1.1 Web Service Interface Pattern

1.1.1 Name and Source

Service Facade
Page 342-350 in the book " .NET Patterns: Architecture, Design, and Process " [NET 03]

1.1.2 Also Known As

-

1.1.3 Type

1.1.4 Intent

To provide an interface that can be used from the client for invoking methods of a web service proxy instead of being dependent on a particular concrete proxy class generated from WSDL.

1.1.5 Problem

When a web service proxy class is generated from a WSDL file, the generation tool maybe does not also generate an interface for the target programming language. Such an interface would be usable for invoking different web service implementation classes that implement the abstract part (i.e. not the “service” element) of the same WSDL interface. Your generation tool maybe instead generates some “setURL” method for the proxy class that then would make it reusable for other web service implementations. In that case, this pattern is not as much needed because then the client could simply use one concrete proxy class for different web service implementations instead of only be referring to an interface. However, some generation tools may only generate proxy classes with hardcoded invocation URL’s without any possibility to change the URL without manually changing the generated code to provide a set method.

How can a web service proxy interface be provided, when the WSDL generation tool does not support it, if the application will deal with web service implementations from different providers, for example as described in the “Service Factory Pattern” ?

1.1.6 Forces

1.1.7 Solution

Consider a WSDL file “abstract.wsdl” that describes the abstract parts of a WSDL but does not include the “service” element that describes the URL for an implementing web service.

A web service provider “A” may provide a WSDL file “a.wsdl” that includes “abstract.wsdl” but also adds the “service” element with the URL for their web service.

Another web service provider “B” may provide a WSDL file “b.wsdl” that includes “abstract.wsdl” but also adds the “service” element with the URL for their web service.

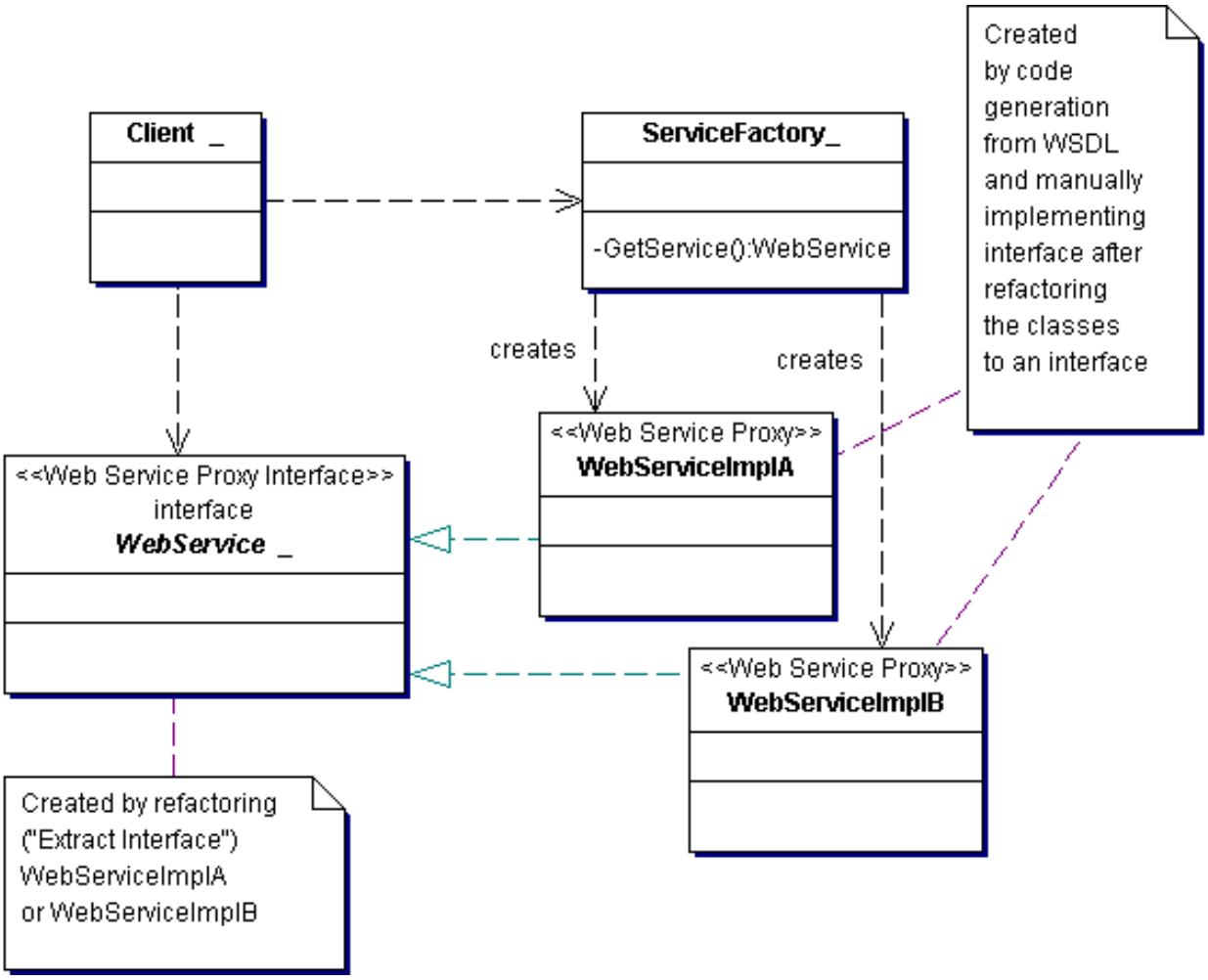
A web service client may then use a WSDL generation tool that generates one WebServiceImplAproxy from the “a.wsdl” file and one WebServiceImplBproxy from the “b.wsdl” file. Both of these generated classes now include hardcoded URL’s.

