Serdica J. Computing **5** (2011), 15–38

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

EXTENDING OBJECT-ORIENTED NETWORK PROTOCOLS VIA ALTERNATIVE TRANSPORTATION BINDINGS (WEB SERVICES OVER XMPP)

Mihail D. Irintchev

Abstract. Distributed and/or composite web applications are driven by intercommunication via web services, which employ application-level protocols, such as SOAP. However, these protocols usually rely on the classic HTTP for transportation. HTTP is quite efficient for what it does—delivering web page content, but has never been intended to carry complex web service oriented communication.

Today there exist modern protocols that are much better fit for the job. Such a candidate is XMPP. It is an XML-based, asynchronous, open protocol that has built-in security and authentication mechanisms and utilizes a network of federated servers. Sophisticated asynchronous multi-party communication patterns can be established, effectively aiding web service developers.

This paper's purpose is to prove by facts, comparisons, and practical examples that XMPP is not only better suited than HTTP to serve as middleware for web service protocols, but can also contribute to the overall development state of web services.

ACM Computing Classification System (1998): H.4.3.
Key words: web-services, distributed web applications, protocols, SOAP, XMPP.

1. Introduction. The Internet has walked a long way from the era of the early Bulletin Board Systems, through the dawn of the WWW—the times of the static HTML single-page web sites, to reach the state it is in today, described by terms such as Web 2.0, cloud computing, and distributed web applications. The main mode of communication has shifted from passive reading/reception of information to active collaboration and web sites have grown to become web-based applications, often distributed among multiple servers.

The open global network created an environment where not only humans interact with web applications via browser interfaces, but applications also communicate with each other via APIs, and their remotely distributed parts intercommunicate in order to behave like a single body. The interfaces which allow this form of communication are called "web services".

Web services are more of an abstract term than a concrete tool or technology and can be implemented in numerous ways, using various protocols. When web services are used for closed intra-communication between parts of a distributed web application they can even employ custom-built proprietary protocols, written to serve and to be used only by the given application. However, when these APIs are intended for open communication with external known or unknown clients, standard protocols need to be used.

One of the most common protocols to carry web service transactions is SOAP. The abbreviation officially stands for "Simple Object Access Protocol", and, as is evident from its name, is intended for transporting object-oriented type of data/messages/function calls, encoded in XML [12]. The protocol's initial version came out in 1998 as a part of a Microsoft project at that time, to be later standardized by the W3C [12]. Its current standard version is 1.2.

Another popular protocol that is used to carry web services is REST—Representational State Transfer, which is directly related to HTTP, since it was designed in parallel with that protocol and is based on the same concepts and operations [11].

It should be noted that in terms of the encoding of the transferred data web services are not limited to XML, although it is a very flexible and powerful mark-up language. Many modern WS solutions tend to use lighter object-oriented data encoding formats, such as JSON [4], due to the considerably smaller encoding overhead. This is especially true when the data types that are about to be communicated are not very complex.

From the point of view of both the OSI 7-layer network model [3] and the Internet Protocol Suite [9] SOAP and other web service protocols stand on the highest network level—the application level. They can serve as a base for

building different web service protocol suites, but ultimately have to be carried by some other protocol on the same or a lower level in order to be send/received through the network. The W3C specification for SOAP allows it to be bound to virtually any carriage protocol [12], but it is most frequently bound to HTTP, although that protocol is hardly the one most fit for the job. The popularity of its use may be attributed to the fact that most servers running web sites must have an HTTP service running anyway, so this way there is no need for a separate software running to handle the web services' requests, or that most firewalls would not block port 80, which is the port commonly used by HTTP. These reasons, however, are not sufficient to generally justify the usage of this protocol for carrying SOAP or other web service protocols.

Having in mind the old saying that for solving most problems there is an easy way and a right way, I would say that using HTTP for carrying web service communication these days sounds more like the easy way than the right one.

2. Thesis. Currently web service traffic is carried out via SOAP and other protocols, running over HTTP more due to the forces of habit and simplicity than to functionality. Not that simplicity is bad, it is definitely good for a certain set of problems, but not for all. A very simple web service, exchanging small amounts of data between a single or several clients and a single web server in a synchronous way, not requiring authentication or any sophisticated protocol security, is definitely a candidate for the good old HTTP as an underlying protocol, since it will serve its needs perfectly.

However, more complicated scenarios quickly run into difficulties due to the inherent limitations of the HTTP protocol. Apart from being well established, simple, and light-weight, HTTP lacks a lot of functionality which is necessary or useful for implementing web services in an efficient fashion:

- **Synchronous:** HTTP is the type of request-response protocol, implying immediate response to the clients' requests from the server, otherwise the TCP session would time out (web administrators usually keep the TCP timeout setting of the web server low enough to prevent a great number of simultaneous sessions). This makes delayed or event-based response from the server hard to implement or nearly impossible. The client software has to maintain an open TCP session by polling the server at frequent regular intervals. Even then it is still possible for a timeout to occur, which brings up the need for reconnecting.
- **Almost no built-in authentication mechanism**: Although most HTTP servers allow certain directories/files to be protected by username and pass-

word combination, usually stored in a local file [1], this is a very rude and rigid way for protection, which is hard to maintain if the pool of allowed users and passwords is dynamic. Much more sophisticated authentication mechanisms are usually required by modern web applications.

