Interoperability and Performance of Web Service Implementation Technologies

A Comparison of Implementation Approaches and a Benchmark for Web Service Frameworks

Stephan Cornelius Arndt Sebastian Engel Felix Gündling
Fachbereich Informatik, Technische Universität Darmstadt, Darmstadt, Germany

Abstract

In this paper we analyze and compare web services implementation technologies with respect to their interoperability and performance. We draw attention to the compatibility of the different approaches *Contract-First* and *Code-First*. Furthermore we compare different web service frameworks regarding their performance We concentrate on the *Apache Axis2*, *Apache CXF*, *GlassFish Metro*, *Windows Communication Foundation* and *gSOAP* web service frameworks (each with SOAP endpoint usage) and the *PHP* SOAP implementation using a simple group management example application.

Keywords Interoperability, Performance, Code-First, Contract-First

1. Introduction

As web services became more and more popular and as they are employed in a variety of domains these days, it turned out that there are especially two web service properties of great significance: interoperability and performance. Before going on it is necessary to understand the meaning of these terms. Therefore we want to have a look at their definitions:

- **Interoperability:** The *Institute of Electrical and Electronics Engineers (IEEE)* defines interoperability generally as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" [1] while James A. O'Brien and George M. Marakas describe it more detailed as "being able to accomplish end-user applications using different types of computer systems, operating systems, and application software, interconnected by different types of local and wide area networks" [2].
- **Performance:** Performance in the context of web service implementation technologies means not only one but two things. On the one hand it represents the measurement execution time of a web service call. On the other hand the term involves the CPU usage and the memory requirements of a web service framework.

The two mentioned characteristics are especially relevant in the context of large enterprises and lines of business. In this environment, different systems of varying platforms and programming languages interact with each other which is the reason for interoperability to be essential. As enterprises have economic interests web services, beyond that, have to be high-performance.

In this paper we want to discuss influences on the interoperability of a web service and test the web service frameworks

- Apache Axis 2,
- Apache CXF,
- GlassFish Metro,
- Windows Communication Foundation,
- gSOAP
- and the PHP SOAP implementation

with regard to their performance.

Although interoperability is one of the basic goals that should be achieved by the usage of web service technologies, web services, however, often are not interoperable. In many cases a situational suboptimal implementation approach used by the developer(s) is the reason for a web service not being interoperable. Due to this we want to consider approaches in terms of interoperability. What are the reasons being responsible for an approach to provoke problems with interoperability? In section 2 we would like to overview the approaches and try to answer this question.

Furthermore we will analyze the web service frameworks mentioned above in section 3 with the goal of doing a benchmark test. For this purpose and for the reason of comparing web services in practice regarding their interoperability we implement a small example application in different frameworks. This example application deals with the managment of persons and groups. There will be the possibilites to create and delete persons and groups, to add a person to and remove it from a group as well as the chance to ask for informations about a specific person or a specific group. To do a meaningful benchmark test the reviewed frameworks are of different programming languages. The named web service frameworks are then tested with regard to their average response time per request, their CPU and their RAM usage. In addition, an interoperability test is used to determine whether the created clients and servers are interoperable.

In section 4 we want to have a look at related work that may be of interest. Considering other work we intent to bring the results gained in section 3 into relation with existing expertise.

Finally, in section 5 we want to give a conclusion of our work.

2. Approaches to develop a web service

There are two main approaches to develop a web service: *Code-First* and *Contract-First* [3]. In this section we would like to discuss these approaches. We want to point out their characteristics, balance pros and cons, compare the approaches and finally come to a small conclusion which method to use in which situation.

While *WSDL* [4] is an accepted standard to describe the functions, data, datatypes and exchange-protocols of a *SOAP* web ser-

vice the equivalent for *REST* web services, *WADL* [5], is not that popular (REST does not need a description document). Due to this we will only discuss the mentioned approaches in the context of deploying a SOAP web service.

2.1 Code-First-Approach

The *Code-First-Approach* is characterized by the implementation as the fundament for the whole process of development. Due to this it is also called *Implementation-First*. Programming the functionality is the first step in developing the web service. Subsequently a generator is used to automatically extract a *WSDL* description out of the written source code. Using this strategy *apparently* decreases the effort of developing as manually creation of *WSDL* descriptions is complex. In a third step the generated *WSDL* description is taken by another generator to produce helpers that can be used by client applications to easily invoke the implemented web service.

