1.1 Service-Oriented Architecture Pattern

1.1.1 Name and Source

Service-Oriented Architecture Pattern
Page 35-56 in the book "Web Service Patterns: Java Edition" [WSP 03]

1.1.2 Also Known As

-

1.1.3 Type

Architectural design pattern.

1.1.4 Intent

To find services dynamically, i.e. in runtime, and start using them at once without having to modify any code first, and without caring about which platform and programming language the service is implemented with.

1.1.5 Problem

You may want to integrate different software systems within your own company, and you may also want to integrate over internet with applications from other companies, written in other languages and executed at different platforms. When your software is able to communicate with software written in other languages/platforms, this concept is called implementation transparency.

When your application is interacting with other applications you may want to be able to dynamically find other applications that implement the same interface. For example, you may have a system that can order books automatically from different resellers and choose the one that has the best price for a particular book. When a new bookstore has started its online business it would be great if your application automatically could start using it. This concept, of dynamically locating and using new components without having to modify the code to tell your application the location of the new component, is called location transparency.

How can you achieve location transparency and implementation transparency with Web Services, i.e. how can you dynamically find new services and invoke them without having to care about platform and programming language of the new service ?

1.1.6 Forces

You may have different software systems within your company that is written in different languages and executed at different platforms, and perhaps you have some kind of batch applications constantly running to export and import data that is needed in both applications. It would be better to integrate the applications. A similar reasoning may apply for your communication with applications from other companies. Today maybe you are using some ftp solution for importing and exporting data to and from your business partners.

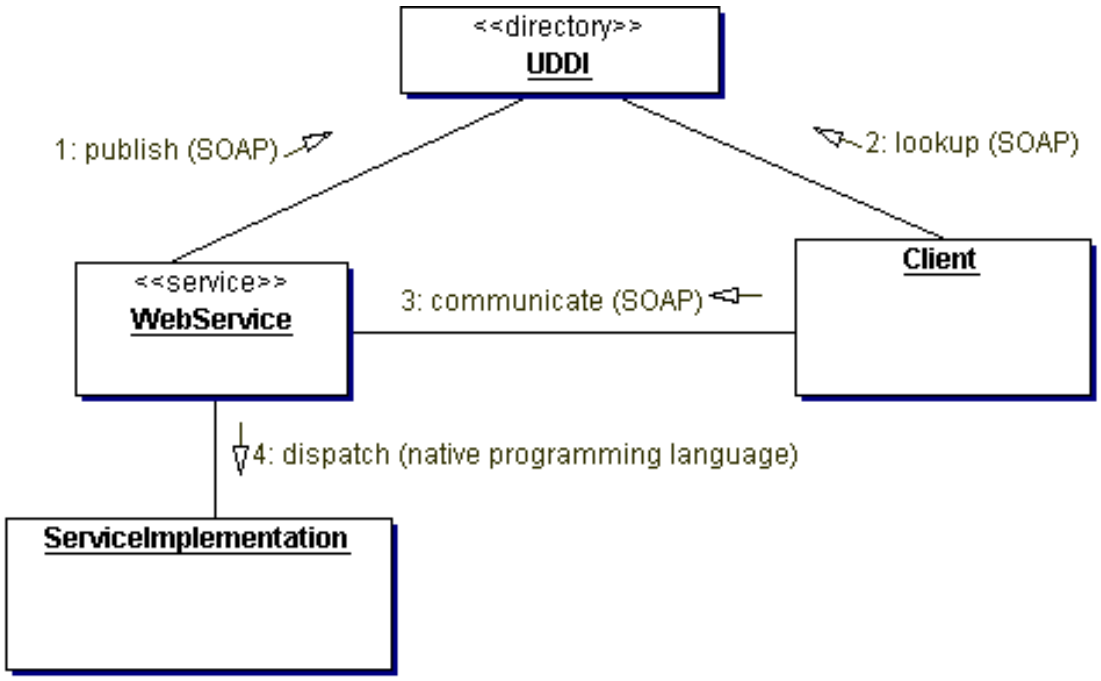
You may be using some kind of SOAP application for communicating with a business partner, but you can not start using other business automatically, and has to do changes manually, perhaps code changes that requires recompilation, but probably you have to at least do changes

in configuration to tell your application the URL of the server to send your SOAP messages to. It might be better to be able to automatically find business partners and start using their services.

1.1.7 Solution

The implementation transparency, i.e. the ability to ignore platform and programming language of the target web service can be achieved by using SOAP, which is an XML based communication protocol supported by most of the common platforms/languages.

The location transparency is achieved by letting the service publish its existence and its supported interfaces to a directory, and by letting the potential clients ask the directory about available services that support a particular interface. By doing these directory lookups, the clients do not need to know in advance where the services are located but the services locations can instead be dynamically discovered in runtime.



Note that the numbers in the high-level collaboration diagram above only illustrate the relative order of which things are happening, and they are not all happening instantly. It may take months after a service has been published (1) before any client finds it in a lookup (2), but when it has been found it can start communicating (3) at once, assuming that a proxy class for the interface has been generated previously. However, if it is the first time a particular interface is going to be used then the messages (2) and (3) is not being done in an immediate sequence, but then the client will first have to generate a proxy class, as described in the “Architecture Adapter Pattern”, and in this situation the location transparency do not really apply.

1.1.8 Consequences

The performance may possible be a problem because of the necessary overhead involved when the SOAP messages are being translated to and from XML and the native programming language in each node in the diagram above.

1.1.9 Related patterns

The Service-Oriented Architecture Pattern (SOA) pattern is similar to the “Client-Dispatcher-Server pattern” [POSA 96] and the “direct variant” of the “Broker pattern” [POSA 96]. Here the word “direct” implies that the client communicates directly with a server when the broker has helped to locate the server, while the indirect variant of the Broker means that all messages are passed through the broker. The book [ESP 03] also describes the “Broker” pattern, and uses the name “Broker as Intermediary” for the indirect broker in [POSA 96].

The SOA provides implementation and location transparency. Thus it can be considered as a combination of the two patterns below.

The “Architecture Adapter Pattern” provides the implementation transparency by using platform independent XML/SOAP and generating proxy classes to avoid doing manually low-level SOAP/XML-programming.

The “Service Directory pattern” provides the location transparency by using UDDI to lookup Web Services that implements the same interface as the Web Services proxy that your application is using.

The SOA pattern is similar to the general integration pattern “Message Bus” [EIP 03]. It describes an architecture that enables separate applications to work together in a decoupled fashion such that applications can be easily added or removed without affecting the others.

SOAP is the most common communication protocol for a web services based SOA, but as described in the “Self-describing Service pattern” [PLoP 02] the service provider can use WSDL to support other protocol bindings.