1.1 Service Factory Pattern

1.1.1 Name and Source

Service Factory Pattern
Page 261-277 in the book "Web Service Patterns: Java Edition" [WSP 03]

1.1.2 Also Known As

Note that this pattern have a similar name to the "Chained/Unchained Service Factory" patterns, but those two are very different as described in the related patterns section.

1.1.3 Type

1.1.4 Intent

To isolate the UDDI communication code that selects an implementation of a web service.

1.1.5 Problem

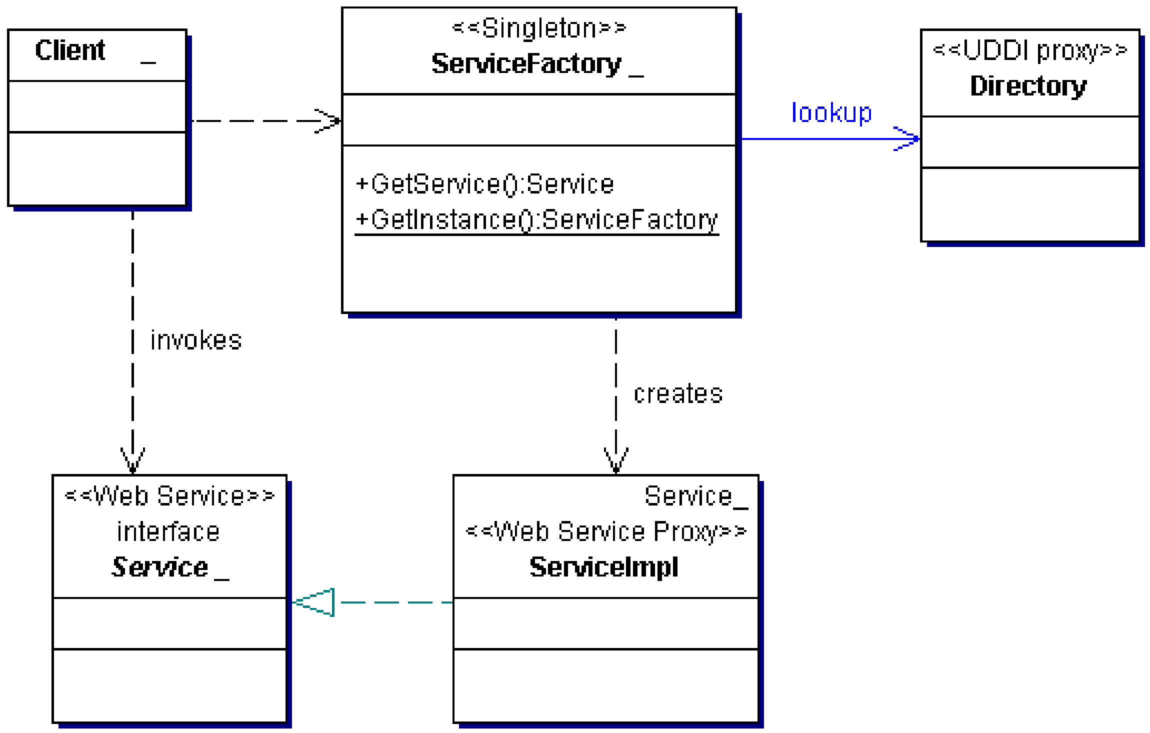
If you implement the “Service Directory pattern” to provide location transparency in your web service application, then the client code that communicates with UDDI can be quite comprehensive.

How can you avoid having a lot of UDDI code within the client classes that use the web service ?