1. Implement functionality of the web service
2. Use generator to extract *WSDL* description out of source code
3. Use another generator to create helpers for client applications out of the *WSDL* description

Figure 1. Approach *Code-First*

This way of developing a web service is commonly used as it enables re-use of existing code while it does not require the reimplementation of the application's logic. Thus the time-to-market is reduced significantly compared to the time needed when using *Contract-First*. Obviously it is the most frequently demonstrated and the best documented approach. Developing non-complex services works very well which has the consequence that especially beginners succeed in executing the *Code-First-Approach*.

This approach seems to be easily feasible but often is not. Interoperability problems arise when developing more complex service applications and when being confronted with major reconstruction measures. Upon what do these problems base?

XML-based communication between web services and clients requires the existence of datatypes that all involved parties know how to handle - regardless of whether these parties run on different platforms or whether they are written in different programming languages. For this reason *XML Schema (W3C)* [6] provides a multitude of basic datatypes and the possibilities to extend and combine them with the goal of creating more complex datatypes. Beyond that there exist mappings to translate a datatype provided by *XML Schema* into a platform- or language-specific datatype and vice versa. The approach *Code-First* can lead to the usage of specific datatypes for which no mapping exists.

This usage of specific datatypes results in interoperability problems which are often not identified immediately. The reason for this is that many frameworks or *WSDL* generators nevertheless try to build XML datatypes out of these specific types for which a standard does not exist. Doing so they prevent the direct identification of the interoperability problem by the developer. As long as client and server base on the same framework the problem does not become noticeable. Web services, however, are generally intended to enable communication between parties running on any platform and written in any framework. If parties that both handle such specific datatypes in their very own way try to communicate with each other the problem becomes visible.

Another problem is caused by *WSDL* generators by not being interoperable themselves. The *Web Services Interoperability Organization (WS-I)* [7], an industry consortium, tried to counteract this problem by publishing the *Basic Profile (BP)* [8], a guideline for the use of web service technologies like *WSDL*. Chris Ferris from IBM (which is one of the founding members of the *WS-I*) characterizes this *Basic Profile* as a "description of what standards and technologies will be required for interoperability between web services implementations on different software and operating system platforms" [9]. Up-to-date frameworks generally implement the provided guidelines. But frameworks and generators that do not (completely) implement them put the interoperability at risk. Although web services conforming the *Basic Profile* reduce the potential of interoperability problems, the *Basic Profile* in general restricts developers and leads to more effort. This effort is worth it in large projects but maybe not in smaller ones.

To mention another point to keep in mind when deciding for one of the approaches developers should be aware of the sort of service they want to implement. A small prototype does not require a predefined XML Scheme. Here, the approach *Code-First* works well. The developer can fully concentrate on programming whereas a generator creates the *WSDL* description. This will lead to huge problems when the prototype becomes more complex or when it should be integrated into an existing project. Afterwards switching over to the approach *Contract-First* can be very difficult.

In general large-scale projects are projects of integration. Probably there already exists a quantity of standard XML schemes in the relative domain. If this is the case, it makes sense to make the own web service be compliant with the dataytypes provided by the given XML schemes. Using the approach *Code-First* in this context would definitely not lead to the goal of interoperability.

Let us have a look at the quality of the interface of a web service: It appeared that web service calls often were misinterpreted as Remote Procedure Calls (RPC) when using *Code-First*. This misunderstanding does not directly depend on *Code-First* but on the developer's level of knowledge. Nevertheless *Code-First* seems to provoke such a thinking. This results in an inefficient usage of the implemented web service: Each web service call induces costs. Interpreting web service calls as Remote Procedure Calls causes a multitude of calls each with a little amount of transmitted data.

