1.1 Chained Service Factory Pattern

1.1.1 Name and Source

Chained Service Factory
Page 143-152 in the book ".NET Patterns: Architecture, Design, and Process" [NET 03]

1.1.2 Also Known As

Note that this pattern have a similar name to the "Service Factory" pattern, but that pattern is very different as described in the related patterns section.

1.1.3 Type

1.1.4 Intent

To provide a general and loosely coupled Web Service interface that will not have to be modified in the future.

1.1.5 Problem

If you are using strongly typed RPC methods, then all clients will have to be updated if something is changed at the web service. For example, if you are a service provider that want to add another parameter to a web service method then you will probably not be able to simply add an extra method, since WSDL 2.0 removes the operation overloading that was supported by WSDL 1.1. You will instead have to change the existing method but that would force all clients to update their implementations, even though they maybe are not interested in using the new parameter that was added to your method.

How can you avoid having to change all clients when you want to make some change to your web services ?

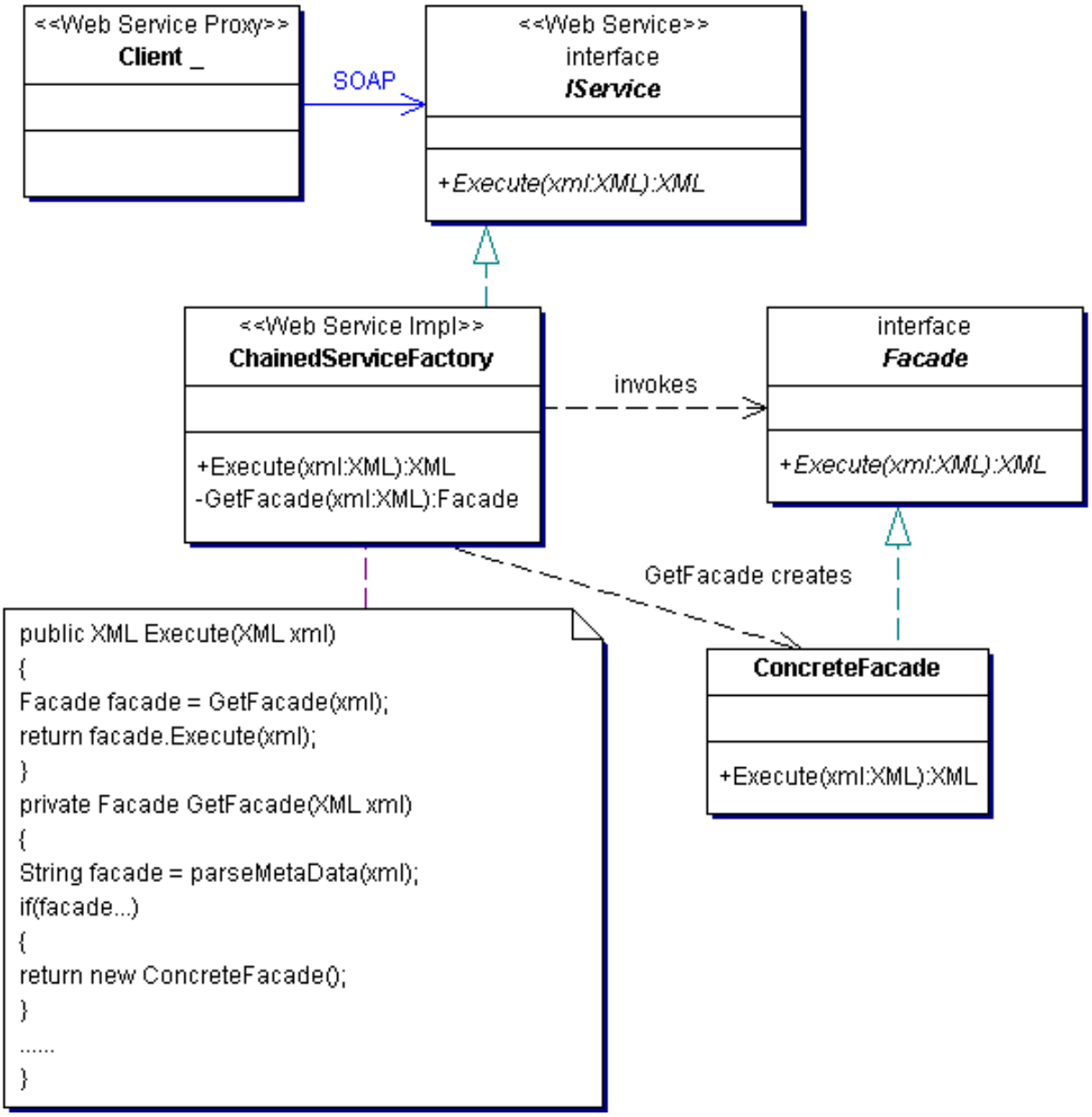
1.1.6 Forces

1.1.7 Solution

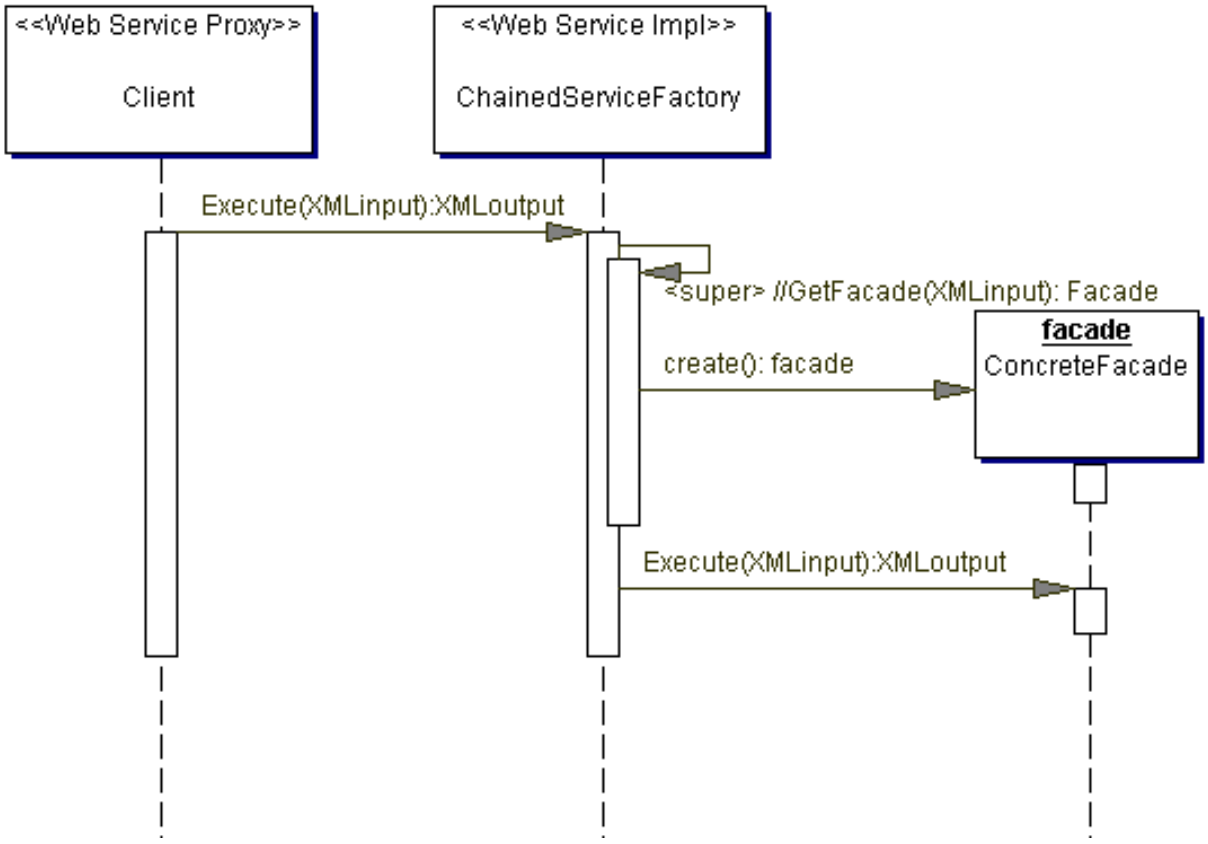
Instead of using RPC methods with strongly typed parameters you can expose a web service with a generic datatype, e.g. XML, which includes not only the parameter values, but also metadata about what actual service methods to invoke.

