

COM+ Design Patterns

Article 1 of 8 in COM+ Design Patterns series

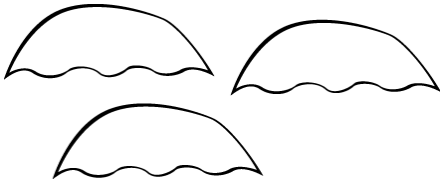
by Randy Charles Morin

In the last five years, design patterns have become extremely important in computer science. The reason they are important is that if you identify distinct common patterns, these patterns can then be re-used. Presented here is another design pattern that explains the evolution of most broad technologies like COM+.

The evolution design pattern is really a meta-pattern, a pattern about patterns. It describes how implementations are often aggregated into designs with a common framework. Once this common framework is established it is then possible to add new functionality across all the implementations.

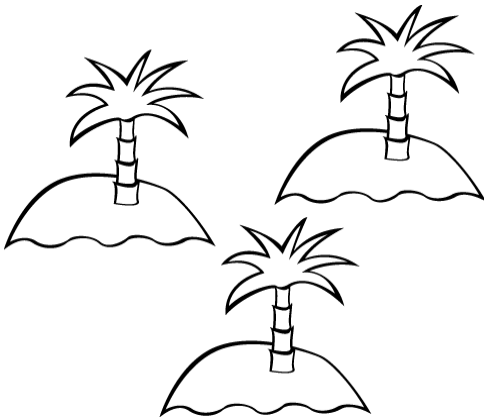
This can best be explained with a couple of pictures. Generally, implementations are originally designed in isolation. Each implementation is like an island. See Figure 1.

Figure 1: Implementations as Islands



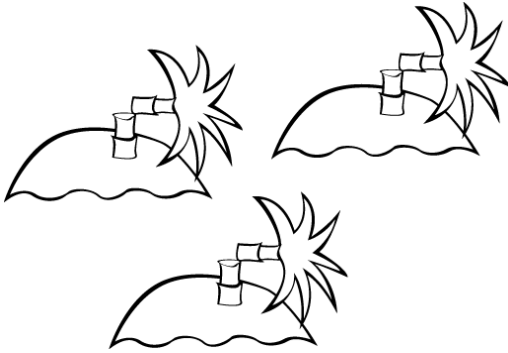
As time passes, each island grows taller and taller as you add new features. See Figure 2.

Figure 2: Taller Islands



Eventually, the group of islands is so complex, being developed in isolation, they begin toppling over under the weight of the complexity. See Figure 3.

Figure 3: Toppled Islands

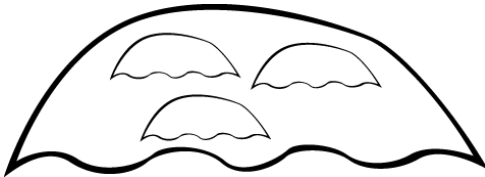


This may seem like a far-fetched example, but in reality, it closely emulates what happens in a software development project. Because each implementation is developed in isolation, it becomes next to impossible for a developer working on one implementation to help on any other implementations. As developers begin turning over (leaving the company for greener (\$) pastures), new developers either within the same company or new hires, have a difficult time learning this extremely proprietary implementation. The implementation topples over.

To the rescue, comes the architect who points out that if the implementation used a common framework, then one developer could easily move from implementation to implementation without having to re-learn all from scratch.

In this new paradigm, the implementations do not exist as islands, as they are no longer implemented in isolation. Now the implementations are dunes on top of a much large island. See Figure 4.

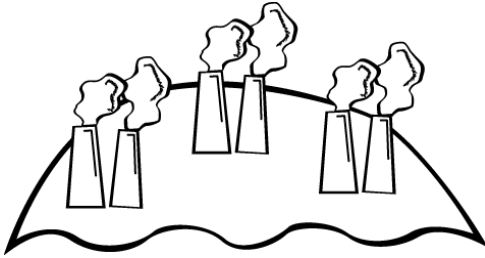
Figure 4: Dunes on a Common Island



This story is usually repeated with each new broad technology and COM+ is one such example. Now if only we could learn from this meta-pattern and not build those islands in the first place.

