer = myCustomer;  
doesn't make this true:  
myCustomer.Orders.Contains(myOrder);

Not unless you implement the recommended patterns (which would keep the association consistent). I admit it's a disadvantage to have to do all this manually.

I can't off the top of my head either – but that doesn't mean such features don't exist. I hope you won't take it as an insult if I suggest that you might be somewhat biased towards session-less ORMs, too :)

heh, of course it looks like I'm biased. Though let it be clear that I took deliberately the decision not to create a session oriented framework because of the problems that come with it, because I wanted a framework which would behave as natural as possible (read: as transparent as possible) so with the least amount of work for the user (the developer). And trust me, I've cursed that decision many many times because inside the framework it's often a challenge how to get things done because there's no central info store with all the info for you. :)

Well, as I said earlier: I suspect it's more of a balancing act that you're making it out to be. I'm certainly *\*not\** trying to say that's a deliberate act on your part, just to be clear – but it tends to be easier to see the advantages of your own system and the disadvantages of others rather than the other way round.

I'm sure that if we had a Hibernate expert here (Ayende, for example!) he'd be much more persuasive on the "sessions are okay" side of the fence.

Either the service is stateful (which can be distributed state, potentially – I believe EJB 3 containers provide this option) or you create a new session, reattach any data you need (whether passed by the client, or whatever) and don't need any server state.

Doesn't java have cross-system object awareness? On .NET, a

## Re: Finally which ORM tool?

distributed state is really a red herring, you still have to serialize/deserialize the state which implies a copy.

I don't think Java "natively" has any cross-system object awareness specified – although I know there have been distributed JVMs built, at least as research projects.

Though back to the small space of a webpage which posts back: the state of the page has to be pulled from somewhere. That's of course ok, there are facilities for that. The problem is though, if you track changes in an object outside the entities, you either have to keep that in memory as well (bad) or you have to rely on the developer to tell you what the state of the entities is. (bad too, as it implies babysitting by the developer).

As I say, I thought Hibernate kept track of what had changed within the entity itself too. I don't know about NHibernate though, or how serialization affects this.

If you need server state, you need server state: having a session based or sessionless ORM isn't likely to change that as far as I can see.

Changes in the entity are the concern of the entity, not of some outside object. If it WOULD, the outside object tracking the changes is tied to the entity till the entity dies.

Having a session tracking changes is precisely tying the session to the entity: as soon as you pass the entity to a place where the session isn't available, the changes are lost. As soon as that happens, the developer using the framework has to tell the new instance of the session what happened: is the entity instance a new entity or an existing one. This is significant.

So as this is a pain, people work around this by changing their system design to avoid this. Which in general implies having stateful repositories in memory so changes/sessions aren't lost. Though that's not always possible or not always desired.

Well, all I can say is that the main projects I worked on which used Hibernate never ran into this as an issue. It would take quite a lot of description and conversation between us to figure out exactly \*why\* it wasn't an issue, but please believe me when I say that it wasn't :)

## Re: Finally which ORM tool?

That's also why they have the add/detach hell already looming on the horizon (Linq to sql will throw exceptions in these cases when attaching graphs) and together with deferred execution of linq queries, it will give a lot of problems for users who expect A but run into snag B.

You call it "add/detach hell" but I honestly can't say I ever had any problems when working with Hibernate in this way.

So you never had to tell the session that the entity object you attached was new or not new?

I can't say I remember ever actually needing to attach/detach. I know that it's *\*available\**, but I can't remember needing to use it. If I *\*did\** need to use it, it clearly didn't cause me enough pain to make it memorable.

Deferred execution is a fundamental (and very useful) part of LINQ which people will just have to understand. It's not mind-bogglingly new, either – using the Criteria API in Hibernate does exactly the same kind of thing, building up a query which can be executed whenever you want it to be.

That's not the same thing! A criteria object doesn't contain a session! the object created by the compiler DOES contain a QueryProvider which has to have access to the persistence core to be able to execute the query by itself.

A Criteria object does contain the session (or at least has a reference to it).

