# Issues with Web Services as a Message Transport

December 10, 2006
Paul A. Knapp
Continovation Services Inc.
pknapp@continovation.com

## Overview

Web Services has evolved as an efficient means of exposing application interfaces (application functions) in a low level transport and platform agnostic way to other applications which need access to these services. Development and deployment efficiency increases brought about by tooling integration of Web Services (eg. Java, .Net, and others) have encouraged significant attention to this technology. This has lead to attention being given to use this technology as a General Message Transport or Message Exchange Protocol.

This document raises some issues found with the use of this technology as a General Message Transport in terms of development and/or processing efficiency with the hope that either: existing solutions to the issues identified can be outlined by the Web Service community; or that the Web Service Community can identify and make the necessary changes to Web Services to address the identified issues; or that the Web Services Community can provide guidance to the development community as to those development activities which are not well suited to a Web Services based solution.

## Message Transports

The purpose of a Message Transport, or Message Exchange Protocol, is to move message content from a Sending Application to a Receiving Application. The sending of a Initial Message from the Sender to a Receiver may or may not result in a Response Message or transport level acknowledgement, and the messages exchanged may or may not travel through intermediate message handling systems. A common scenario is where all message sessions are initiated by one party type, the Client, and may or may not be responded to by a Host receiver.

Examples of Message Transports currently in use are: various implementations of VISA; HL7 LLP; HL7 MLLP; and HL7 ebXML. These and other transports may enclose the message content or Payload within a wrapper of some sort, and may provide other features such as reliable messaging, once and only once delivery, and additional payload or exchange model meta data.

The challenge for an implementer is that they must obtain or develop a system which supports the selected, or multiple, Message Transport exchange model(s) and then integrate that into their business system(s). For message sending this includes optionally checking the payload, creating the message envelope, and low level message sending.

For message receiving this includes parsing the message envelope, checking the structure, harvesting the payload, and optionally checking the payload.

The opportunity brought about by the growth of Web Services technology and tools is that this tooling could be used to eliminate the need to obtain or custom implement new Message Transports by simply implementing Web Services as the Message Transport.

## Web Services

Simply stated, Web Services is a suite of technologies which enables a developer to easily: expose an application method as a 'Service'; to publish the service location as an 'Endpoint'; to map between 'Endpoints' and application methods; and to map between application data types and exchangeable data types. This allows developers on both the Client and Service sides of the exchange to be completely platform agnostic with respect to the other party with which they are communicating.