Before going into the approach *Contract-First* we want to summarize the identified pros and cons of *Code-First*:

Advantages:

- Probably little effort when not dealing with large-scale projects
- No necessity to be well-schooled in *WSDL* and *XML Schema*
- Existence of a multitude of supporting tools
- Availability of lots of examples and documentations

Disadvantages:

- Most likely leads to problems with interoperability, especially when dealing with large-scale projects of integration of heterogeneous applications and platforms
- Seems to faciliate the misinterpretation of web service calls as Remote Procedure Calls which has a negative effect for the efficiency of the web service's usage

2.2 Contract-First-Approach

We are now familiar with the characteristics of the approach *Code-First*. As an alternative we want do consider the approach *Contract-First*. In contrast to the procedure when using *Code-First* developers at first define the web service's interface by creating a *WSDL* description. The implementation of the web service then occurs in a later step. Due to this *Contract-First* is also called *WSDL-First*.

When composing the *WSDL* description developers limit theirself to only use basic datatypes provided by *XML Schema* and datatypes that can be created out of these given types by also pro-

vided possibilities of combining and extending them. Acting this way a developer minimizes the risk of getting into trouble with interoperability. Besides the definition of all datatypes which the service is supposed to work with the first step moreover includes the specification of messages that will be sent between clients and the service or vice versa. To avoid confusion it is convenient to distribute all these definitions over several schemes.

Then the actual *WSDL* description of the web service's interface is written. Schemes that define the required datatypes and that specify the messages will be imported into this description. To make the following implementation of the service easier generators can be used to create code or code skeletons in the desired programming language for clients and services out of this *WSDL* description.

1. Create schemes with datatypes conforming *XML Schema* and with specifications of messages
2. Import written schemes into actual *WSDL* description
3. Use generator to produce code or code skeletons out of the *WSDL* description

Figure 2. Approach *Contract-First*

Only using datatypes that are covered by *XML Schema* has the consequence that the occurrence of problems caused by incompatible datatypes becomes improbable. In addition to that a developer is not forced to trust in generators to create *WSDL* descriptions that are cosistent with the *Basic Profile* provided by the *WS-I*. Nevertheless the self-written *WSDL* description should be analyzed in terms of conformity. For this purpose the *WS-I* provides a tool that checks whether the description fulfills the *Basic Profile*.

While the use of *Contract-First* results in a benefit of a (probably) interoperable web service it unfortunately involves some negative aspects. Many of the existing examples and documentations about developing a web service refer to the approach *Code-First*. Furthermore most of the provided tools and IDE-supports are related to the approach that we identified to be a higher risk for problems with interoperability. Besides tools that should faciliate designing of *WSDL* descriptions often are not without fault which can be explained with the complexity of *XML Schema* and *WSDL*.

Using *Contract-First* has yet another disadvantage. Creating a *WSDL* description at first and not generating one out of an existing implementation requires developers to be well schooled in *XML Schema* and *WSDL*. Especially for beginners this is a con as it costs time and effort and so prevents a quick development of a web service.

In contrast to the approach *Code-First Contract-First* does not provoke the developer to misunderstand a web service call as a Remote Procedure Call. This misinterpretation is avoided by *Contract-First* requiring the developer to give thought to the interface of the web service to implement. This way, a non-efficient usage of the web service is prevented.

Before comparing the described approaches we want to summarize the characteristics of *Contract-First* as we did it for *Code-First*:

Advantages:

- Most likely leads to an interoperable web service which is indispensable when dealing with large-scale projects of integration of heterogeneous applications and platforms
- Seems to prevent the misinterpretation of web service calls as Remote Procedure Calls by forcing the developer to give a thought to the web service's interface

Disadvantages:

- Requires the developer to be an expert in *WSDL* and *XML Schema*
- Involves large effort
- Deficit of supporting tools
- Lack of examples and documentations

2.3 Comparison of the approaches

In this subsection we want to juxtapose and directly compare the characteristics of the described approaches. For this purpose we choose in each case a property to consider and analyse the approaches in terms of this chosen property.

The approaches will be compared by consideration of the following properties:

- Interoperability of the web service to be implemented
- Effort required for implementation
- Availability of tools that support the developer
- Availability of documentations and examples
- Quality of the web service's interface

Interoperability: At first, let us consider their probably most important attribute: Do the approaches enhance the interoperability of the web service that is to be implemented or do they prevent the service to be interoperable? This question can easily be answered: While *Code-First* massively increases the risk of getting into trouble with interoperability *Contract-First* enhances the interoperability. The interoperability depends on the datatypes that are used in the implementation. *Code-First* allows the use of specific datatypes that are not consistent with established standards. In contrast to this *Contract-First* prevents the usage of such datatypes and only permits the usage of standard conformable ones. This is done by charging the *WSDL* description to be consistent with the *Basic Profile* provided by the *WS-Interoperability Organization*.