See

[http://www.hibernate.org/hib\\_docs/v3/api/org/hibernate/impl/CriteriaImpl.html](http://www.hibernate.org/hib_docs/v3/api/org/hibernate/impl/CriteriaImpl.html) (watch for wrapping)

That's the fundamental difference and that difference is going to cause problems, simply because the linq query object contains a session.

Also, if you pass a variable to the query, the value the variable has

## Re: Finally which ORM tool?

at EXECUTION time is used, not at CREATION time:

That certainly needs to be clearly understood – but I'd say it's useful, too, in cases where you need to do the same query multiple times but with different parameters.

Captured variables always need to be handled with care, and I can see that it could well trip up novices, but I don't think it's unreasonable.

```
string customerID = "CHOPS";

var q = from o in nw.Order
where o.CustomerId = customerID
select o;

// .. some other code
customerID = "BLONP";

foreach(Order o in q)
{
// which orders are read? BLONP's!
}
```

So the query isn't a query definition alone, it's also the resultset.

No, the query itself is a query definition alone. It's only the enumerator you get when you call GetEnumerator() which relates to the result set.

That's a combination of concerns which will cause problems, and IMHO unnecessary. Just because some people thought it would be 'easier' for people to have deferred execution, it's now the main way to execute a query. In fact I can't do:

```
q.Execute();
```

I can do:

```
List<Order> orders = q.ToList();
```

though that will create a duplicate list. However I have to if I want to get the # of orders. If it would be a query specification ALONE, I could have used:

```
mySession.GetCount(q);
```

You can just call Count(). That will execute immediately, and return the count – but without having to fetch all the data.

Re: Finally which ORM tool?

## Re: Finally which ORM tool?

and the results:

```
IList results = mySession.Execute(q);
```

Would that have been such a problem? IMHO not at all.

I don't see that there's a problem anyway, given the ability to call Count().

It's very natural. Also, because in that last example it's not deferred executed, the query is created when the var q statement is executed in code, so the changed variable problem isn't there.

But equally the ability to reuse the query very simply by changing the variables isn't there either. If you want to avoid the meaning of the query being changed, either copy the variable values or just don't change the variables.

Also, if I have a process routine, which has to process 100,000 rows in a complex set of routines, it's natural to page through the rows, process the page read, and persist them in a transaction, to avoid having all data in memory.

Now, how would you do that, with a session object INSIDE the query? You can only do that if you can either use System.Transactions, or if you can place the fetch AND save logic in the same routine. Otherwise you have to pass sessions around to the query creation routine, because the query creation routine REQUIRES the session object, and you need to pass the session object you're using at that moment, otherwise you'll get deadlocks in the db.

I'd usually use the "a context has a current session" idea at that point, where a context might or might not be "the current thread" depending on the situation. (You may need to be careful of thread agility.)

If the query would just be that, a query specification, you wouldn't have had this problem.

Now, call me stupid, but why on earth would anyone with a little bit of knowledge of o/r mapping design the system DELIBERATELY so that deferred execution was there? What problem does it solve? IMHO none.

Re: Finally which ORM tool?

I've always found the deferred execution very natural, both in LINQ and Hibernate – I create a query, do other things if necessary, and then use the query when I'm ready. Why should creating the query execute it immediately?

The equivalent of Execute() *is* either starting to enumerate the results, or calling ToList(), or calling one of the aggregates such as Count(). If you want to execute the query as soon as you've defined it, that's easy enough to do – whereas if you want deferred execution in a situation where defining the query *does* execute it immediately, that's harder.

Now, if you want to argue that it would be nice to be able to defer execution further, allowing it to be unrelated to a session, I can certainly see the benefit of that – commonly used queries could be created at start-up and then just executed against arbitrary sessions. I'd have no problem with that whatsoever – a definite benefit. That's still deferred execution though.

—

Jon Skeet – <skeet@xxxxxxxx>

<http://www.pobox.com/~skeet> Blog: <http://www.msmvps.com/jon.skeet>

If replying to the group, please do not mail me too

.