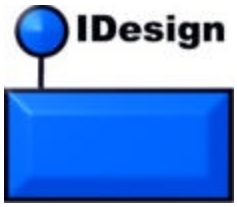

The IDesign Method

Capturing Critical Design Aspects
Version 2.2

Author: Juval Lowy
www.idesign.net

Table of Content

What is the IDesign Method™

IDesign specializes in helping companies design robust, maintainable and scalable enterprise applications based on the .NET platform, and doing so in a fraction of the time it takes normally to design and prototype such systems. To that end, we have crafted a simple yet effective analysis and design technique, called the IDesign Method™. The IDesign Method™ is unmatched in its focus on the required run-time behavior of the application. The IDesign Method™ helps us analyze the requirements, suggest alternatives, discuss tradeoffs, and yield a design that achieves the quality, scalability, security, availability, schedule and throughput goals. In just 3-5 days we typically deliver not only the system architecture (and some 40-60 diagrams) but also a vertical slice implementation of the architecture and even a simple stress tester. WCF is a core part of any service-oriented system based on .NET, and is thus used extensively in the vertical slice. Since developers spend up to 95% of the development time on the run-time aspects of the application (from hosting to transactions to security), having the design done and prototyped in a few days is a huge productivity and quality boost, as well as training the developers on the ins and outs of WCF along with practical application of system features.

We have applied the IDesign Method™ techniques at companies large and small, from the Fortune 100 to the startup. For example, the architecture work performed for HP is now part of an official *Microsoft case study* on .NET architecture and design.

IDesign Method™ Notations

While providing the architecture, we share best practices, guidelines and pitfalls. In particular, we share more than anything the thought process used to approach a design task. But designing a solution is only the first step. Often developers have the most trouble with ways of capturing design decisions beyond mere logical tiers. In particular, how to capture dynamic behavior and aspects such as allocation of services to assemblies, allocation of services to run-time hosting processes, identity management, authorization and authentication boundaries, code-access security policies, transaction boundaries, and logical threads of execution. Formal notations like UML offer no way of representing these important run-time aspects. To address that, the IDesign Method™ relies on simple diagrams where the boundary or the aspect is clearly marked out, using simple symbols, such as a box or a bar. What the box or the bar represent is simply the context of the diagram. This document describes only some of the notations used by the IDesign Method™, and not the actual analysis process leading to them. We teach that in our world-acclaimed WCF Master Class. Some notations such as queued calls, code-access security policies and logical threads are not covered in this document. The purpose of this document is merely to introduce the basic notations used by the IDesign Method™.

IDesign Method™ Notations Example

Consider the classic multi-tier architecture in Figure 1. The manager components encapsulate the use cases or the execution sequence, and the engine components encapsulate some business rules. Clients may call the manager components directly or in some cases use an ASP.NET or WCF web service to wrap them.