- **Not intended to carry XML traffic:** Many IT specialists would respond to this claim by bringing up the fact that XML data is text-based data, no different in nature than any other text-based traffic, having its specially designated MIME type to describe it in the HTTP headers as such—*application/xml* or *text/xml*. This is unarguably true, but this also means that the protocol and the server providing the transportation do not know and do not care if the XML document that is being transported is well formed and/or valid. People may also argue that this is no job for the communication server, but provided that such a check will eventually need to be done at higher application level anyway, then why not do it at the entry point, provided that the protocol is known to be carrying a SOAP envelope or other XML-encoded messages.
- **Designed for one-to-one or many-to-one network topology:** Generally, the SOAP processing model provides for several possible roles a given SOAP node can occupy: sender, receiver, initial sender, ultimate receiver, and intermediary [12]. This allows for the so-called SOAP message path to be constructed, so that a message can originate from a given initial sender, traverse a path of N intermediary nodes (which can also process and/or amend the original message) before it reaches the intended ultimate receiver. This mechanism allows for building added-value chains of SOAP nodes. Unfortunately, when SOAP is transferred over HTTP the roles of sender and initial sender blend—they are one and the same node. The same counts for the receiver and ultimate receiver – again this is the same node, since there are actually only two actors – the HTTP client and the HTTP server with fixed roles, which cannot be reversed. Immediate response is expected by the receiver. There is an option to use a SOAP extension called WS-Addressing which adds transportation headers to the SOAP envelope, allowing for intermediates nodes (which support this technology) to actually create a chain [13]. WS-Addressing has to pass the addressing/transportation information burden to the SOAP envelope, since no such support is present in the underlying protocol—HTTP.

Due to the aforementioned limitations there is a number of applications which are hard to implement. Software developers run into problems and seek complicated solutions or workarounds when implementing such web applications

using web services over HTTP. Such software can be:

- Distributed web applications;
- One-to-many/many-to-many types of network nodes topologies;
- Those requiring good built-in security and encryption mechanisms;
- Those requiring easy and secure built-in authentication models;
- Applications requiring message queuing or other non-real-time communication;
- Applications employing an asynchronous mode of communication;
- Event-driven applications.

For many years now there exist XML-based application-level protocols, having many advantages that HTTP lacks, designed with the concepts employed by the latter list of applications in mind. These protocols are usually transported directly over TCP, not relying on additional intermediate application-level protocols. They employ designated client/server software, not exploiting the HTTP server, leaving it to do just what it is designed for—serve web pages to browsers. Such protocols can easily and neatly be used as middleware for carrying web service communication.

An exemplar message-oriented XML-based middleware protocol is XMPP (eXtensible Messaging and Presence Protocol) [16]. It was originally created in 1998 to carry instant messaging and was formerly known as jabber[1], due to its past and present association with the jabber IM platform. Its inventor Jeremie Miller decided to create an open-source standard for exchanging instant messages because he had enough of using four different closed IM services via running four separate client applications. After the jabberd server was released as an open-source project in 1999, many developers joined to contribute jabber client software written for Linux, Macintosh, and Windows, as well as server add-on components, code libraries for various languages, etc. The protocol was later standardized and officially released as RFC 3920 and RFC 3921 by the IETF in October 2004. It is being further developed and extended in the same open-source fashion through an open standards process run by the XMPP Standards Foundation [10].

This flexible protocol is currently used for a number of applications [10]:

- Instant Messaging
- Multi-user/group chat (MUC)
- Voice (VoIP) & video communication

[1]XMPP and jabber are used interchangeably in this paper when applied to the server software

- Geolocation
- Gaming
- Collaboration platforms
- Data/Content syndication
- Systems control

XMPP combines a robust list of features/services which provide the software developer with a strong arsenal of tools for building sophisticated web applications. These features will be examined in the following (exposition) parts of the current paper. XMPP servers (also commonly known as jabber servers) offer a number of built-in and additional services and can be used to create a network of WS nodes, perfectly serving as middleware for more-complicated WS-driven web applications. Even some of their most basic features, like service discovery, for example, can contribute a lot to building more versatile software.

These are the some of the areas in which web services using SOAP (or an other similar protocol) can be improved if transported over XMPP:

- Security
- Flexibility
- Routing
- Services & nodes discovery
- SOA (via message-driven communication model)
- Event-driven communication
- Value added chains
- Distributed/cloud computing

It should be noted that the latter of these are very hard (or nearly impossible) to implement over HTTP, even with additional SOAP extensions, so using XMPP as middleware not only improves web services but enables them to offer more than what is possible with HTTP as carriage protocol.

3. Exposition. XMPP communication is established between different XMPP nodes (usually clients) and one or more XMPP servers. The client nodes need to register (or be registered) to an XMPP server in order to communicate through it. From the view point of web services these XMPP nodes can play any role: client or server or even both, which is not possible with HTTP.

Here is a graphic showing the XMPP nodes and the server they connect to. The CISCO® router icon is intentionally used to depict the XMPP server,

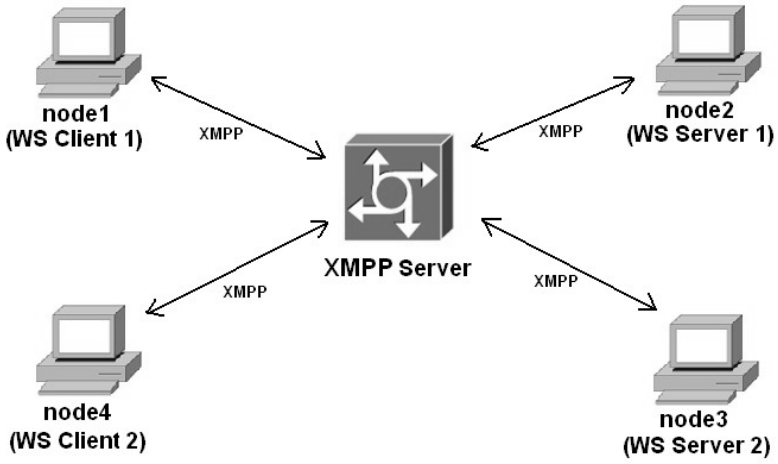

Fig. 1. Sample XMPP nodes and server

since it does act as a web service router in such setup.