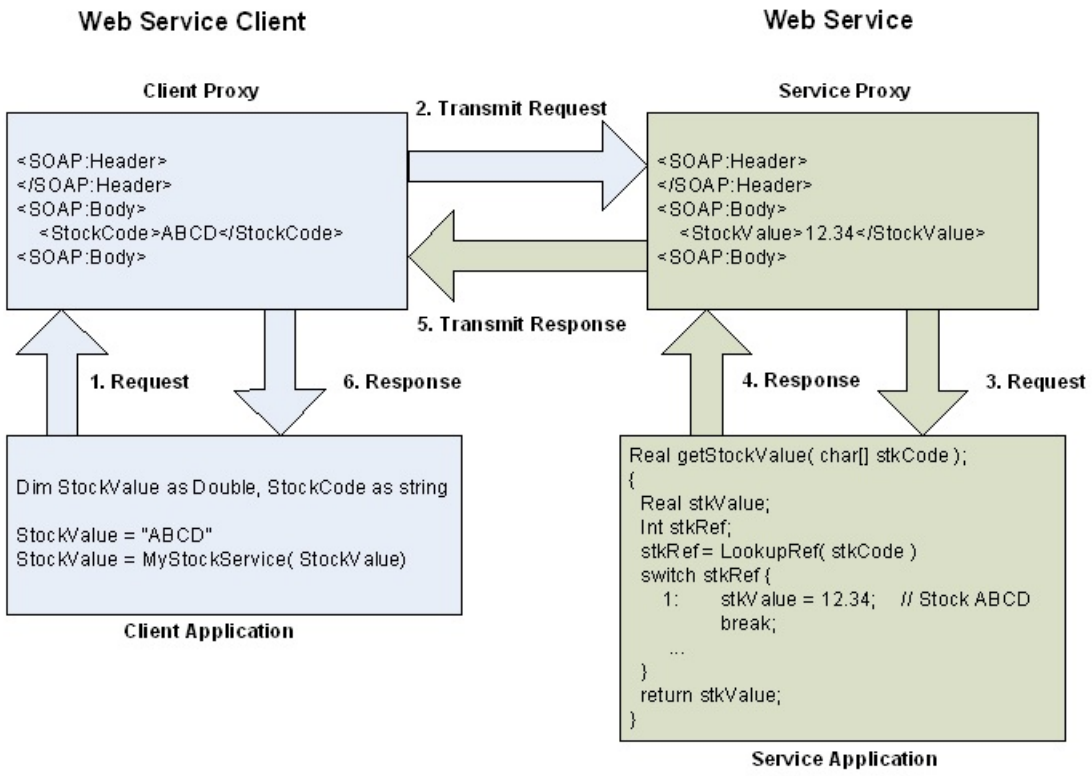


Fig 1 Simple view of Web Service Client and Service

As shown in the example above, the developer of the Client system is able to access the Service as exposed on the Service System as though it was a local function within the developer's environment. The message content, function arguments, are converted from the Client application platform specific data representation into an exchangeable data

format for the Client Proxy to send to the Service Proxy which converts the data into the Service application platform specific representation and calls the Service application to process the content. The Service application returns a response which the Service Proxy packages into a message to the Client Proxy which delivers the response to the Client application.

This is an effective technology, and efficient from the developer perspective in that they can focus on application and functional development and leave the Web Services 'plumbing' largely to the Web Services tooling within their development environment.

While there are many valuable features to the Web Services technology, two are key to the issues addressed by this document. The actual messages exchanged between the Client and Service Proxies may be of a variety of formats (SOAP is assumed and used for these discussions) but the logic is expected to apply to all:

1) The wire form of the message (header, body and packaging) information is created and consumed by the proxy, only the application relevant body elements are received by either the Client or Service application.

2) The body elements are converted into the local platform specific data format for either the Client or Service application.

These features are fundamental to the ease of design, integration and consumption of Web Services.

**Application Environment Diversity**

An important consideration is how diverse the environment is with respect to application development tools, operating systems, and computer hardware. Generally one does not choose for their environment to be very diverse, typically to minimize infrastructure, knowledge and support costs. However, there are many business forces which tend to drive IT environments to greater diversity over time such as:

Business or functional mergers which bring together multiple homogeneous but different systems.
Integration of specialized software or hardware which is based on a different platform, eg a database or specialized acceleration system.
Growth of work coupled with hardware improvements may lead to varied hardware platforms.
Developments in software technology, new projects and a fundamental inability to convert everything at the same time may result in supporting multiple software platforms.

The difficulties and processing costs associated with moving information between applications located on separate systems or between applications from different development environments within the same system should not be underestimated.

**Common Application Environments**

While that following examples are not intended to be exhaustive, they are intended to be exemplars of commonly occurring environments in which communicating applications live.

**Case A:**

**Client and Service both are the terminal applications for the work being exchanged.**

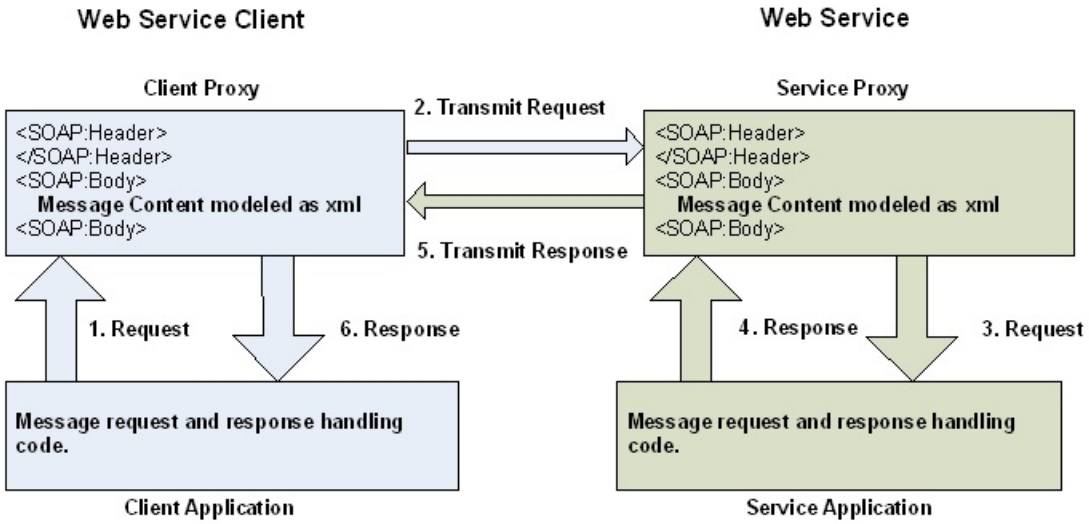


Fig 2 Both Client and Service are Terminal Applications

If the client does not need to authenticate with the Service, or needs to authenticate (eg. via SSL) but the Service only needs to know that the authentication passed, then this is a perfect candidate for Web Services mediating the exchange of information. Both the Client and Service developers need to perform a minimum of work to attach the Web Services technology to their respective applications to extend the functionality of their applications to the other.