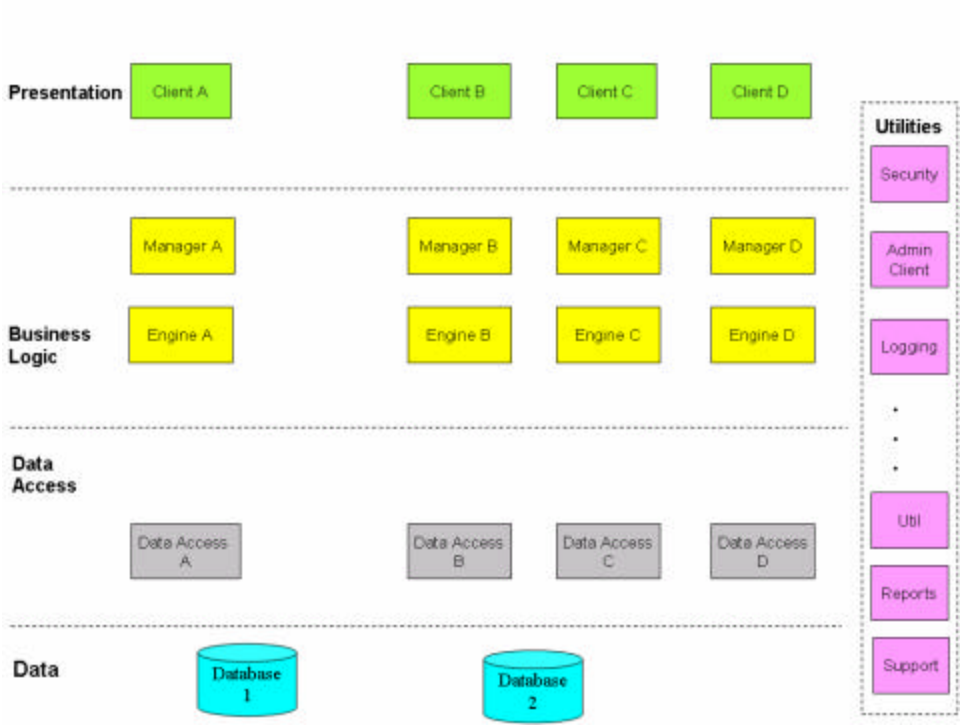

Figure 1: A static architecture view.

However, in a service-oriented approach, it is better to factor such architecture to a set of independent services. This roughly involves refactoring vertical slices of calls through the middle tiers into services. You can use WCF to expose the managers as services directly, or introduce a logical service layer between the client applications and the middle tier. Which option you choose typically depends on how extensively you would use WCF: would you use it only as the gateway into a service, or would you use WCF vertically across all logical components. All things being equal (especially throughput and performance), IDesign recommends the latter, to maximize the internal decoupling that WCF and SOA provide. The refactored architecture is shown in Figure 2.

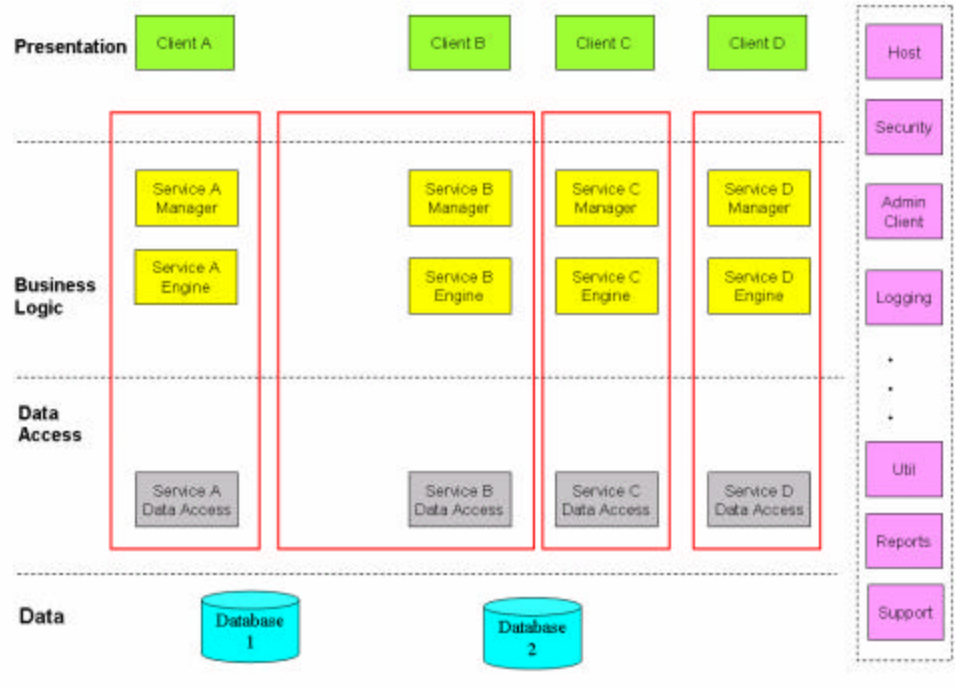

Figure 2: The logical services.

The architecture manages a few services, each with its own client, business logic and data access. These components in turn can be WCF services as well. The architecture follows a closed architecture pattern, where each component or service is allowed to call only components or services in the tier immediately underneath it. In addition, there are separate utilities such as hosting, logging or security services that all components and services can use. You can superimpose on the static view from Figure 2 the call chains of each use case from your used cases, as shown in Figure 3.

