The Proposal Design Pattern

The Proposal design pattern is probably the most significant design pattern you've never heard of. Despite being written up in the November 1997 issue of Object Magazine, this pattern languishes virtually unknown.

The essence of the pattern is that real world "changes of state" are almost always agreed upon by accepting a proposal of that change that has been prepared in advance.

Perhaps the most obvious example is the process of getting married. Before "popping" the actual question, a potential groom will usually purchase a ring. Under some cultural traditions, the groom will also approach the brides' father to request permission before asking the bride herself; in others, this occurs afterwards.

Another example is the process involved in purchasing a house or apartment. First, you need to find a property to purchase - this may involve on-site inspections of several possibilities before a final decision is made. Finance is likely to be required, as few can afford to purchase a property outright. Typically, the finance house or bank will have requirements of their own that must be satisfied - for a formal valuation of the property, for an inspection to ensure the property is "up to code", for health and/or life insurance, and so on.

Defining Proposals

When we develop a business domain for our system, we identify key business concepts and wrap them up as objects in our system. Most often the concepts we capture are the artefacts of the business. Colour Modelling (Peter Coad and Jeff de Luca) includes archetypes for Parties/Places/Things, Roles, Descriptions and Moment/Intervals. With Proposals, we add a new archetype for consideration - an encapsulation of a specific business process, one that takes our object domain from a specific initial state to a given final state.

In most cases, state changes in our domain are simple enough to be encapsulated as simple methods in our development language of choice. Sometimes, however, a state change is too complex to be handled in this way - perhaps because the breadth of impact on the domain model is too large, or perhaps because the business rules involved are complex. When this is the case, creating a specific object to mediate the state change - a proposal - can provide significant value.

Bevan Arps (bevan@nichesoftware.co.nz) is a professional software developer and self confessed geek based in Wellington, New Zealand. With a career that spans analysis to testing, hardware installation to user training, and tech support to technical writing, he is currently a C#/.NET developer working for the Reserve Bank of New Zealand. Bevan's blog can be read online at www.nichesoftware.co.nz.

Document Version	1.0
Last Update	April 2010

Characteristics of Proposals

So what are the characteristics of proposals - what makes them different from other objects that we're used to writing and using in our systems?

Proposals are almost always transient - they exist for a time and are discarded when their purpose is fulfilled. They may also be discarded after a specific period of inactivity. In some systems, proposals are archived for analysis after they have served their original purpose, providing an interesting source of statistics and metrics.

Proposals are often persistent - able to be stored in, and recalled from, some form of persistent storage. This stems from the lifespan of most proposals, typically measured in days or weeks. Losing all incomplete proposals to a system crash, server reboot or system upgrade is seldom acceptable, thus the requirement that proposals be persistable.

Proposals handle invalid (or unacceptable) information easily. Instead of throwing a tantrum (sorry, exception) at the first sight of an invalid data type or an unknown part number, a proposal will simply advise that there is a problem, and require it to be resolved before allowing further progression.

Proposals provide a rich validation experience, well beyond simple type validation, enforcing complex business rules across the entire proposal. They can do this easily because each proposal captures the entire context around a specific state change.

Examples of Proposal use

A point of sale system might use proposals to capture the details of a sale, prior to that sale being completed: All of the items being sold, the method of payment (including any partial payments) and so on.

Some items being purchased, such as liquor, may require signoff by a supervisor after suitable age verification before being sold. Similarly, some forms of payment (such as gift tokens) or authorisation to charge to an account, might require authentication.

If there is an impediment to the sale - no supervisor is available, for example - the details might be shelved within the system, allowing for other sales to be processed without requiring any information already entered to be abandoned. Once the sale has been completed - or abandoned - the proposal itself might be discarded, though in some cases it might be retained for data-mining.

A customer relationship management system might use a proposal to capture an application for VIP status with your business.

First you have the suggestion that a particular customer should be accorded VIP status - perhaps this is an application from the customer themselves, perhaps a recommendation generated by a rules engine harvesting candidates from your sales history, perhaps a nomination from someone in senior management.

Once the process begins, details of the customers purchase and credit histories will be obtained, a recommendation made (perhaps by the system, perhaps by a middle manager) which then requires confirmation. The proposal itself would be retained as a record of the checks made, regardless of the final decision.

When to use Proposals

Generally, you'll only want to use the proposal pattern if you're already using a formal business domain - in the style described by Eric Evans in his book Domain Driven Design - in your application. If you do have such a business domain, the Proposal pattern can provide you with significant benefits.

Proposals can simplify the implementation of validation in your domain. Once you get past simple type validation, domain validation often becomes difficult, largely because the validation routines lack the *context* of validation. Details that are optional in one case might be mandatory in another, and irrelevant to a third. Because proposals encapsulate a specific state change in your domain model, they are rich in context and validation rules can exploit that context to good effect.

If your business domain involves any complex state changes, such as placing or shipping an order, using the Proposal pattern allows you to centralise that state change in one place. Such centralisation allows you easily review the code for the change in one place, and will help with future maintenance.

Proposals provide a natural way to allow *parking* a transaction or operation part way through, allowing it to be picked up later for continuation. This can increase the flexibility of a system, allowing users to suspend activities when necessary due to interruption or lack of information.

Related Patterns

Outside of a formal business domain, the proposal pattern bears a great similarity to the classic **Object Factory** pattern. Both involve a (potentially) complex object which is used to create new objects. The key difference is in the underlying motivation - the entire reason for creating an object factory is to simply object construction, whereas this is a side effect of the proposal pattern.

References

Tibberts, John and Bernstein, Barbara. "Reuse through Proposals." Software Engineering, Object Magazine November 1997: 51-65.