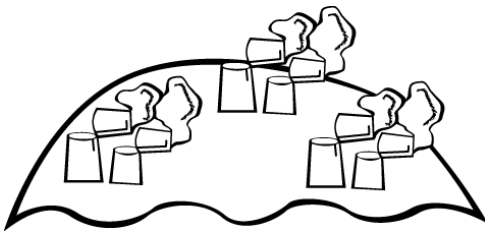
Most applications in the last few years do not really suffer from the island design-flaw. The amount of re-use in computer science has been increasing rapidly and should continue to increase for years to come. Typically, the island design is no longer a problem, but it has been replaced by another flawed design that is nearly as bad. This design flaw is known as smokestack design. See Figure 5.

Figure 5: Smokestacks



Now that you have got everybody on your team re-using one or two common components, you no longer have that island look. Your implementations look now like smoke stacks. This is somewhat better, but still the same problems arise, only to a lesser degree. Although your basic building blocks are now re-used, the next layer of building blocks is developed in isolation. Eventually, the stacks topple. See figure 6.

Figure 6: Smokestacks



So, back to the drawing board again and rationalize more of your design. You'll likely find that components like parsers, string classes, database pooling, logging, and threading are implemented more than once across your array of servers. Keep pushing and as you turned islands into smokestacks, you will turn those smokestacks into dunes.

This story is exactly what has transpired in the world of COM+ in the last ten years. All of Microsoft's object technologies were unique implementations that were built in isolation in order to meet those time-to-market goals. Word and Excel were two such implementations. Eventually, Microsoft created DDE so that these implementations and others could exchange data (from islands to smoke stacks). This designed framework was re-designed into OLE (bigger islands and smaller smokestacks) and again into COM (continents and dunes). With COM, Microsoft finally had that common framework that would enable object-oriented development on their Windows platform of operating systems.

Now that Microsoft had a common framework for object-oriented programming, it could then extend this framework by adding increasingly more features. They added DCOM and MTS and now COM+ to this framework. In the future, they will add SOAP and AppCenter to this framework.

DCOM added the ability to distribute these objects across computer boundaries. MTS added the transaction design pattern to COM. MSMQ brought the queue pattern under the COM umbrella. COM+ brought many more design patterns under the COM umbrella. What are those design patterns that are enabled in COM+?

- Atomic-Unit-of-Work,
- Consumer-Producer,
- De-Coupled Events
- Late Activation
- Object Pooling
- Publish and Subscribe
- Stateless Objects
- Object (and Call) Contexts
- Role-Based Security
- Centralized Management

This isn't a complete list either. Many of the new features added to the COM+ model satisfy two and three distinct design patterns.

Microsoft also wanted to introduce two other design patterns in COM+, but decided in the end to postpone or cancel the release of these features.

In-Memory Databases are becoming an important design pattern in this age where web servers are being called upon to services millions of users. In order to capture this design in the COM+ model, they temporarily introduced a beta-product called In-Memory Databases (IMDB). Microsoft has abandoned this technology and it is not known if they are going to make any further attempts with this type of technology.

The decision to abandon this technology was because IMDB did not meet most of the requirements that customers expected in a database server. In particular, Microsoft expressed that the lack of support for query processing and stored procedures was the reason IMDB was cancelled.

Another pattern originally scheduled for release with COM+ but delayed, is load balancing. Microsoft delayed the release of the load-balancing feature in COM+ a few months later when they released Application Center 2000.

Application Center

Over the last couple of year, I've been programming with Visibroker flavored CORBA (Common Object Request Broker Architecture). This product has a really nice add-in called AppCenter that allows the operator to easily start and stop Visibroker-based servers in a management console that is very similar to Microsoft's Management Console.

Inprise's AppCenter Management Console is very impressive. You can start applications, create dependencies between applications, graphically display command-line arguments, create timers that start applications, view the standard out and error of each application, view the events generated by each application and so on.

Microsoft's Application Center (<http://www.microsoft.com/applicationcenter>) is essentially a borrowing of name and technology from their biggest competitor, Visibroker. The Visibroker line of products has an agent (termed smart agent) that enables static load balancing of Visibroker objects. The smart agents are smart location services, in that they operate on top of the Visibroker location service. They perform load balancing by maintaining lists of available objects and cycling through the list in order to statically load balance the objects.