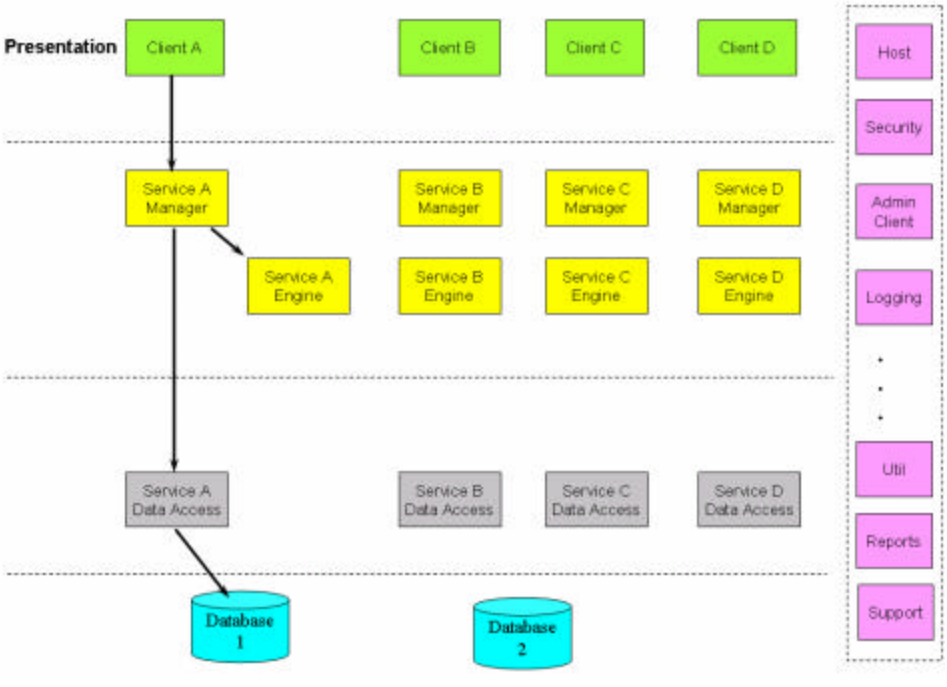

Figure 3: A call chain through the architecture.

Assembly Allocation

After deciding which clients, components and services go to which assembly, capture that in a separate diagram, as shown in Figure 4. In general, client applications reside in application assemblies and everything else in class libraries (you can always expose services a process using a host as later on). When not using the WAS for hosting, you will also need a host application assembly that can host the various services.

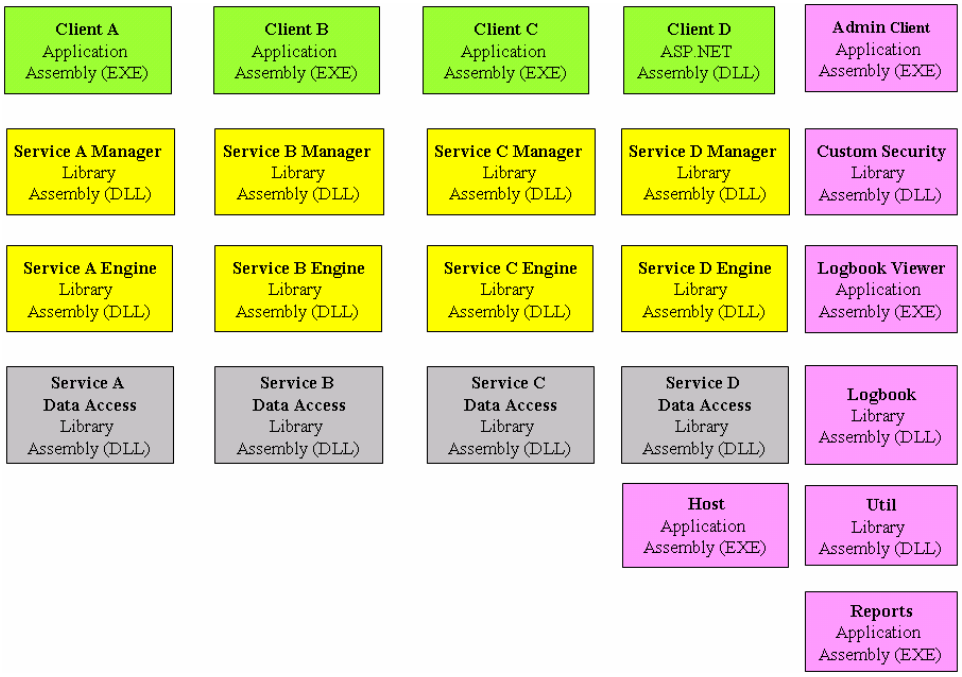

Figure 4: Allocation to assemblies.