Effort: Which one of the approaches requires less effort? This question can not be answered as easily as the previous one as it depends on the developer's level of knowledge and on the web service that shall be implemented. At first sight *Code-First* appears to be less costly than *Contract-First*: There exist a lot of tools to generate *WSDL* descriptions. Furthermore, developers do not have to be experts concerning *WSDL* and *XML Schema*. In contrast, using *Contract-First* requires a developer to be familiar with *WSDL* and *XML Schema*. In addition to that there are significantly less tools supporting the use of *Contract-First*. Nevertheless there are reasons to suppose that *Contract-First* is the approach being less costly: Not leading to problems with interoperability is one of the main advantages of *Contract-First*. In case of a bigger project of integration *Code-First*, however, leads to those interoperability-problems. Doing so it causes a lot of trouble to get these problems under control. Apparently, *Contract-First* is worth the effort. *Code-First* initially seems to be easier but most likely is not when dealing with larger projects.

Supporting tools: For which of the approaches more and better tools are offered to support the developer? Surprisingly, there exists a multitude of tools supporting the approach *Code-First* while there are only a few instruments supporting *Contract-First*. Additionaly, there are lots of generators that are able to translate the code of the web service into a *WSDL* description. Besides, there are lots of generators which produce code for client applications out of the generated *WSDL* description. In contrast to this there are significantly less tools available for assisting the developer to accomplish the creation of *WSDL* descriptions. This unequal distribution is curious as web service technologies were introduced to negotiate in-

teroperability problems for which *Contract-First* obviously is the more qualified approach.

Documentations and examples: Which of the approaches is better documented? For which of them more examples exist? Similar to the availability of supporting tools there exists a multitude of documentations and examples for the approach *Code-First*. In contrast to this there are significantly less documentations and examples available for *Contract-First*.

Quality of web service's interface: Does the implementation approach have an influence on the quality of the constructed web service's interface? As it is possible to create identical web services with both of the approaches we can negate that there is a direct influence. First of all the quality of a web service's interface depends on the developer's level of education and experience. Nevertheless there is an indirect influence: *Code-First* can make developers misunderstand a web service call as a Remote Procedure Call. This misunderstanding has a negative impact on the efficiency of the web service's usage since it leads to a multitude of web service calls each with a small amount of transmitted data. This behaviour is very expensive. Maximizing the amount of data transmitted with each web service call would be a better strategy regarding the efficiency. In contrast to *Code-First* the approach *Contract-First* does not lead to this misunderstanding. In fact it requires the developer to give thought to the interface of the service to be implemented. Doing so it most likely prevents the developer to misinterpret a web service call as a Remote Procedure Call.

The following table summarizes the gained insights:

Property	*Code-First*	*Contract-First*
Interoperability of the web service to be implemented	Probably leads to web services not being interoperable	Minimizes risk of getting into trouble with interoperability
Effort required for implementation	Requires less effort at first (required effort can increase when being forced to repair occuring problems with interoperability)	Requires much effort
Availability of tools that support the developer	Multitude of supporting tools	Deficit of supporting tools
Availability of documentations and examples	Multitude of documentations and examples	Deficit of documentations and examples
Quality of the web service's interface	Can lead to the misunderstanding of a web service call as a Remote Procedure Call	Most likely prevents the misunderstanding of a web service call as a Remote Procedure Call

Table 1. Comparison of *Code-* and *Contract-First*

The comparison of the approaches' characteristics show us that there is no general-purpose solution. Which approach to use depends on the developer's level of knowledge and on the desired service. In the following we want to name scenarios and the approach to use in the respective scenario.