The general Web Service method is both a “factory” and “delegator” i.e. it first uses metadata to choose the right Façade to instantiate, and then it can use other metadata as parameters when the Façade method is invoked.

The book where this pattern was found is very .NET focused and uses a .NET Dataset as the generic datatype while claiming that “Using DataSets provides the architecture with the most flexible alternative”. I do not believe in that statement and assume that Sun Microsystems also would disagree. Therefore XML is used here instead of the Microsoft DataSet .



The “GetFacade” method in the web service class parses the XML to determine which Façade to instantiate based on the metadata in the XML, similar to the method “PacketTranslator.GetService” at page 150 in the source book that gets the information from a DataSet. The switch-statements in the “Execute” web service method at page 149-150 may instead be put into a “GetFacade” method as in the diagram above, to become a parameterized Factory Method [GoF 95].



[the occurrence of "<super>" in the diagram is not intentional and will be removed in the final version]

Client – A web service proxy object that sends a SOAP message with general XML rather than XML content that will be translated to a strongly typed RPC message.

ChainedServiceFactory – A general web service implementation that will parse the metadata in the XML to determine which actual class that the client wants to invoke. In other words, the ServiceFactory chooses which ConreteFaçade to delegate the message to.

Façade – A façade object that will continue to parse the XML content to determine more details about the requested invocation, i.e. which method to invoke and the values of the parameters.

1.1.8 Consequences

The pattern can support a set of completely different services by using the Chained Service Factory as a central controller that also would make it easy to handle http monitoring and measuring of the system load in one place.

The pattern will be more complicated to use for the clients compared to using strongly typed RPC objects generated from WSDL, since they will have to construct general XML content to send as parameters and also they may have to parse the XML content in a SOAP response message.

The untyped and general interface will provide useless semantic compared to RPC methods with meaningful method names and typed parameters. For example, if java is used, the only parameter to the web service may be the ”org.w3c.Document” interface. With such an untyped interface you will not be able to get much help from the compiler to detect errors in the expected metadata parameter, but instead may get runtime errors.

1.1.9 Related patterns

The “Chained Service Factory” and “Unchained Service Factory” are almost the same. “Chained” means that hardcoded control statements is used in the ServiceFactory to instantiate predefined Façade classes. ”Unchained” instead uses reflection to choose which Facade to be instantiated based on the metadata in the XML-parameter.

This thesis document include one "Service Factory" found in one book and two other patterns named "Chained Service Factory" and "Unchained Service Factory" that were found in another book. Unfortunately, the "Service Factory" is very different from the "Chained/Unchained Service Factory" patterns. The "Service Factory" is a client sided pattern that creates a Web Service proxy for the client, while the "Chained/Unchained Service Factory" is a server sided pattern that creates a so called facade that will be invoked for taking care of the request from the client.

The concept of including metadata with information about what service method to invoke, is similar to the “Format Indicator pattern” [EIP 03].

If you compare this “Chained Service Factory pattern” with the J2EE patterns “Front Controller” [CJP 03] (FC) and “Application Controller” [CJP 03] (AC) then the FC is the SOAP engine that delegates the incoming request to the “ChainedServiceFactory”, which is similar to the “action management” part of the AC that chooses and invokes the request-processing components. In the AC pattern these components are called targets but are corresponding to the facades above in this pattern.