However, the Service application may be challenged if:

The Service application needs to know the authentication credentials or context to perform it's processing, such as determining if the question asked it allowed for this type of Client.

The Service application needs to store the 'wire form' of the question, the input arguments, and the question was not a simple number or string, but some other complex type, such as xml encoded, which has now been converted to an object.

The Service application needs access to the header or message package wire form for storage or message processing.

**Case B:**

**Client is a terminal application, but the Service applications routes to worker applications.**

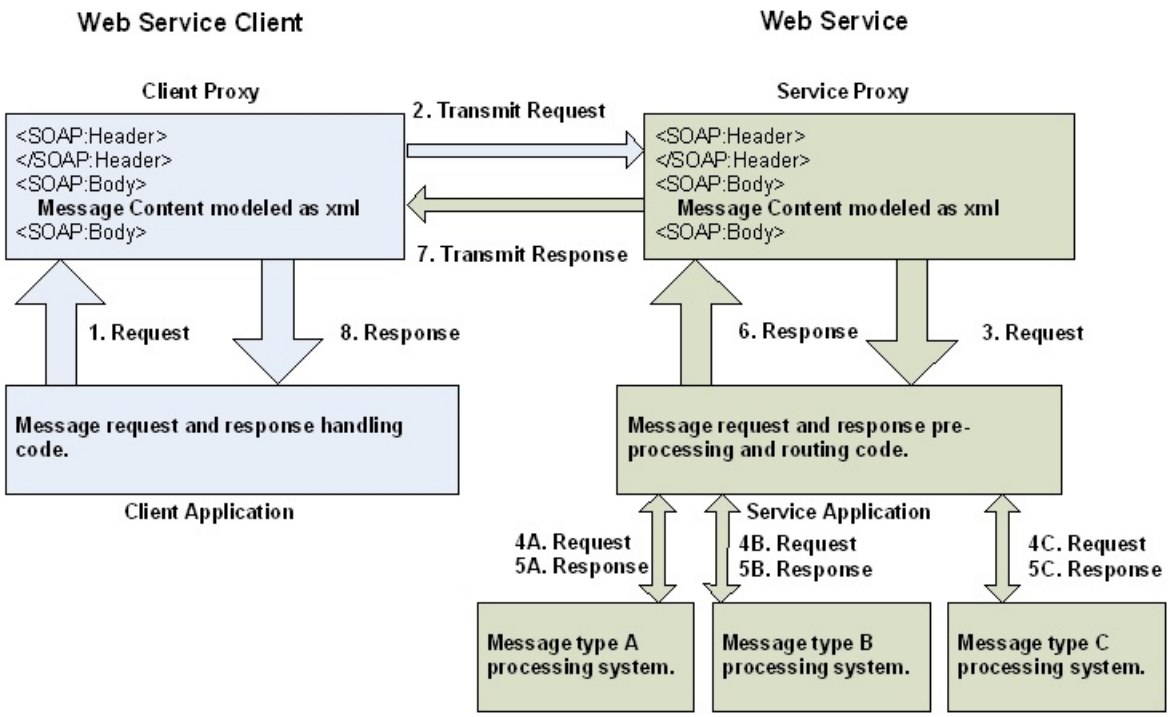


Fig 3 Client is Terminal application, Service application routes to Worker applications.

The worker application may be processors of different message types (eg. one for Medical, one for Pharmacy, one for Dental, one for queries, etc.), or processors of the same message for different books of business (Medical for company A, Medical for company B, etc) or specialized processing subsystems (eg. schema accelerator, database hosted processing, xml <-> string form translator) or any other type of distributed processing environment.

In this situation not only are the issues above present, but the Service application must also pass the question, arguments received, on to the worker applications for actual processing, and typically this exchange must be in string or wire form of the original message.