The servers, on the other hand, can form a so-called server federation, and can intercommunicate, so that a node authenticated at one server can send and receive messages from nodes registered with another server [10].

When a connection is established between a node and the server a TCP session is opened, encryption options are exchanged, authentication (or sign-up) is performed, after which a continuous XML stream is started, opened by a <stream:stream> XML tag. Consecutively, when one of the nodes decides to close the session deliberately it would send a closing </stream:stream> tag. All intermediate communication is performed by sending designated XML elements, holding data and commands. These basic XMPP communication elements are called stanzas [10]. The most common stanzas are *presence*, *iq*, and *message*. The latter two are employed for carrying web services: *iq* should be used for request/response type of communication, while *message* is designed for one-way type of information submission [5].

As stated in the thesis, HTTP has a number of disadvantages for transporting WS data, which is one of the reasons that make the search for a better carriage protocol necessary. In brief, these disadvantages are:

- No native asynchronous communication support
- Poor built-in authentication mechanism

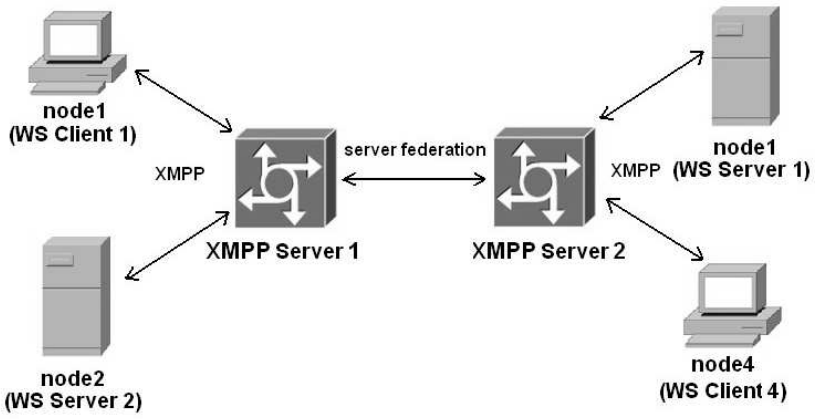

Fig. 2. XMPP Server Federation

- Not intended to carry XML traffic by design
- Not suitable for many-to-many, one-to-many type of network topologies, which are often employed in distributed/cloud computing
- No built-in event-driven communication support

Unlike HTTP, XMPP does not suffer from the above listed flaws. On the contrary, it has a number of additional features, from which web services can benefit greatly:

- **Channel encryption:** XMPP offers a built-in TLS encryption option [10].
- **Authentication:** The SASL authentication platform, supported by XMPP, offers a number of authentication options, ranging from anonymous access or plain text authentication to sophisticated authentication mechanisms, such as digital certificates or using a 3rd party Kerberos V server [10].
- **Presence:** Detailed presence information is exchanged between XMPP nodes, allowing other nodes to have knowledge regarding the availability status of a given node [10].
- **Contact lists:** Often called *roster* in XMPP terms, the contact list is a list of known (trusted) nodes that a given node communicates with, which is stored on the XMPP server.
- **One-to-one messaging:** Communication mechanism between two XMPP

nodes, classically employed by instant messaging systems, but actually fit for exchanging any sort of data [10].

- **Multi-party messaging:** Provides many-to-many type of communication in a virtual IRC-like chat room [10].
- **Notifications (events):** A very important and useful service (as will be described later in the paper), providing a mechanism for a number of nodes to be subscribed to a given node for certain events. When the events occur the subscribers get notified instantly via the XMPP server. It is possible to build hierarchical trees of nodes/services for subscription, allowing for categorization/specialization of the provided services [10]. This functionality is the backbone of message-driven asynchronous communication, allowing the building of SOA-based web applications.
- **Service discovery:** This feature allows a given node to survey the services provided by another entity and find additional entities, associated with it [10]. This service can also run in a very light-weight fashion over the DNS protocol and use it to discover local XMPP nodes and the services they have to offer (also known as DNS-SD).
- **Capabilities advertisement:** This extension to the present service aids the service discovery feature, allowing nodes to present information about the services they have to offer in shorthand notation. This info can be cached by other nodes to form a kind of a "yellow-pages" catalog with the nodes and their corresponding features [10]. This service, combined with the previous one (service discovery), can be used to provide an improved alternative to the existing WS-Discovery SOAP extension [8].
- **Structured data forms:** This XMPP feature defines a format for exchanging information between nodes via structured forms with defined data types, passed in XML (as almost everything in XMPP). This is useful for querying a given node for a set of data using a defined form with fields and receiving the corresponding response with the fields filled out. Good for communicating ad-hoc data between nodes [10].
- **Workflow management:** A service allowing interaction between nodes, following a structured workflow with support for typical workflow actions. This makes it possible to intuitively map business processes (usually following a defined workflow) to web services. Often used together with structured data forms [10]. This service can provide an alternative to the existing WS-CDL [14] and WS-BPEL [2] recommendations for implementing business processes and added-value chains over WS.

– **Peer-to-peer media sessions:** A feature allowing a media session to be negotiated and menaged between two XMPP entities. It can be used for implementing voice chat, video chat, file transfer or an other real time application (usually binary) [10].