The client now may want to use the Service Factory pattern too choose one of these two implementations. Note that a Service Factory does not necessarily have to select an implementation by communicating with an UDDI server, but could instead hardcode some kind of algorithm that chooses one of these two implementation classes.

The client that will use a web service proxy would prefer to not have any dependency to the two concrete proxy objects, but rather only invoke methods of an interface, i.e. use polymorphism.

The solution, if the generation tool does not generate any interface, is to create and implement the interface manually. Use the refactoring technique “Extract Interface” [FOWLER 99] to extract the interface “WebService” from either of the generated “WebServiceImplAproxy” or “WebServiceImplBproxy”. Then both of these classes need a small change to implement the interface, e.g. in java you would have to add “*implements WebService*” to the class signatures, while you would have to add “*: WebService*” in C# (or “*: IWebService*” if you follow the .NET convention to use “*I*” as a prefix for interfaces).

In the .NET-focused book where this pattern was found, the author discusses shared assemblies and seems to assume a .NET only environment in the section for this pattern. The diagram in the book is more complicated than here, and the description of the pattern in this document ignores shared .NET assemblies totally, and only covers the aspect of the pattern that it in some environments may be necessary to manually create interfaces if the WSDL code generation tool does not do it for you.



This pattern describes how to create a common interface for the proxy classes that are created from WSDL files from different web service providers when the only difference in the WSDL files is the “service” element with URL’s to the web services. The pattern only describes how to create the structure in the class diagram, but for a sequence diagram that illustrates the usage of the interface refer to the “Service Factory pattern”.

1.1.8 Consequences

The pattern is an option to using one generic proxy object that can be used for changing the web service URL, if such a generic proxy can be generated.

1.1.9 Related patterns

This pattern can be used instead of using a “setURL/setInvocationURL” method in a generic web service proxy object as is being mentioned in the “Service Directory pattern” and the “Service Factory pattern”.

This “Web Service Interface” pattern has almost the same name as the “Service Interface” [ESP 03] which is a pattern that describes how to provide a service interface to application logic that can be reused from different contexts. The implementation of the service interface only takes care of the network communication code but otherwise delegates to the application logic.