1.1 Physical Tiers Pattern

1.1.1 Name and Source

Physical Tiers Pattern
Page 225-244 in the book "Web Service Patterns: Java Edition" [WSP 03]

1.1.2 Also Known As

-

1.1.3 Type

1.1.4 Intent

To facilitate interprocess communication between a stand-alone application and a local web service engine at the client machine that receives notification from a remote web service.

1.1.5 Problem

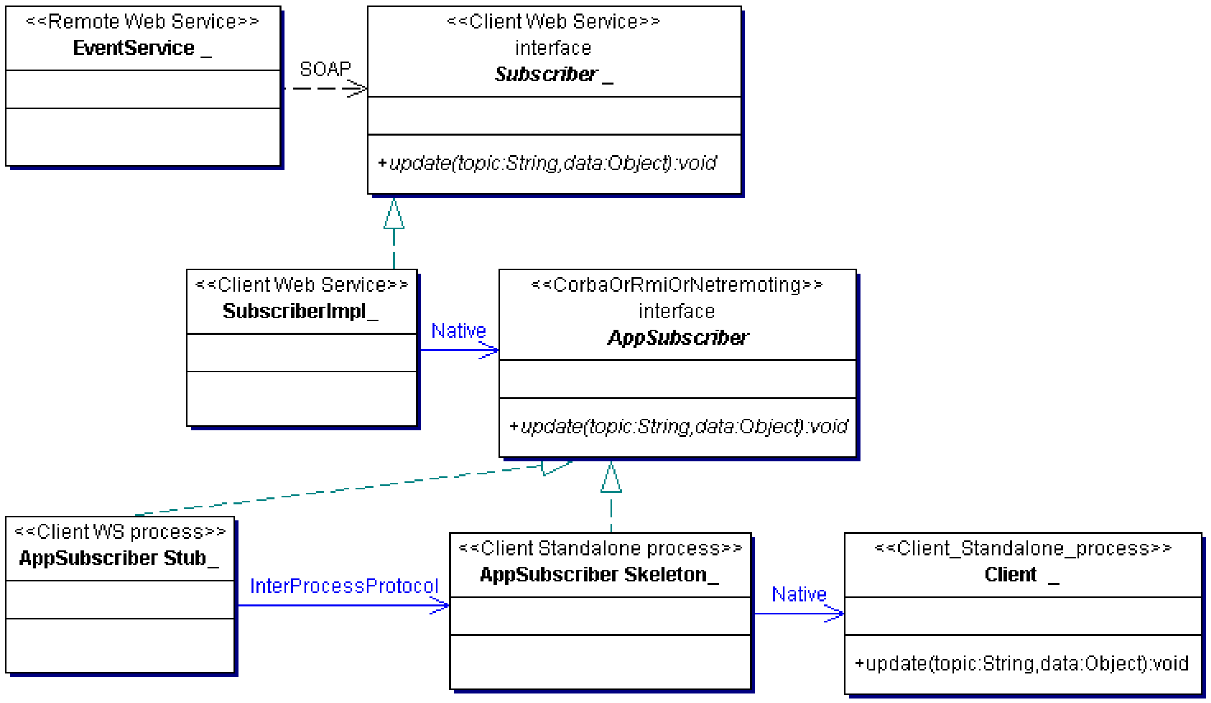
In notification patterns such as Observer or the Publish/subscribe pattern, the client must provide a web service itself to be able to receive the notifications from another web service. However, maybe the client that wanted the notification is a stand-alone application and is not executing as an object within the computer process of a web service engine.

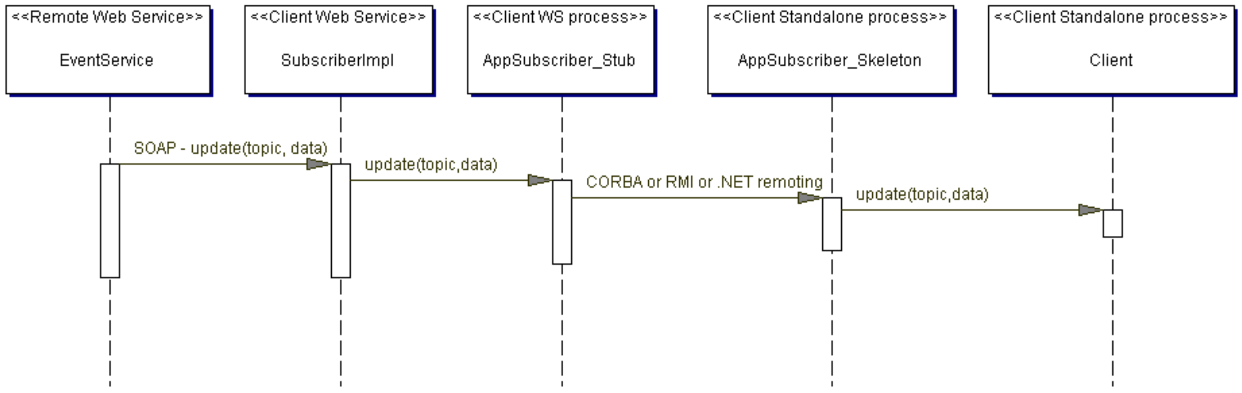
How can a stand-alone client application receive notifications from a remote web service if the client application is running in a different computer process than the local web service engine that receives the notifications to the client machine ?

1.1.6 Forces

1.1.7 Solution

Use an interprocess protocol that can be used for communication between objects in different computer processes. That protocol is used between the client web service and the client stand alone application. The protocol should of course be easier to implement than a web service, because otherwise you could just as well implement a web service within the stand alone application, as described in the “Faux implementation pattern”. If both the client application and the client web service are implemented in java, then RMI is the most natural choice, and if both are written with a .NET language then probably .NET remoting is the best choice.





EventService – A remote web service providing a subscription or notification mechanism. It could be an "EventService" as described in the "Publish/subscribe pattern" but it could also be an "Observable" as described in the "Observer pattern".

SubscriberImpl – A web service at the client machine that receives SOAP notifications about some event. It communicates with the stand-alone application with some interprocess protocol, for example java RMI through a stub object.

AppSubscriber_Stub – A proxy object, executing within the client web service, which can invoke methods of the AppSubscriber_Skeleton that executes in another computer process.

AppSubscriber_Skeleton – An object that executes in the client stand alone process. It can receive method invocations from the AppSubscriber_Stub in another computer process.

Client – The object in the stand-alone application that wanted to receive the notification about something.

1.1.8 Consequences

If it is not possible to use Java RMI or .NET remoting because of the languages the stand alone application and client web service are written in, then CORBA might be another option, but then the question is whether CORBA really would be easier to implement compared to provide a “Faux implementation” of a web service in the stand alone application.

1.1.9 Related patterns

“Faux implementation pattern” can be used to implement a simple web service in a stand-alone application, instead of using this pattern.

A “Connector pattern” is mentioned some times in the source book [WSP 03], but it seems unclear whether this should be considered as a synonym to “Physical Tiers Pattern” or exactly what the difference is between those two patterns.