Several of the features listed above can be particularly useful for implementing advanced WS architectures, while others are applicable for implementing and improving common functionality, used in simple scenarios as well. Such common features are *Presence*, *Authentication*, *Contact lists.* They can be used by "conservative" WS implementations even in a one-to-one scenario to perform basic operations, such as login, availability check, authorization:

– **Login**: The XMPP login is performed by negotiating a communication session with the XMPP server (usually secured over TLS), followed by an offering and choice of an authentication method (plain / digest-MD5 / digital certificate / other), depending on what the server and client support. At this point an XMPP client can also claim a new account to be created for it, in case the server supports automatic sign-up and this option is enabled. Once the client's identity has been determined the actual XMPP communication session begins [10]. This flexible login mechanism, offering different authentication options, is very useful, since the web developer can "outsource" the authentication part of the web service to the XMPP server, not having to worry about login and security on the application level. Also, it is more efficient and secure compared to web-based application-level authentication, passed over HTTP, since web sessions (if not re-established every time) rely on session identifiers (session IDs), usually passed as cookies or via GET method over HTTP. These identifiers can be stolen in several ways (via an XSS attack, for example) so that another entity can impersonate using the stolen session ID, thus obtaining access to the session. In XMPP the login is done once in the beginning and from there on an open TCP session is maintained between the node and the XMPP server, which makes it almost impossible to penetrate.
– **Availability info:** Once logged, the XMPP client (or node) submits information regarding its presence at regular intervals to the server and also receives back presence info for all the other nodes connected to this server and authorized by the given node. This is a very useful mechanism for keeping track of which nodes are up and running. The presence service also allows for more elaborate status information to be passed, practically containing any desired (and reasonably small) piece of info [10]. When applied to web services such can be the load status, for example, letting the client

nodes of a given SOAP server know how loaded it is at the moment, so that they may choose to connect to another, less loaded provider of the same service.

– **Authorization:** Each node has a contact list, stored on the XMPP server, defining other nodes which have been authorized to communicate freely with the given node [10]. This "trusted" list is pretty much the same as your ICQ friends/authorized contacts list, except that this XMPP feature can be very useful when dealing with web services.As with the login service, the web developer may transfer the burden of authorizing and checking the identity of nodes wanting to communicate with a given web service to the XMPP server. It will deal with the login and then will check whether these nodes are present in the node's contact list.

All of these will be demonstrated in the proof-of-concept/implementation part later in the current document.

The XMPP features allowing web services to be taken a step further in their development are: *Multi-party Messaging, Notifications, Service discovery, Capabilities advertisement, Workflow management*. These tools are powerful enough to open a whole new window of opportunities for WS developers to create what may be called in jargon "WS 2.0" web applications (similar in meaning to the nowadays popular expression "Web 2.0"). Here is how some of these features can be used to enhance web services run over XMPP:

– **Many-to-many/One-to-many communication**: Unlike HTTP, where the WS communication usually follows a one-to-one or many-to-one pattern, with XMPP it is possible to easily form other communication patterns, employing the Multi-party messaging mechanism. The participating nodes join a common multi-user chat (MUC) session and then perform their corresponding roles—listening / receiving / sending messages. It should be noted that it is not necessary for the nodes to be authorized by each other in order to perform group communication. They just need to be logged in the XMPP server and need to have joined the MUC session, identified by an XMPP ID. It is also possible for the nodes to remain anonymous, retaining their real IDs hidden, being represented by a "nickname" chosen for the purpose, provided that the server settings for this "chat room" allow anonymity.

– **Web Service Discovery:** The XMPP discovery and capabilities services, performed with the *disco#items* and *disco#info* commands respectively, are the means by which a given XMPP node can query the server (with the

former) regarding the provided services (say conferencing, subscriptions, other) and receive further information (with the latter) regarding a given service from the returned list. This structure is usually nested, constructing a hierarchy of nodes. This would mean that usually the queried service can return a list of items, which can be further queried with the *disco#info* (to understand what they are) and *disco#items* (to get further nested items, if any) commands. In other words, the service tree can be traversed recursively to create a full catalog of items and services, although a full traversal is not necessary in most cases. [10]

In the SOAP world a web service is defined by its WSDL file, which tells the other what messages it can receive and what responses it would give in return. It is possible to make SOAP-supporting service nodes to advocate their ability to deliver web services by returning the corresponding info to the *disco#info* command. Moreover, an XMPP extension can be implemented, causing these nodes to return the URI of their WSDL, so that the client browsing the hierarchy can create a catalog of the supported services with their corresponding WSDLs and XMPP IDs to connect to. This functionality can turn out to be the basis for creating a better alternative for discovering web services than the not-so-successful (and even less open) UDDI initiative [7].

– **Event-driven communication:** The XMPP publish/subscribe system allows web developers to build event-driven real-time WS-based applications with great flexibility and without the complexities imposed by the nature of the HTTP-based communication. This perfectly fits the SOA software model, in which communication is usually not synchronous and does not follow an uninterrupted line in time.

I hope to demonstrate at least some of these opportunities put into practice in the current paper (in the implementation/proof-of-concept part).

The XMPP protocol is extended through the so-called XMPP Extension Protocols (XEPs), each official one having its unique identifier, e.g., XEP-0004. These exists an official recommendation for binding SOAP to XMPP for transportation – XEP-0072. It defines the baseline that should be followed in terms of XML structure when implementing SOAP transfer over XMPP. This includes the XMPP stanza XML elements and attributes that should be used to hold the SOAP envelope in different scenarios, the WSDL binding section that should be set to specify the SOAP over XMPP binding, error condition rules, etc. [5]

Despite the existence of XEP-0072 this transportation binding is still quite new and not yet popular. Therefore, implementing a SOAP over an XMPP

based web service in any of the common web development languages of the day (PHP, Perl, Ruby, Python, etc.) is not a straight-forward job due to the lack of ready-made code libraries that would make the job easier. There exist enough SOAP libraries for all modern web development languages, some of them even have it as a built-in component (e.g., PHP 5.x). However, XMPP code libraries [17] are still not as many and not as stable as SOAP libraries, which have been around for nearly a decade. Nevertheless, the situation gets better by the day as more and more developers from the open-source community contribute XMPP libraries they have developed/improved to do their job, since XMPP is getting more and more popular for all kinds of applications.