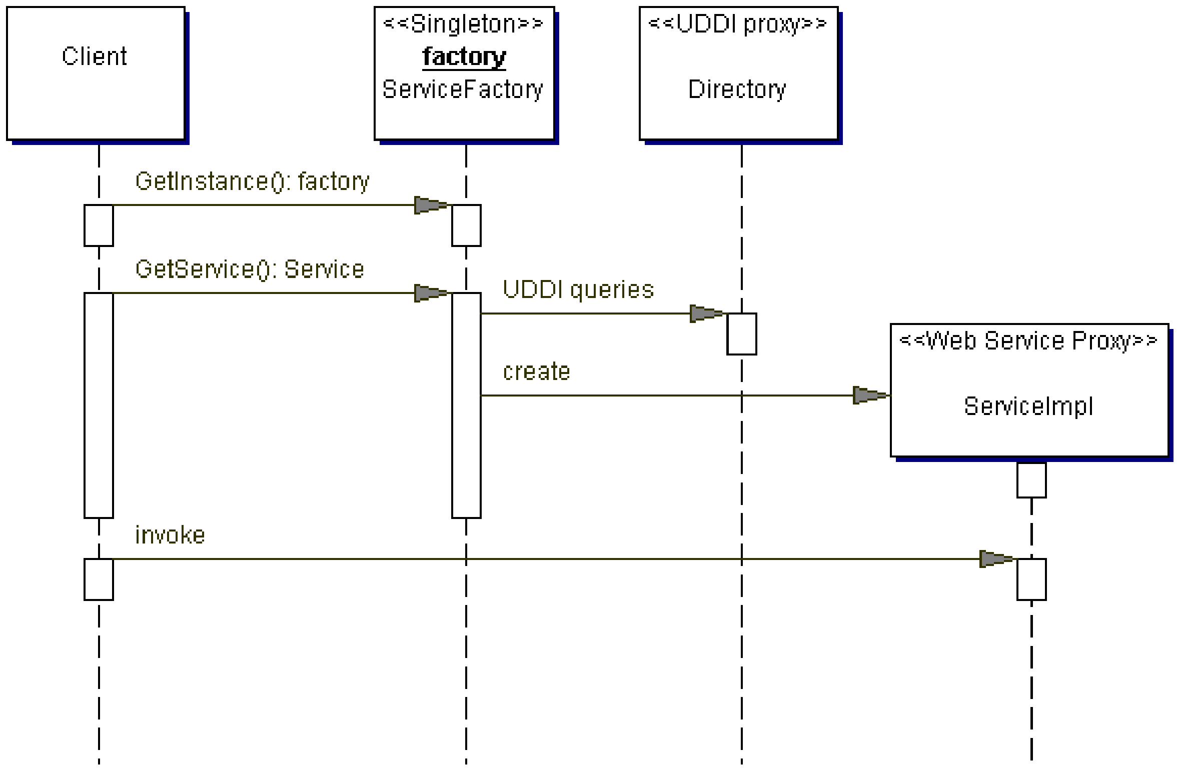
1.1.6 Forces

1.1.7 Solution

Isolate the complexities of the UDDI code into a ServiceFactory object.



The important thing to note here is that the Client never will have to use the Directory class itself but all the complexities regarding communicating with an UDDI Directory is done from within the ServiceFactory that has a simple interface that the Client will communicate with. The simplified sequence diagram only shows a message labeled “UDDI queries” from the “ServiceFactory.getService” method but it will usually include much more logic than a simple UDDI invocation before it can choose a Web Service proxy to return. Compare with all the messages sent from the “Client” class in the “Service Directory pattern”. Those messages will now instead be sent from “ServiceFactory” at the label “UDDI queries” in the diagram.



Client – The consumer of a web service.

ServiceFactory – A singleton [GoF 95] object that provides a simple API to the client and implements the low-level details of communicating with the UDDI directory.

Directory – A proxy object for the UDDI directory web service.

ServiceImpl – A web service implementation that is used by the client when the ServiceFactory has chosen and returned it to the Client.

Service – An interface for the web service. If an interface is used, then the client should only have a dependency to this interface and not to any concrete web service implementations, unless the implementation is generic and can be used for all web services. If the implementation is generic, with something like a “setInvocationURL” method that can be set from the ServiceFactory, then this Service interface does not have to be used but the client can instead just refer to the only proxy implementation class. If the generated proxy class is not generic enough to be reused for different implementations of a web service, and thus an interface needs to be used, then maybe the interface will not be automatically generated by the tool that generates the implementation from a WSDL file. In that case, refer to the pattern “Web Service Interface” to see how to create such a Service interface with refactoring.

1.1.8 Consequences

The client objects does not have a dependency to any particular UDDI proxy, and the implementation of the ServiceFactory might choose web service implementation by searching in many different UDDI servers. If the public UDDI cloud is used, it would not be necessary since the public UDDI directories are replicated, but different private UDDI servers provided by business partners might be used, and if those are not replicated it could be meaningful to search in more than one UDDI registry.

Instead of using UDDI to select a web service implementation, the ServiceFactory may hard-code the implementations that can be used. The client objects do not care whether UDDI is used or not, and the ServiceFactory can later be changed from hard-coding to start using UDDI without having to change any code in the client objects.

The ServiceFactory can receive a parameter, as in the parameterized Factory Method [GoF 95]. For example, in a web service for ordering products, a parameter might be the minimum number of a certain product in stock. Then the ServiceFactory will choose a business provider that currently have at least that many products available.

The UDDI invocations from the ServiceFactory can be a performance bottleneck, and therefore the factory may cache some of the information received from UDDI.

1.1.9 Related patterns

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.

This pattern is similar to the “Service Directory” but extends it with a ServiceFactory instance as an extra Façade (simplified API) [GoF 95] object between the client and the directory, that the client can use instead of directly using the rather complex API in the UDDI proxy object.

The “Web Service Locator strategy” in the “Service Locator pattern” [CJP 03] is a similar pattern that wants to hide the implementation details of the lookup mechanism. The description of the strategy is quite short, but it does not seem to abstract as much of the code as the “Service Factory” does, since none of the two public methods in the code sample returns a proxy object to a web service, but one of them instead returns a URL to a web service while the other returns a connection object. The “Service Locator” pattern includes caching of lookups to improve performance, and actually that is also mentioned in the "Service Factory", though the caching has not as much focus there.

The “Web Service Interface” pattern may be necessary to use if the tool that generates a class from the WSDL file does not also generate an interface that can be used for different implementations of the web service, and if the generated proxy class does not provide something like a “setInvocationURL” method that enables the class to be reused for different implementations of the web service.