Scenario 1: A developer wants to quickly build a running system that is not too complex and only intends to create a solution for a concrete problem. Such a developer may decide for *Code-First*. If being interested in a pragmatic method and not in spending to much time and effort in extensive scheduling this is probably the preferred approach. There is no need for a deeper comprehension of *XML Schema* and *WSDL*. Instead the developer can use a generator to create *WSDL* descriptions. However, there is a risk of getting into trouble with interoperability, but these problems will not get noticeable as long as client and service base on the same framework.

Scenario 2: A developer looks for a web service solution for a (large) project of integration - integration of heterogeneous applications and platforms. In this case interoperability is not only desired - it is indispensable! Due to this the only sense-making approach is *Contract-First*. In projects like this the effort related with the design of the *WSDL* description will pay off and spare big problems with interoperability.

Scenario 3: A developer wants to build a web service as a part of a bigger project. In this case the web service has to be conform to existing parts of the project and to those which will be developed in future. Moreover it is beneficial to implement a web service that is accordant to other services of the same domain, a service whose XML schemes conform the domain's existing standards. As in scenario 2 interoperability is essential and therefore *Contract-First* is the approach the developer should use.

Scenario 4: A developer wants to make an existing application available to others. Using *Code-First* would be a quick-and-dirty solution, the easiest and quickest way to do this. This is one reason for *Code-First* being that popular. But when the web service, later on, shall be made a part of a large project of heterogenous applications and platforms, not having invested in good code and design will turn out as the wrong decision. The developer, maybe, will have been promoted to a higher position in the meantime and other developers will be confronted with the problems. Because of this *Contract-First* should be used if there is the possiblity that the web service will be a part of a larger project some day and if the business restraints let the developer invest time and effort to implement an interoperable solution.

Web service technologies were introduced to enable exchange of data between parties of any platform and any programming language. The majority of web service projects are projects of integration that require web services to be interoperable. In general *Contract-First* is the approach of choice if a developer is interested in avoiding problems with interoperability in advance. A good design is the base of a successful implementation of a web service that is extendable, enables the integration of heterogeneous applications and platforms and is conform to existing standards of its domain.

In practice there are developers using an approach that can be seen as a "meet-in-the-middle" of *Code-* and *Contract-First*. At first these developers implement the service class. Then a generator is used to create a *WSDL* description out of this implementation. Knowing this *Code-First-approach* to cause problems with interoperability the developers afterwards manually edit the generated *WSDL* description. By doing so they try to transform the started *Code-First-approach* into *Contract-First*. This strategy, however, is not recommended as it is not efficient. If the application to be implemented as a web service is not suitable for being developed using the *Code-First-approach Contract-First* is the only sense making solution and will be worth the effort.

3. Comparison of web service frameworks

In this section we would like to analyze web service frameworks in terms of their performance. We develop an example application and create web services of different frameworks implementing this example application. These web services, then, are tested regarding their performance. For this purpose we develop a client that executes web service calls to strain the implemented web services. It is the goal to compare the frameworks by acquiring a benchmark for them.

In addition to the consideration of the performance of the web services we analyze their interoperability. For this reason we perform a "cross-test" which concerns two of the web service implementations. First a client that is implemented by using the first of the two frameworks tries to communicate with the web service implemented by using the second framework. Afterwards a client that is implemented by using the second framework tries to communicate with the web service implemented in the first framework. If both experiments succeed the implementations can be seen to be interoperable.

We already learned that not *Code-First* but *Contract-First* is the reasonable approach to use when intending to achieve interoperability. Due to this and to make a comparison of the implementations possible we implement the web services using *Contract-First*.

3.1 The web service application

Before developing web services there has to be an example application which is implemented by the web services then. The example application we decide on provides an interface for managing users in groups. Below we describe the application in detail:

We have two complex datatypes:

- A user consisting of a unique username and some simple additional attributes: first name, last name, age and gender. This way it would be possible to address the user for example in a welcome or leave email notification
  ("Welcome ${if gender "Mrs." else "Mr."} ${lastname} ...").
- A group that can be accessed using a unique group ID. One group can contain several users.

The web service will provide the following operations:

- Add a new user
- Delete a user
- Get information about a specified user
- Add a group
- Delete a group
- Get information about a specified group
- Add a user to a group
- Remove a user from a group

This service could be adopted by a business concern to add the functionality to manage users in groups without touching other existing web service implementations. But since we won't implement security features like encryption or authentification which are out of scope of this paper's topic, the created services are not ready to be deployed in an enterprise environment.