**Case C:**

**Client application is a messaging agent for the actual client side application and Service application is the terminal application for the work being exchanged.**

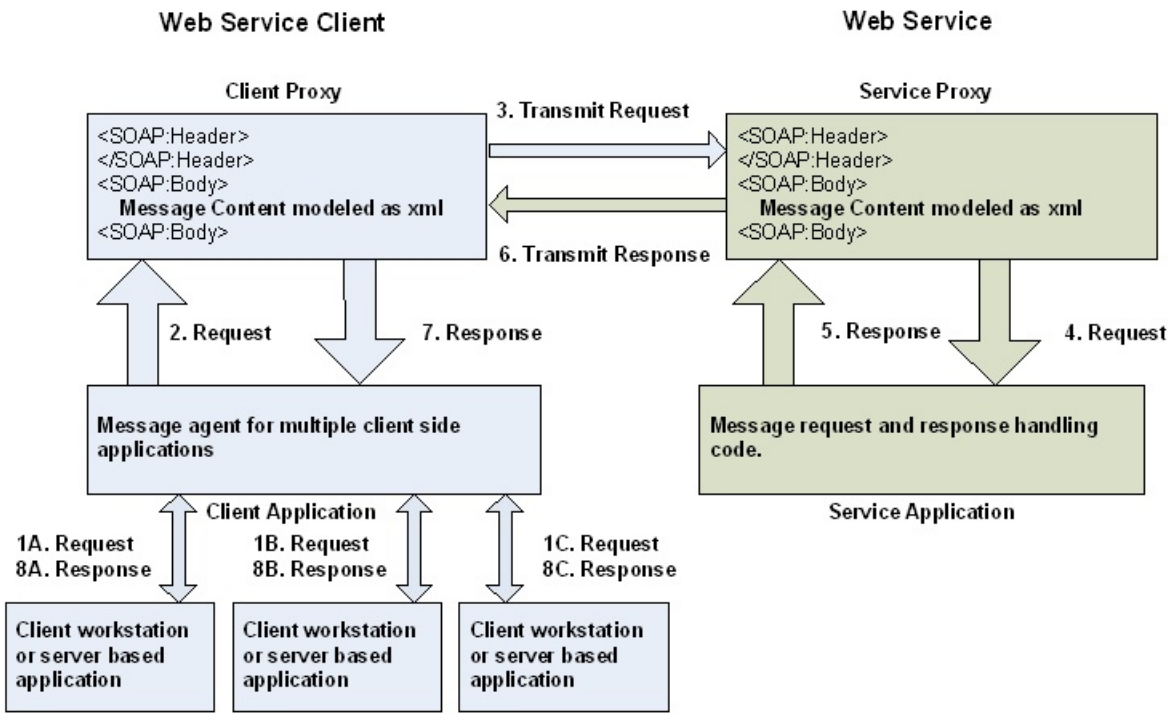


Fig 4 Client is a message agent, Service application is the Terminal Application

The same issues are present again here when the actual application which originates the message and consumes the response is functionally remote from the Client application. The application separation on the client side may be due to the messaging system being provided as a service to a cluster of client side applications, or a database may be used to intermediate, or the client application and the Client message agent may be from different sources, or a variety of other reasons.