Presently time there already exist some solutions employing XMPP to exchange information in a web service-like fashion. These are mostly specialized use cases, serving a particular specialized problem, such as a distributed computing network for solving biomedical/chemical problems [15].

At the time of writing this paper I could not find any available open ready-made SOAP over XMPP solution. Most probably this is due to the fact that the idea is very new and that even if there are such solutions most of them would be custom-built ones. I did find an open-source web service over an XMPP client/server library for Java, which actually did not use the SOAP protocol [18]. This library was used in the distributed bioinformatics solution mentioned above [15]. I also found a paid XMPP SDK for Microsoft .NET, used in some web software based on Microsoft technologies, such as Silverlight, .NET and .Net Compact (mobile platform) [19], but no real alternative for the most popular scripting languages of the web, such as PHP, Perl, Ruby. I believe it is only a matter of time before people implementing similar solutions and applying them for solving their real life problem of the day will compile and contribute a base library/framework for quick development of SOAP over XMPP web services.

4. Implementation (Proof-of-Concept). I would like to start this part with a disclaimer that although the demonstrated solution can serve as a basis for developing a fully functional PHP code library for SOAP over XMPP it is currently in an early development state. Its sole purpose right now is to serve as a proof-of-concept code, which demonstrates in practice the ideas expressed in this paper.

As mentioned, the solution is written in PHP (ver. 5.2.x), using the following code libraries:

- The built-in PHP SOAP module
- The NuSOAP library

– The XMPPHP library (modified and extended by me to provide more XMPP features)

The solution runs Apache 1.3.x web server, but theoretically should be able to run on any other web server with PHP 5.2.x support, having all the modules required by the upper-listed libraries.

The XMPP server side runs an Ignite Openfire server, which is developed in Java, making it a multi-platform software. It may or may not employ a 3rd party database server to store its data and settings. In my case I chose to use the MySQL DBMS so that the server's information can be easily managed by external software or even manually via any MySQL client software.

All the software used is free-of-charge and open-source, providing the solution with a good and flexible base free from obligation to commercial licenses base for further development and distribution.

Using the technologies and libraries listed above I implemented a solution including several scripts acting as client/server XMPP nodes and an XMPP server they connect to. The nodes, also called XMPP bots (or simply bots) or scripts from now on, can act as SOAP clients or servers, requesting or delivering web services via SOAP over XMPP. It should be noted that in such a layout it is not necessary for a given node to be strictly a client or a server—it may assume both roles, depending on the needs.

The source code of all example scripts, base classes, and the used libraries can be obtained from here: `http://irintchev.com/soapxmpp/code.zip`

In order to demonstrate the ideas expressed in the former sections and to present practical proof for the claimed thesis I developed the following sample use cases of WS-driven web applications, serviced via SOAP over XMPP, making use of the features of the XMPP server:

I. Use Case 1 – Classic client-to-server case. A travel-assistant application needs to find out an exchange rate from a given currency to another for a budget sheet. A WS client uses an XMPP bot to access a WS server, running on another XMPP bot. The nodes are connected through an XMPP server.

It should be noted that it is not necessary for the XMPP server to reside on a different physical server. It can run on the same machine as one of the nodes, for example the designated SOAP server. It is possible in practice to have all three parties running on the same machine, as I usually do during the development stage, since from both XMPP and SOAP point of view these are different nodes and their actual physical location is just a matter of DNS resolution.

This application does not represent any revolutionary break-through, but

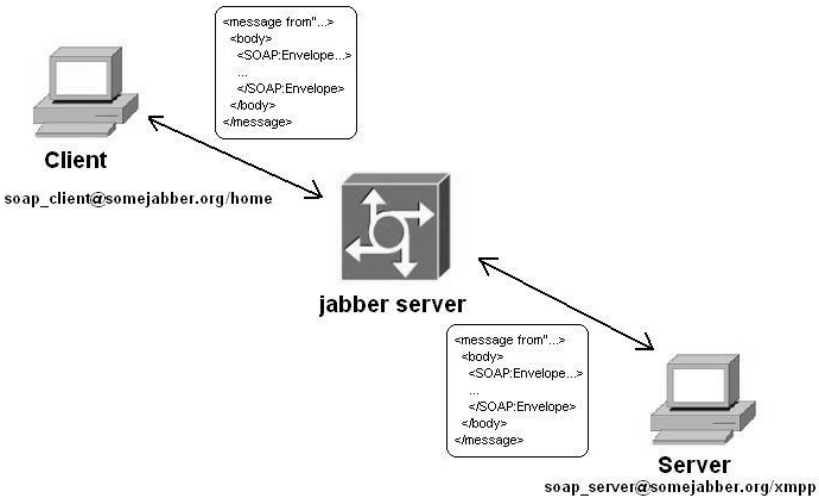

Fig. 3. Use Case 1 (simple)

rather serves the purpose of a simple "Hello World" type of example of how basic SOAP over XMPP works. Despite its simplicity, it still has several advantages when implemented over XMPP, compared to a classic approach over

- ✓ Since the two nodes are registered to the XMPP server, it takes care of login/authentication details, so the programmer does not have to do that at application level.
- ✓ The XMPP server also knows that these two nodes have authorized each other, so it can establish a basic publish-subscribe service between them. This actually results in queuing all the missed requests and responses (up to a predefined limit) and delivering them to their corresponding recipient nodes when they become live again (like getting missed messages on ICQ when you log in). Of course, the SOAP nodes may decide to totally discard these calls/responses if they want to or the server can be configured not to store them.

