1.1 Faux Implementation Pattern

1.1.1 Name and Source

Faux Implementation Pattern
Page 245-260 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 provide a simple lightweight SOAP message listener from within the computer process of a stand-alone application when it needs to receive notifications from remote web services.

1.1.5 Problem

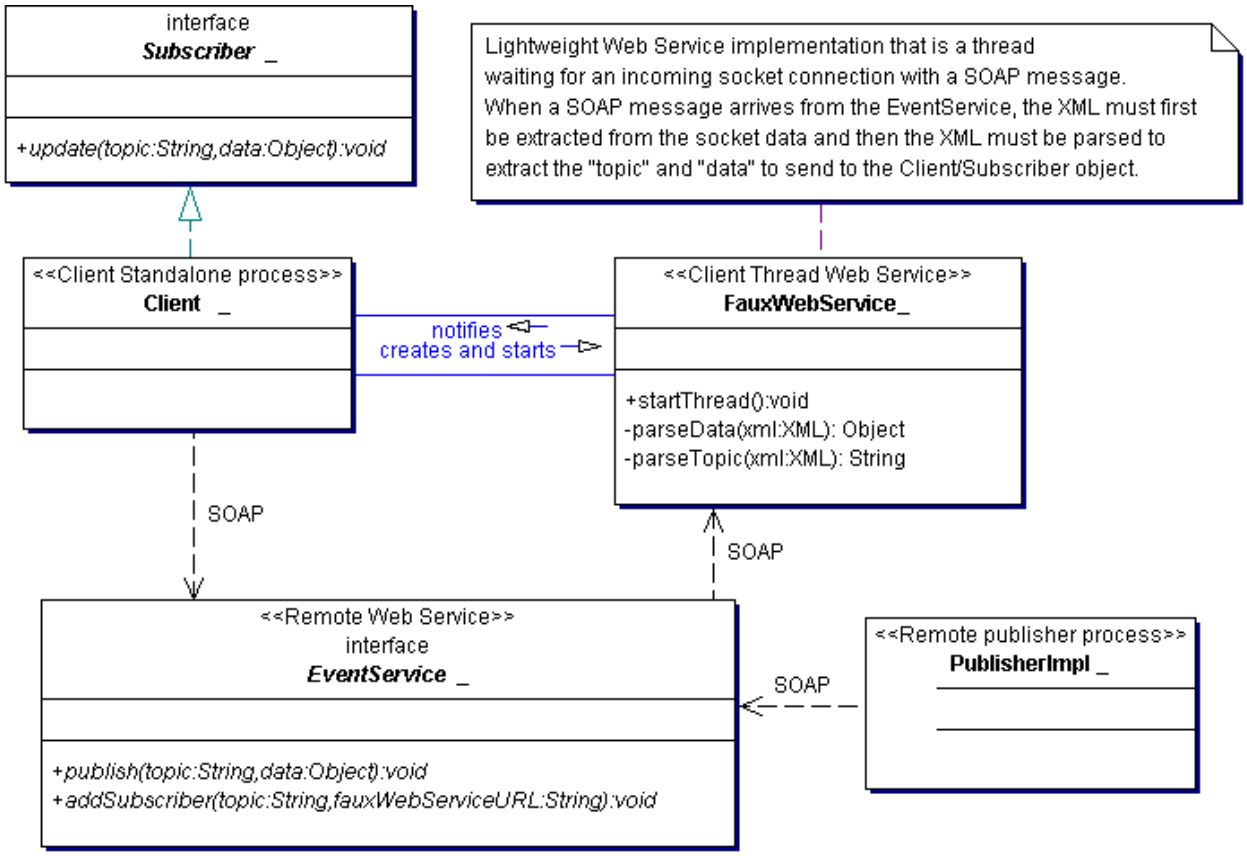
In some notification patterns, for example Observer or the Publish/subscribe pattern, the client must provide a web service itself to be able to receive the notifications from another web service. Typically, a stand-alone application is not bundled with an integrated full-blown web service engine. If a local but external (from a computer process aspect) web service engine at the client is used to receive the notifications from another web service, then somehow the local web service needs to communicate with the stand alone application through an interprocess protocol, as described in the “Physical Tiers Pattern”. However, if both the stand alone application and the web service is not written in java or if both is not written in .NET then you would have to use CORBA or some other rather complicated protocol, and you maybe would prefer an easier solution.

