

Introducing the Expert's Library

by
Volker Koster

Expert's Library



Introducing the Expert's Library

by
Volker Koster

Regardless of your part in software projects, I'm sure you've learned this truth: Conducting Software Projects is tough business. The bigger they are, the tougher they are.

Despite the industry's progress in various areas like languages, frameworks and architectural paradigms (object orientation, UML, EJB, SOA etc.), large software projects still seem to be risky business. One of my colleges put it this way: "Our problems never fail to scale to our technological progress".

During my "career" as a Software Developer (had I but listened to my parents and learned something real), complexity most always turned out to be the worst enemy to a project's success. I guess this makes the question of "What impact does this have on complexity?" the benchmark for every project activity, tool, framework, architecture or new hype.

Complexity is what makes software projects such a risky business and it tends to sneak into every project sooner or later. The bigger the project, the more people are involved and the longer the software lives, the more will complexity take its toll.

Complexity comes in many different flavors.

Most obviously, we are facing complex problem domains. These may themselves be intricately interwoven with one another, thus further enhancing the level of complexity. Any solution we develop never works "stand alone" but will more likely be part of some complex process chain.

These and other "modern" requirements have a tendency to force us into employing complex implementations for our solutions. A single concept of a given problem domain may be implemented in hundreds of lines of code, distributed amount countless classes and files. I regularly found myself in situations, where reconciling code with its original intent took much longer than fixing the bug itself.

Abstraction is our preferred way to deal with complexity. Abstraction manifests itself in models. Models let us abstract from irrelevant detail and help us to focus on the essentials of a given problem domain. In a way, they work like sunglasses: the more filtering you apply, the less you see and the less it hurts. But beware, too much filtering will eventually block out some of the essentials too. That makes building adequate models a complex task if its own.

And once you have a good model, you are far from done.

The carefully extracted (modeled) concepts of your problem domain have to be implemented somehow. Here too, things may blow up on you. The nature of implementation is to

deal with detail. This obviously constitutes a natural conflict. Model and implementation are constantly in peril to diverge from one another. Once you submit to this model-implementation gap, you may as well dispose of your model, because you will never be able to reconcile your model concepts from your implementation code. From that point on, code is the only reliable source of information about what goes on within your application and you are thrown back into the most complex layer off all.

This is where you definitely don't want to be.

The **Expert's Library** focuses on dealing with complexity, especially on narrowing the gap between model and implementation.

Instead of any rocket science, you will find some state of the art software development techniques and a lot of painfully gained experience brought to paper in this series. We do not attempt to cover a given subject in a comprehensive or complete manner, but instead take a pick on topics we consider or experienced as being helpful or promising. With these topics however, we indeed go into a lot of detail.

We will start off with the most obvious way of keeping the bond between model and code intact, namely by generating code from models. "Not another code generator, please" I hear you sigh. But stay with me for another second. There is a lot of consent regarding to the fact that it is if not impossible, than at least not practical to generate 100% of an application's code.

But it may be possible to decompose your code into something we will call "The Abstraction" and something we will call "The Configuration" of this abstraction.

We will give examples which will demonstrate that it is indeed possible and absolutely beneficial to generate 100% of the configuration code. This can be done by turning the logic and the rules behind your configuration into something called a Domain Specific Language (DSL) and by building a code generator especially for models specified in that language.

My personal opinion is, that a lot of the SOA hype (like it or not) we experience today, originates in the fact that SOA in its core, very successfully, takes on a similar approach.

Take on a bird's view of a given SOA implementation. You may consider the services as your abstraction and the rules for plugging them together as the configuration of these abstractions.

The services encapsulate your business logic and you most probably don't want to generate these. But the configuration is done on a higher level of abstraction following its own syntax.

From that point of view, I would consider BPEL a Domain Specific Language where the domain is "making services work together".

In most cases, no code is generated from a BPEL file but instead it is interpreted by some BPEL runtime engine. But does this make any real difference?

There surely is more to SOA than just that. But think about it, at the very core of its success you will find a strict separation of abstraction and configuration.

This approach may help your projects too.

The **Expert's Library** will start off with some DSL examples designed using the open source framework "openArchitectureWare (oAW)". Other examples using IDEA's Meta Programming System (MPS) are planned for the future. But you will also find some very practical tips on how to make life easier with some Eclipse best practices. A collection of our favorite links will be constantly extended.

Software projects are and most probably will be tough business. Consider the **Expert's Library** as a pick on some promising weapons to tackle complexity.

We are very interested on what you think. Please feel free to send me your feedback.

About the Author



Volker Koster is an Executive Architect at MT AG, Ratingen, Germany.

He's been on software projects since 1990, mostly in Oracle and Java environments.

You can contact him by e-mail at volker.koster@mt-ag.com.

Your comments are very welcome.

© Copyright 2007, MT AG Ratingen

Balcke-Dürr-Allee 9

40882 Ratingen

Tel. +49 (0) 21 02 309 61-0

Fax +49 (0) 21 02 309 61-10

info@mt-ag.com

www.mt-ag.com