In order to be truly objective, we should note that the provided example has a certain disadvantage when implemented over XMPP:

- ↯ For a trivial WS call, not requiring authentication or message queuing, it is redundant to use XMPP, since there is a communication overhead for connecting to the jabber server and exchanging presence information

between the XMPP nodes and the server.

The case is demonstrated by the nusoap_currency_client_simple.php and xmpp_currency_server_simple.php scripts in the code archive.

II. Use Case 2 – Multi-User Chat. Extending the basic idea from the first example, let's assume that exchange rate information can be obtained by multiple clients from multiple servers, which post information regarding the current exchange rates at some intervals of time (may be various). The client software needs to keep track of this information when live, so that it can update a local exchange rates table. This time for the purpose of the mentioned information exchange there is a designated MUC room, set up on a conference service node at the XMPP server. Server bots connect to the server, join the room, and post current exchange rates info. WS client bots can join the room and "listen" for updates (Fig. 4):

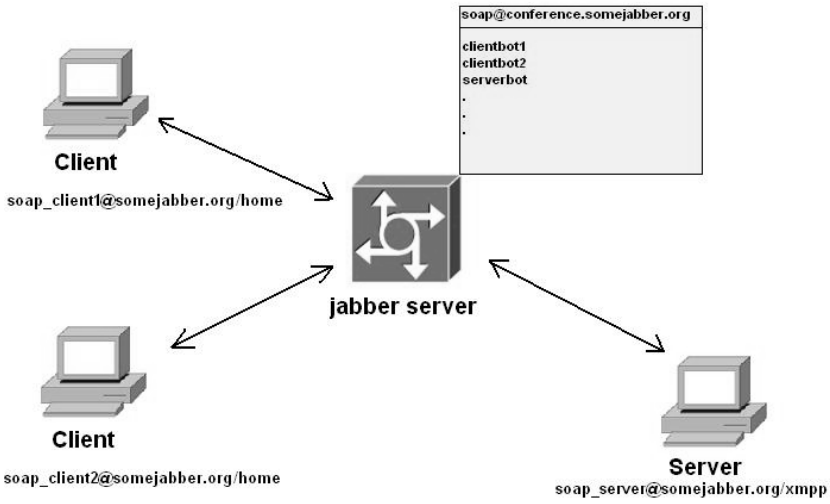

Fig. 4. Use Case 2 (MUC)

It should be noted that the node posting exchange rate info is not actually a SOAP server. It uses the client class, since it is the active party which makes the calls without having to wait for an answer. The users of this information are not classic SOAP clients either—they are XMPP bots, which simply connect to the designated MUC room at the conference server, listen to the broadcasted messages and use them selectively. In scenarios like this it is also applicable to use lightweight protocols, such as JSON, to encode the data, since most of the

SOAP envelope features are not used at all. This can make the presented WS solution even more efficient, eliminating some extra XML overhead, thus reducing the total volume of data transferred.

The clients of this service, once joined to the group chat session, receive all posted messages from all the other participants, as if they had been sent to them, so then it is up to the business logic of the application to set filtering rules, upon which the significant posts are determined.

So what advantages would such a WS solution have?

- ✓ It may be useful for distributed web applications where one or more nodes post notifications/messages regarding some change of state to a group of listening nodes, joined in a common MUC session.
- ✓ It is useful for one-to-many / many-to-one / many-to-many types of solutions, some of which can hardly be implemented over HTTP.
- ✓ Fits the SOA model.
- ✓ As in the first case, the XMPP server takes care of authentication details.

A disadvantage of this model would be:

- ↾ All broadcasted messages are sent to all participants in the MUC room, so even if they decide to ignore some of them they will still receive them all, which involves some network (and processing) overhead. This problem is addressed in the following (third) use case.

The case is demonstrated by the nusoap_currency_client_muc.php script in the code archive.

III. Use Case 3 – Publish/Subscribe (pubsub).

Let's picture a modified version of the second use case, in which the client applications need not to listen to all the updates all the time (and parse them, and filter them), but decide to subscribe to one or more preferable currency exchange info servers and get notified only when a certain event (like a change in rate higher than x percents) actually occurs. Even better, due to the hierarchical nature of the publish/subscribe service, it is possible to classify pubsub nodes into category trees with the different branches corresponding to categories and subcategories, and the leaves representing the final pubsub nodes to which the WS servers publish. A client subscribed to a given category would receive event notifications from all the nodes/leaves belonging to this category and all its subcategories recursively [10]. The graphic below (Fig. 5) illustrates the idea:

An exemplary use of the hierarchical pubsub system in our case[2] would

[2]The given hierarchical example is not implemented in the scripts from code samples below,

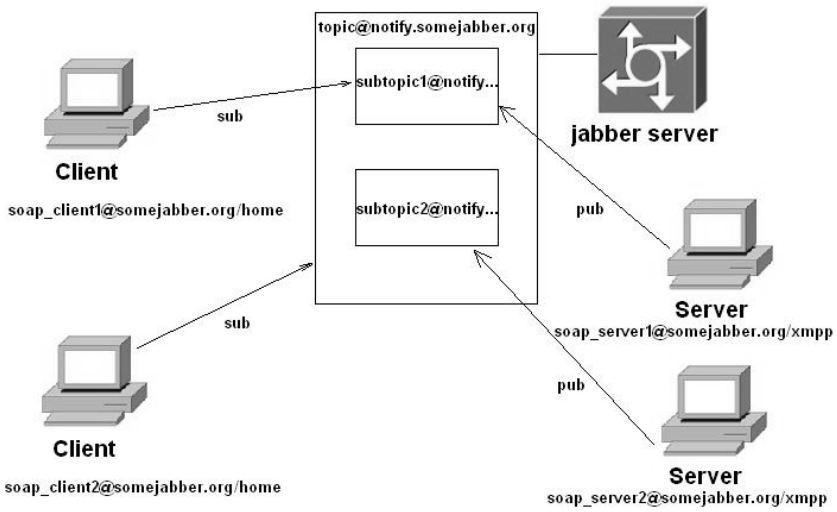

Fig. 5. Use Case 3 (pubsub)