**Case D:**

**Neither the Client nor Service applications are the terminal applications.**

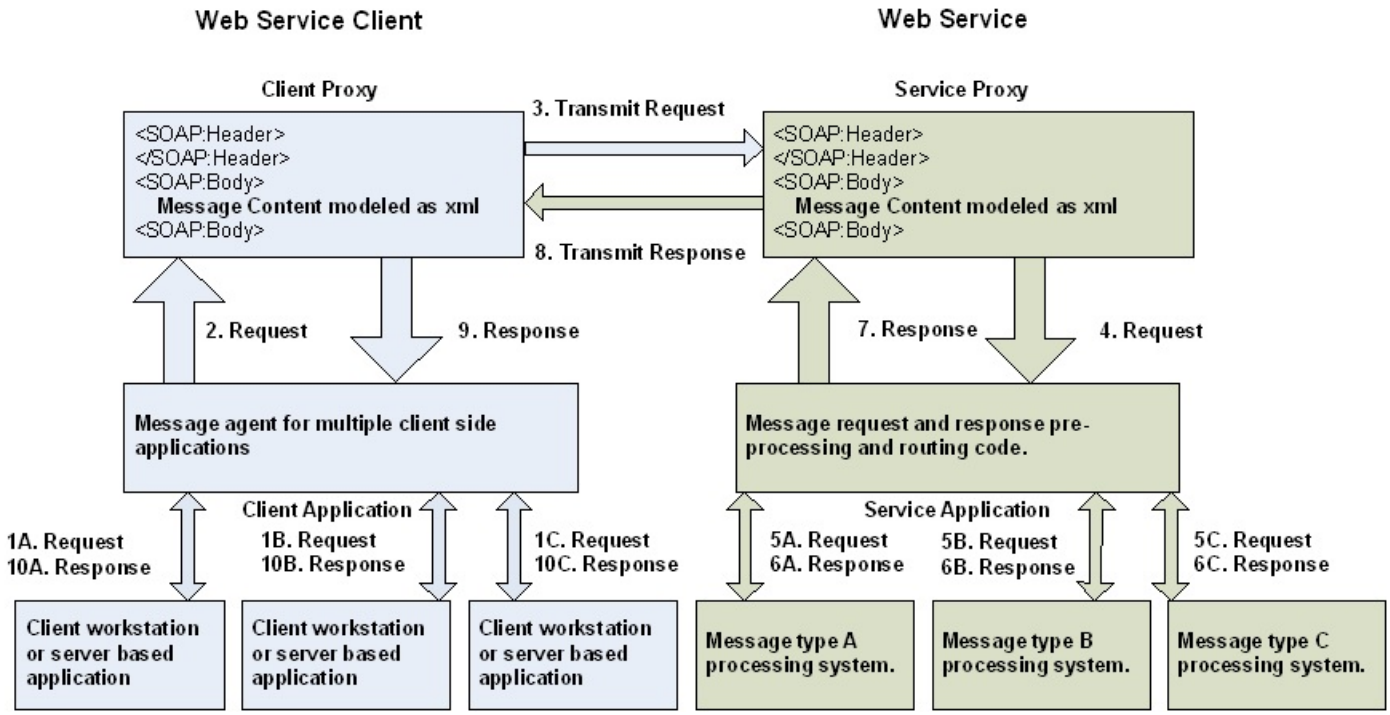


Fig 5 Client is a message agent, Service application routes to Worker applications

The issues already seen above are again seen here, and the impacts may be felt on both sides of the message exchange.

**Conclusion**

When the Client and Service side applications are the terminal applications for processing the message content exchanged, Web Services, with some exceptions, is an efficient technology to employ for the information exchange.

However, when the terminal applications which originate the message content or receive the message content are remote, that is processes on different equipment or on the same equipment but developed in a different development environment, there are two key features of Web Services which provide hurdles, the overcoming of which decreases the efficiency of using Web Services technology in the first place. These are:

1) The wire form of the message (header, body and packaging) information is created and consumed by the proxy, only the application relevant body elements are received by either the Client or Service application.

   This results in the application not having access to header or authentication information which may be relevant to the processing application.

2) The body elements are converted into the local platform specific data format for either the Client or Service application.

   Applications receive complex content, such as xml, as objects which may not be able to be passed in their native form to worker applications on the local or a remote system.

These challenges could be addressed by:

Having the application reach back to the Web Services Proxy to obtain the wire form of the message received or credentials from the Proxy, if supported.
Having the application re-create the wire form by re-serializing the object.
Escape and encode all complex content, eg. xml, in string form.
Having the application emulate Web Services so that it natively receives the security context and the message wire form and is thereby free to perform whatever processing is needed.

However, each of these approaches reduces the development or processing efficiency. The third option presented above has already been considered and deemed inappropriate by experienced implementers in the HL7 community. The latter option completely avoids the efficiency advantages of Web Services by 'pretending' to implement Web Services to maintain development or processing efficiency and serves to reduce the solution to now being exactly the problem which it was intended to avoid.

Also, while each message exchange use case will likely have a different mix of Cases A, B and C represented, for a significant number of messaging use cases the predominant computing infrastructure models are likely to be Cases B and C or the combination of both as shown in Case D. There is no business advantage seen for both sides of a

message exchange to be pretending to implement a protocol which meets neither of their actual business needs.

**Potential Solutions**

If there are ready solutions to these issues within Web Services as currently defined they are not known to the author or the reviewers of this document, and so some edification from the Web Services community would be greatly appreciated.

Should this use of Web Services, which attempts to extend the role of the technology beyond a simple efficient service model into a message exchange role, be asking too much of the technology, then this guidance would be useful to the developer and implementer communities so that the benefits of the technology may be sought in more appropriately cast problem areas.

Finally, if the solutions don't currently exist but there is interest in solving these issues with Web Services, then, only as a suggestion for discussion, it is proposed that there be added a flagging mechanism into WSDL to allow the indication that the Web Proxy is interfacing locally with a non-terminal application which results in the authentication context information and wire form of the exchanged message being passed in as arguments to the application rather than the current argument architecture. Further flagging may be appropriate to allow control over the amount of message schema processing performed by the Proxy and the conveyance of that processing to the application.