Microsoft's Application Center will also perform load balancing using a component called the Component Load Balancing Server. The COM Load Balancing Service (COMLBSVC) will maintain a list in memory of load-balanced servers. The list will be sorted by the COMLBSVC using the load and rank of the server.

The smart agent in Visibroker used static load balancing, that's because they did not attempt to load balancing except by traversing the list of available objects. COMLBSVC will perform an enhanced type of load balancing known as dynamic load balancing. It is said to be dynamic, because the service does not just traverse a static list of servers and send an equal amount of calls to each server. The list of servers in Microsoft's Application Center is arranged by load and rank, thus calls are better distributed to those servers that have less load and rank higher.

Another feature of Visibroker AppCenter that will also be available in Microsoft's Application Center is centralized management of servers. This will be done from the Components Services Explorer by simply adding servers to an application cluster. Assuming of course that the applications are installed on each machine in the application cluster, the COM+ environment will automatically activate the applications across all the servers in the application cluster.

SOAP

Microsoft is currently trying to encourage use of a new transport technology into their COM+ framework. This object-oriented transport is called Simple Object Access Protocol (SOAP). Don Box of DevelopMentor, a group of individuals from Microsoft and Dave Winer of Userland Software, created the standard for this technology. The standard can be viewed on the web at <http://www.w3.org/TR/SOAP/>.

The immediate reaction of the industry was that SOAP was going to replace DCOM as Microsoft's object-oriented transport. This may someday be true, but currently the intent seems solely to make the SOAP transport available for web traffic. COM+ application can still use DCOM over HTTP, TCP/IP, UDP/IP, IPX, SPX and others.

The advantage of the SOAP transport is that it combines existing standards into a superset standard. Because the existing standards for XML and HTTP are well known and widely used, it should prove trivial for developers to adopt this new standard.

The most popular COM+ transport is DCOM over TCP/IP. This transport is very much proprietary except that it runs over TCP/IP, the Internet transport standard. The DCOM protocol uses a large range of Internet ports and does not inter-operate very well with well-secured firewalls.

SOAP is targeted at HTTP and this standard requires only one port to be opened, usually port 80. SOAP can run on any number of transport, but the bulk of efforts have SOAP running as an RPC over HTTP. It is therefore extremely friendly to firewall operators who likely already opened port 80 for general web traffic.

Additionally, the data format of the SOAP protocol is extensible markup language (XML), a well-known Internet standard. This standard is very close to the HTML standard in that both were derived from a common standard, Standard Generalized Markup Language (SGML). It should prove trivial for developers familiar with HTML to begin developing in XML. For more on SOAP, visit <http://msdn.microsoft.com/soap> on the web.

The Far Future

Application Center and SOAP are the near future of COM+, but you might also be interested in knowing the far future of COM+. What about DCOM to IIOP interoperability? Actually, such technologies are almost here. Very light forms of DCOM-IIOP inter-operability are already being used. SOAP is one such effort.

Microsoft is also still working on a complete dot-NET framework. This framework when complete will promise to offer a more developer friendly framework. Some of the features of the new dot-NET include a new language C# that is targeted towards replacing client-side Java on the Microsoft platform. I completely disagree with the C# initiative and believe that it is doomed and that Java will flourish despite Microsoft's attempt to monopolize client-side Internet components.

This is the first article in a series of eight on COM+ Design Patterns. The next seven articles discuss the individual design patterns provided in COM+. The second and next article in the series is on COM+ application design patterns.

About the Author

Randy Charles Morin is the Lead Architect of SportMarkets Development from Toronto, Ontario, Canada and lives with his wife and two kids in Brampton, Ontario. He is the author of the www.kbcafe.com website, author of Wiley's Programming Windows Services book and co-author of many other programming books. Bruce Duffus (technical) and Jacqueline Morin (artwork) helped Randy in preparing this article.

References

- SOAP <http://msdn.microsoft.com/soap>
<http://www.w3.org/TR/SOAP/>
- Application Center <http://www.microsoft.com/applicationcenter>
- Design Patterns by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, published by Addison-Wesley Professional Computing