be to categorize the different publishing servers geographically by continent (level 1), country (level 2), and bank (final leaves), allowing for client nodes to subscribe to:

a) given final node—bank server (e.g., BNB's currency exchange info service);
b) given country (e.g., Bulgaria, receiving notifications from all Bulgarian bank servers);
c) continent (e.g., Europe, receiving notifications from all European bank serves).

Such category trees allow clients to perform filtering at subscription level, effectively avoiding the communication overhead, the need for filtering at their side, and the need for having to manage multiple subscriptions if more info is needed.

The pubsub approach pushes web services a big step forward, compared to where HTTP-based web services stand. The publish/subscribe mechanism is a very powerful tool for building event-driven web applications, since it directly relates to the event-based model of behavior they employ. To achieve this over HTTP it is necessary to use the so-called polling mechanism. This means that

trying to keep the sample code as clean, short, and easy to understand as possible.

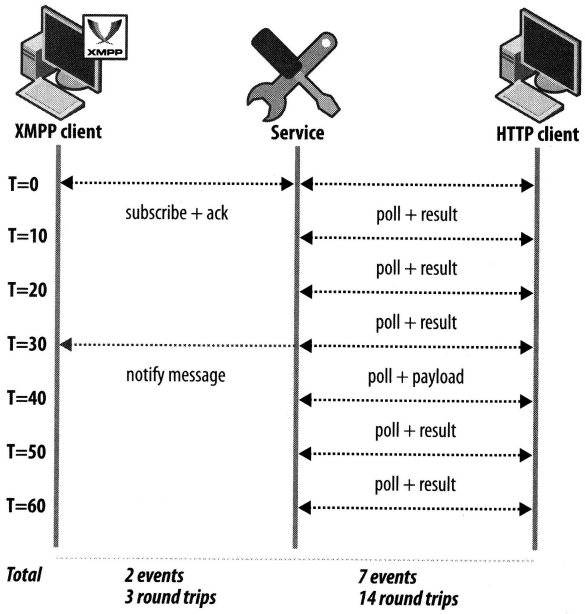

Fig. 6. XMPP pubsub vs. HTTP polling [10]

the client would need to connect to the HTTP server running the WS at certain intervals of time to check whether the event it is expecting has occurred, since the server has no means for feedback, once the short-lived HTTP session is closed. This may be OK when events happen frequently enough to make the ratio between the number of requests with a void/negative response and the ones with a positive one (after the event has occurred) small enough. In this case the polling overhead is small. However, if the events do not happen so frequently or if the application requires real-time behavior (immediate action after the event has taken place, without the lag till the next polling) then the polling model is inadequate, or at least inefficient. The graphic accompanying the current paragraph demonstrates the differences between the two approaches [10].

Once subscribed to the target pubsub node, all the client has to do is connect to the XMPP server and listen for incoming messages. Certainly some filtering needs to be added to distinguish the source of information, in case the client is subscribed to more then one pubsub node or other incoming communication is expected too.

The basic anatomy of a pubsub notification consists of an event part and a payload, containing the actual data posted. Some other great features of the

pubsub service worth mentioning are the possibility to generate different (custom-named) events, the possibility to receive the event info without the payload, and to retrieve the payload at some later moment from the XMPP server [10]. These enable the WS XMPP nodes with the following options:

- WS Server nodes may publish information under different events to the same pubsub node, allowing for diversification of the provided service
- Clients may choose to process only certain events from a given pubsub node, not all of them (efficiency, organization)
- Clients may choose not to receive the information payload for certain events at all, but register their occuring (save traffic overhead)
- Clients may request from the XMPP server the payloads of previous events at some later moment, provided that the server is configured to store them.

The ideas and techniques demonstrated in this third use case, accompanied by the sample scripts, demonstrate the following advantages of the approach:

✓ Event-driven communication over XMPP can be implemented in a more efficient and deliberate fashion than over HTTP, due to its native support in XMPP.

✓ Communication overhead can be reduced even further by distinguishing events and choosing not to receive payloads immediately, bur rather request them from the XMPP server at later time.

✓ Pubsub node hierarchies represent a great tool for organization and efficient grouping/aggregation of event with no equivalent in other middleware protocol.

✓ The XMPP pubsub service is excellent for building distributed real-time applications, driven by web services. This is very useful for implementing cloud-based web applications.

What disadvantages might this model have?

↯ The only disadvantage that may be considered is the necessity for the client to send presence information to the XMPP server when online, which brings about some network overhead. However, the very efficient fashion in which event-driven communication is implemented over XMPP pubsub definitely compensates for this disadvantage, at least in most cases.

The case is demonstrated by the nusoap_currency_client_pubsub.php and nusoap_currency_server_pubsub.php scripts in the code archive.

5. Conclusions. From the facts, comparisons, and practical examples given in this paper it can be deduced that the XMPP protocol with its numerous built-in features and extensions is more adequate than HTTP to serve as communication middleware for applications employing web services. This is so because of the limitations that HTTP imposes on implementing contemporary web applications, which employ the concepts of event-based communication, asynchronous calls, distributed processing (cloud computing), etc. Unlike HTTP, the XMPP approach addresses these issues, providing built-in support for such modes of communication. Moreover, this flexible protocol opens many new possibilities for extending the abilities of web services in underdeveloped or underused areas, such as: service discovery, value-added chains, modeling and implementing business processes, real-time applications, and others.