Run-Time Processes Allocation

Next, decide on allocation of services to run-time processes based on the need of the application for fault isolation or security isolation. Your WCF services can be either in a separate process from the client (using IIS6, WAS, or self-hosted), or they can be loaded in-proc. Group all assemblies that share the same process and enclose them in a box, as shown in Figure 5. If you are using IIS6 or the WAS for hosting, show those processes as well. For example, in Figure 5, services A, B, and C are self-hosted but service D uses the WAS (it also has a web client, which is irrelevant to the hosting decision).

Figure 5: Allocation to processes.

Identity Management

Having a process boundary enables you to have an identity boundary, meaning, having the client and the service in different processes run under different identities, but you can still have both client and server run under the same identity. After deciding on the identities based on the credentials required to perform the operations as well as whether or not the services impersonate the callers, mark identity boundary using a box, as shown in Figure 6. Remember, the further from the client, the less relevant its identity is.

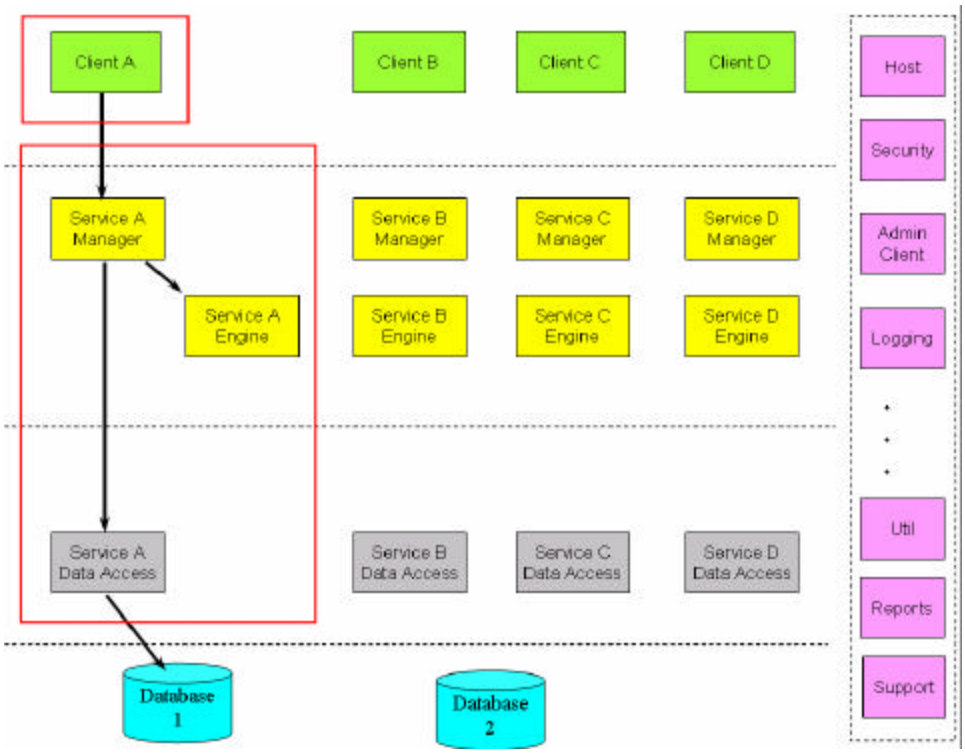

Figure 6: Identity boundaries.

Call Authentication and Authorization

As a design guideline, always perform authentication and authorization at every crossing of service boundaries. Mark authentication using a solid bar, and authorization using a patterned bar as shown in Figures 7 and 8. Note that authorization does not necessarily occur exactly where authentication does. For example, there is no point for an in-proc service to authenticate its intra-process callers, but it can still authorize them. In addition, authorization is meaningless without authentication. Each tier should only authenticate its immediate callers, and implicitly trust its caller to authenticate its up stream caller.

