LocalDTO

Martin Fowler
21 October 2004

If you've been keeping an eye on my fellow ThoughtBloggers you'll know that it seems one of my fowlbots has blown a fuse, the Australian sun obviously frizzles these Swedish models.

Jon's annoyed with Data Transfer Objects, but it's not that DTOs are a bad thing, just like any pattern they are useful in a certain context. Patterns always have two parts: the how and the when. Not just do you need to know how to implement them, you also have to know when to use them and when to leave them alone.

He does have a point - although I only talk about them in the context of remote interfaces, I didn't reiterate their remote context in the write up of the pattern itself - and I do run into people who like to use them in non-remote cases. So here's my chance to make up.

DTOs are called Data Transfer Objects because their whole purpose is to shift data in expensive remote calls. They are part of implementing a coarse grained interface which a remote interface needs for performance. Not just do you not need them in a local context, they are actually harmful both because a coarse-grained API is more difficult to use and because you have to do all the work moving data from your domain or data source layer into the DTOs.

Some people argue for them as part of a Service Layer API because they ensure that service layer clients aren't dependent upon an underlying Domain Model. While that may be handy, I don't think it's worth the cost of all of that data mapping. As my contributor Randy Stafford says in P of EAA "Don't underestimate the cost of [using DTOs].... It's significant, and it's painful - perhaps second only to the cost and pain of object-relational mapping".

Another argument I've heard is using them in case you want to distribute later. This kind of speculative distribution boundary is what I rail against with the FirstLaw. Adding remote boundaries adds

complexity. So I'd echo Randy's advice "start with a locally invocable Service Layer whose method signatures deal in domain objects. Add remotability when you need it (if ever) by putting Remote Facades on your Service Layer or having your Service Layer objects implement remote interfaces."

One case where it is useful to use something like a DTO is when you have a significant mismatch between the model in your presentation layer and the underlying domain model. In this case it makes sense to make presentation specific facade/gateway that maps from the domain model and presents an interface that's convenient for the presentation. It fits in nicely with Presentation Model. I hope to talk about this more in the new volume. This is worth doing, but it's only worth doing for screens that have this mismatch (in this case it isn't extra work, since you'd have to do it in the screen anyway.)

Update: Thibaut Barrère reminded me of another purpose for Local DTOs, communicating between isolates in multi-threaded applications. A good way to handle multi-threading is to partition your application into isolated areas where each area has only one thread running over it. This eliminates the needs for concurrency controls within that area. If you need to communicate between these isolated areas, then use a message queue with DTOs as messages to transfer the information.

ThoughtWorks®