1.1 Unchained Service Factory Pattern

1.1.1 Name and Source

Unchained Service Factory
Page 152-162 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 that will not have to be modified in the future, while also letting the web service provider add new web services without having to recompile the existing web service implementation that is playing the role of a general controller.

1.1.5 Problem

Essentially the problem is the same as in the “Chained Service Factory”, but as an additional goal you may also want to avoid changing the existing web service interface implementation when you add new services. To be more specific, you may want to avoid the control statements (if/switch) that will choose one of a few hardcoded ServiceFacades, as illustrated in the “GetFacade” method in the class diagram in the “Chained Service Factory pattern”.

1.1.6 Forces

1.1.7 Solution

The solution is essentially the same as in “Chained Service Factory” but you will use reflection instead of choosing between a few hardcoded classes to instantiate. For a code example of how to implement it with C# see the book that is the source of this pattern. Below some java code is illustrating how the “ServiceFactory.GetFacade” might be implemented to “unchain” the ServiceFactory from using hardcoded classes, by using reflection:

```
private Facade GetFacade(org.w3c.dom.Document xmlDocument)
{
        String stringWithClassName = getClassNameFromMetaData(xmlDocument);
        return (Facade) java.lang.Class.forName(stringWithClassName).newInstance();
}
```

1.1.8 Consequences

In the code example above, the client is assumed to know the name of the concrete Façade class to be instantiated. Of course, a mapping method could be used instead to translate a publicly known name to an actual class name, with something like this:

String stringWithClassName = getClassNameFromPublicName(getPublicNameFromMetaData(xmlDocument));

The method “getClassNameFromPublicName” may then be implemented by translating into class names by using a configuration file, i.e. it would not be necessary to recompile when new services are published, but you just would have to tell the clients about the new service name and also change the configuration file that maps that public name to the actual name of the class to instantiate.

Of course, it is also possible to combine the “Chained Service Factory” and “Unchained Service Factory” by using control statements in the beginning of the “GetFacade” method to choose a predefined class to instantiate and then as a last resort in the “else clause” try to instantiate a class with reflection by using a class name that can be provided as parameter in the metadata. This possibility also means that it is not a potentially big problem to expose such a low-level detail as a class name, since if you for some reason would have to or want to change that class name, you may in your “Unchained Service Factory” GetFacade method add an if statement in the beginning of the method to instantiate the old class that have got a new name that no longer can support reflection from clients that use the old name. Of course, in that situation you will have to make the recompiling that the pattern usually avoids by only using reflection.

1.1.9 Related patterns

The “Chained Service Factory” and “Unchained Service Factory” patterns 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.