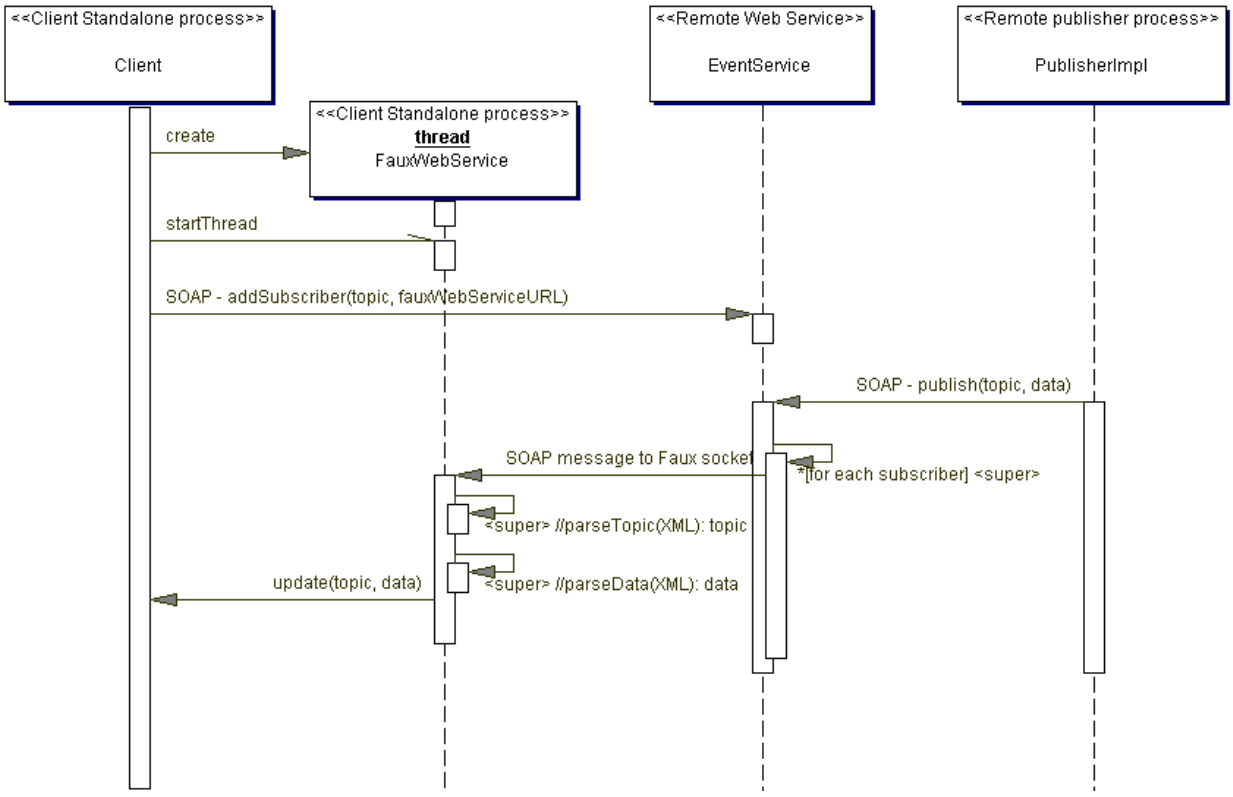
How can a stand-alone client application, that lacks integrated web service provider features, receive notifications from a web service ?

1.1.6 Forces

1.1.7 Solution

You can implement a simple web service engine by using a class that executes in a separate thread and is waiting for incoming socket connections with SOAP messages.


interface
Subscriber _
+update(topic:String,data:Object):void
Lightweight Web Service implementation that is a thread
waiting for an incoming socket connection with a SOAP message.
When a SOAP message arrives from the EventService, the XML must first
be extracted from the socket data and then the XML must be parsed to
extract the "topic" and "data" to send to the Client/Subscriber object.
<<Client Standalone process>>
Client _
notifies
creates and starts
<<Client Thread Web Service>>
FauxWebService_
+startThread():void
-parseData(xml:XML): Object
-parseTopic(xml:XML): String
SOAP
SOAP
<<Remote Web Service>>
interface
EventService _
+publish(topic:String,data:Object):void
+addSubscriber(topic:String,fauxWebServiceURL:String):void
SOAP
<<Remote publisher process>>
PublisherImpl _


<<Client Standalone process>>
Client
<<Remote Web Service>>
EventService
<<Remote publisher process>>
PublisherImpl
create
<<Client Standalone process>>
thread
FauxWebService
startThread
SOAP - addSubscriber(topic, fauxWebServiceURL)
SOAP - publish(topic, data)
SOAP message to Faux socket
*[for each subscriber] <super>
<super> //parseTopic(XML): topic
update(topic, data)
<super> //parseData(XML): data

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

Client – An object in the application that wants to receive a notification through the Subscriber interface it is implementing. The object creates the FauxWebService in a thread and then when it subscribes to a topic at an EventService it provides the URL to use for invoking the FauxWebService when notifications are sent.

FauxWebService – The object runs in a thread with a socket waiting for incoming SOAP messages, and when such a message is received then the XML will be parsed to receive the topic and the data to forward to the Client subscriber in the update method.

EventService and PublisherImpl – These objects behaves exactly like they do in the Publish/subscribe pattern. They do not care about how the client application receives the notification. In other words, from their point of view, it is irrelevant whether the client uses the “Faux Implementation Pattern” or the “Physical Tiers Pattern” or even something else.

1.1.8 Consequences

The phrase “faux implementation” implicates that the implementation of the web service is not a “real” implementation of a web service. However, it can be discussed exactly what should be considered as a real implementation and what is a faux implementation. For example, a very easy faux implementation could do nothing more than simply be listening for any incoming SOAP message without even trying to parse it, but could just be used for triggering a request to the web service for getting more information about whether something has changed, as an option to using the “Event Monitor pattern” that continuously requests a web service. If you keep adding features to the faux implementation until you have implemented a full-blown web service engine, the question is exactly when did the faux implementation stop being faux ? And is it even possible to say that there is such a thing as a faux implementation as long as it implements the interfaces it is supposed to implement ? The answer to that question is not obvious, but the author of the source book [WSP 03] believes that the answer is yes.

When you implement this pattern, you are probably more or less reinventing the wheel, since most platforms already have web service engines that might be used instead. The reason for instead implementing a simple “faux” web engine yourself, is that it may be complicated to integrate an existing web engine into your application or that you think it would create an unnecessary large footprint (i.e. storage or run-time memory) to your application.

1.1.9 Related patterns

Instead of using this pattern the “Physical Tiers Pattern” can let a local web service engine communicate with a stand-alone application by using some interprocess protocol such as Java RMI or .NET remoting.