The current state of development of the web, having rich client interfaces, AJAX-driven web sites, and cloud computing, pushes the boundaries of the classic HTTP client-server communication model. Existing technologies are being employed to the full extent of their functionality in order to implement modern dynamic web applications with the look and feel of local software applications, but without the need to download such, effectively using the web browser as a universal interface. A great deal of the complexity in these "Web 2.0" approaches is due to the fact that software developers are stuck with users using many different (old and new) browsers, which are only fit to "understand" HTML, Java/ECMA Script, and occasionally Flash, constrained to communicate over HTTP. This is a process of "squeezing out" the maximum out of these limited, but widely supported existing technologies. However, the case doesn't have to be the same with web services, where there is no browser, hence there is no protocol or language barrier to keep in mind. Nevertheless, it has become a custom for most web services to be implemented over HTTP for simplicity or habit.

It would be unwise not to consider all the additional features and services that modern XML-based protocols have to offer, compared with HTTP. The obstacles in the path of modernizing web services are far less than the ones faced by web site developers when trying to improve web user interfaces, for example. This holds true not only for building in-house distributed web applications over XMPP, but also for public uses. In my opinion over time more and more individuals and organizations will choose to employ XMPP for conducting web services. Some of them will contribute their code libraries, thus sharing their approaches with the other in an open-source manner, effectively creating an environment for the global change over to richer, more flexible and effective web services to take place.

Abbreviations dictionary

API	– Application Programming Interface
AJAX	– Asynchronous JavaScript and XML
DNS	– Domain Name Service (the protocol)
DNS-SD	– Service Discovery over DNS
HTML	– Hypertext Mark-Up Language
HTTP	– Hypertext Transfer Protocol
HTTPS	– Hypertext Transfer Protocol Secure
IETF	– Internet Engineering Task Force
MIME	– Multipurpose Internet Mail Extensions
MUC	– Multi-User Chat
SASL	– Simple Authentication and Security Layer
SOA	– Service-Oriented Architecture
SOAP	– Simple Object Access Protocol
SSL	– Secure Socket Layer
TCP	– Transmission Control Protocol
TLS	– Transport Layer Security
UDDI	– Universal Description Discovery and Integration
VoIP	– Voice over IP
W3C	– World Wide Web Consortium
WS	– Web Service(s)
WS-BPEL	– Web Services Business Process Execution Language
WS-CDL	– Web Services Choreography Description Language
WSDL	– Web Service Definition Language
WWW	– World Wide Web
XML	– Extensible Markup Language
XMPP	– Extensible Messaging and Presence Protocol
XSS	– Cross-Side Scripting (popular JavaScript-based attack, aiming at stealing session ID)

REFERENCES

[1] Apache HTTP Server Documentation, *Authentication, Authorization and Access Control* `http://httpd.apache.org/docs/2.0/howto/auth.html`, 06.08.2010

[2] Barry D. K. Business Process Execution Language (BPEL). `http://www.service-architecture.com/web-services/articles/`

business_process_execution_language_for_web_services_bpel4ws.html, 16.11.2010

[3] CISCO, Internetworking Technology Handbook. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/Intro-to-Internet.html #wp1020627, 16.11.2010

[4] Crockford D. The application/json Media Type for JavaScript Object Notation (JSON) (RFC 4627), 07.2006. http://www.ietf.org/rfc/rfc4627.txt?number=4627, 16.11.2010

[5] Forno F., P. Saint-Andre. XEP-0072: SOAP Over XMPP, 14.12.2005 http://xmpp.org/extensions/xep-0072.html, 14.08.2010

[6] MatriX developer tutorial 05.11.2010 http://www.ag-software.de/matrix-xmpp-sdk/matrix-developer-tutorial/, 25.11.2010

[7] OASIS. UDDI Specifications TC – Committee Specifications. http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm, 17.11.2010

[8] OASIS. Web Services Dynamic Discovery (WS-Discovery) Version 1.1, 01.06.2009 http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html, 17.11.2010

[9] Oracle. TCP/IP Network Administration Guide. http://docs.sun.com/app/docs/doc/801-6632/6i10cirge?a=view, 16.11.2010

[10] Saint-Andre P., K. Smith, R. Tronçon. XMPP The Definitive Guide O'Reilly. First Edition, April 2005.

[11] Tilkov S. A Brief Introduction to REST. 10.12.2007 http://www.infoq.com/articles/rest-introduction, 17.11.2010

[12] W3C. SOAP Version 1.2 Part 0: Primer. Second Edition 27.04.2007 http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, 06.08.2010

[13] W3C. Web Services Addressing 1.0 – SOAP Binding, 09.05.2009 http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/, 17.11.2010

[14] W3C. Web Services Choreography Description Language Version 1.0, 09.11.2005 http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/, 17.11.2010

[15] Wagener J., O. Spjuth, E. L.Willighagen, J. Wikberg. XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services, 04.09.2009 http://www.biomedcentral.com/1471-2105/10/279

[16] XMPP Standards Foundation, About XMPP. http://xmpp.org/about-xmpp/, 16.11.2010

[17] XMPP Standards Foundation. XMPP Software: Libraries. http://xmpp.org/software/libraries.shtml, 15.08.2010

[18] xws4j, XMPP Web Services for Java. http://xws4j.sourceforge.net/ 25.11.2010

Mihail D. Irintchev
Institute of Information and
Communication Technologies
Bulgarian Academy of Sciences
1113 Sofia, Bulgaria
e-mail: mihail@irintchev.com

Received August 27, 2010
Final Accepted January 27, 2011