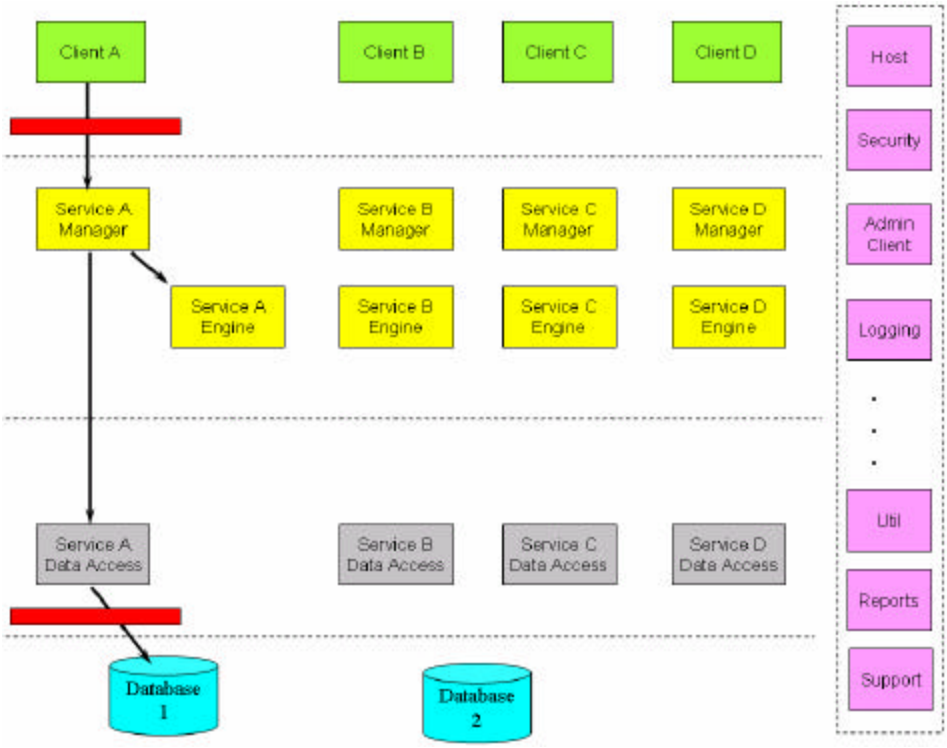

Figure 7: Authentication.

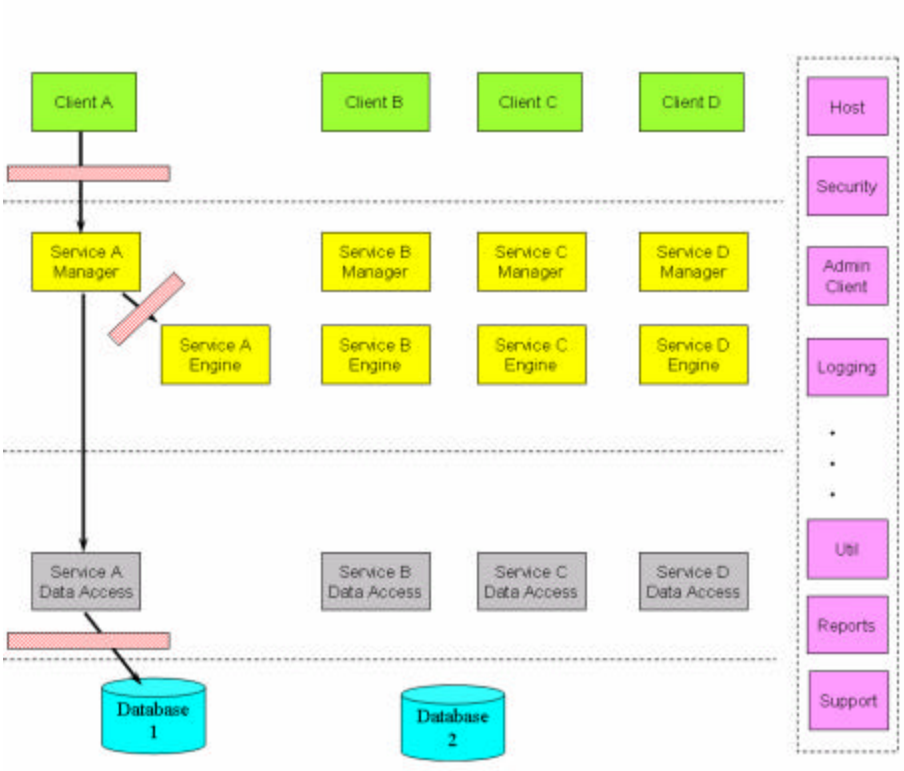

Figure 8: Authorization.

Transactions Flow

After designing your transactions, the transaction roots, participating services and resources, mark the transactions using as box as shown in Figure 8.

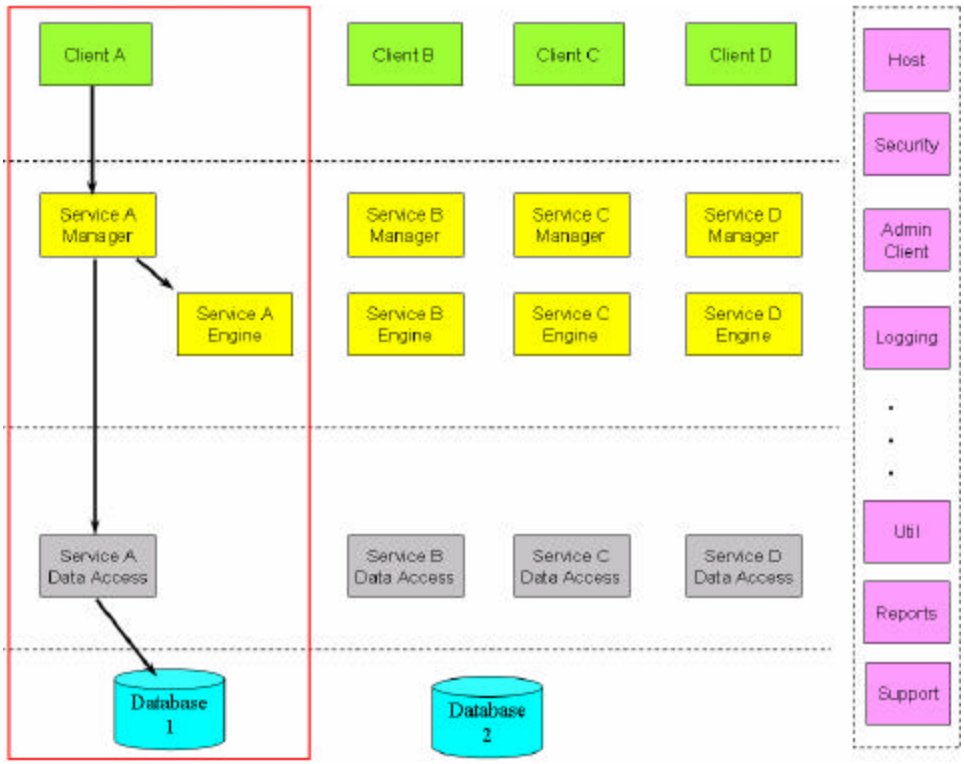

Figure 8: Transactions.

Resources

Programming WCF Services

By Juval Lowy, O'Reilly 2007

Learning WCF

By Michele Leroux Bustamante, O'Reilly 2007

Programming .NET Components 2nd Edition

By Juval Lowy, O'Reilly 2005

6.3 The WCF Master Class

The world's best, most intense WCF training starts by explaining the motivation for service-orientation, and then continues to discuss in depth how to develop service-oriented applications using WCF. You will see how to take advatandge of built-in features such as service hosting, instance management, asynchronous calls, synchronization, reliability, transaction management, disconnected queued calls and security. While the class shows how to use these features, it set the focus on the 'why' and the rationale behind particular design decisions, often shedding light on poorly-documented and understood aspects. You will learn not only WCF programming but also relevant design guidelines, best practices, and pitfalls. The material presented includes IDesign's original techniques and utilities and goes well beyond anything you can find in conventional sources. Don't miss on this unique opportunity to learn WCF from the IDesign architects who have been part of the strategic design effort for WCF from the beginning, and who offer a profound insight on the technology and its applications. Any .NET developer or architect would benefit greatly from the class.

More at *www.idesign.net*

The Advanced .NET 2.0 Master Class

This intense class is world-acclaimed due to its selection of topics, the depth of the discussion, and its focus on best practices, design guidelines, original tools and utilities, pitfalls, tips and tricks.

More at *www.idesign.net*

The IDesign Serviceware Downloads

IDesign serviceware downloads are a set of original techniques, tools utilities and even breakthroughs developed by the IDesign architects. The utilities are largely productivity-enhancing tools, or they compensate for some oversight in the design of .NET or WCF. The demos are also used during our *Master Classes* to demystify technical points, as lab exercises or to answer questions. The classes' attendees find the demos useful not only in class but after it. The demos serve as a starting point for new projects, and as a rich reference and samples source.