Why do we decide on the application described above? As it is our intention amongst others to analyze the interoperability of the web services to implement, the application has to contain complex datatypes like the mentioned user and group types with the group containing a collection of users. If the application only contained simple datatypes, issuing a firm statement about the services interoperability would not be possible. Choosing a rather intricate application of web service technology we hope to get test results that can be applied to web services that are build for productive use.

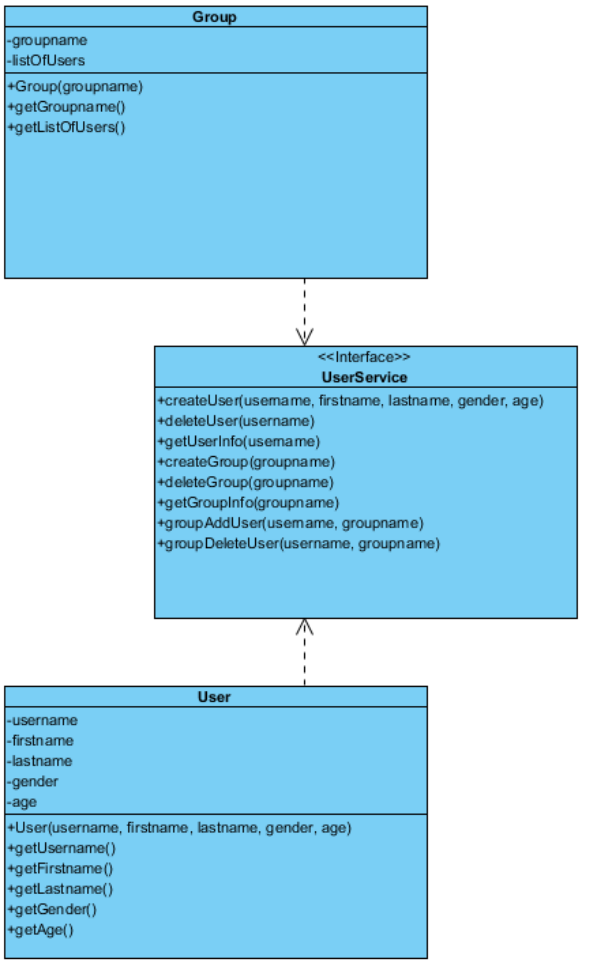

Figure 3. The example application

The web services have to deserialize the incoming SOAP XML to internal data structures for processing. Furthermore the services have to convert the complex datatypes to XML when creating a response. This way it is possible to see how the different frameworks will perform deserializing and serializing of more complex datatypes from and to XML. So with this example of web service usage (the user group service) it should be possible to reveal the bottle necks of the different frameworks better than with a too simple web service.

3.2 The web service frameworks

In the following we list and describe the different web service frameworks we use to implement our example application. The descriptions don't contain a complete feature listing but shall list the "unique selling points" of each framework with the goal of naming our motivation to implement it. So if a framework description lists a specific feature, this does not imply that another framework whose description does not mention this feature does not provide it, too.

One may notice that three out of six web service frameworks are Java frameworks. The reason for this is that the field of web service programming (especially in the environment of enterprise web

services) is dominated by Java. There are only very few well documented SOAP web service frameworks written in other progamming languages. The only major non-Java web service framework in our selection is the Windows Communication Foundation.

3.2.1 Description of the chosen frameworks

Apache Axis 2 (Java): Apache Axis 2 is hosted at the Apache Software Foundation. It is the successor of one of the most widely used web service frameworks, Apache Axis and calls itself "the third generation Web services engine" which is "more efficient, more modular and more XML-oriented than its predecessor Apache Axis" [12]. Axis2 is a complete rewrite where developers focused on performance, robustness, efficiency and reliability. These goals (especially the high performance) are accomplished by building on Apache AXIOM, "a new high performant, pull-based XML object model" [13], which enables Axis2 to parse only information that is relevant to respond to a request. This behavior should result in a lower memory and CPU processing time footprint as well as shorter response times. Axis2 supports both, synchronous and asynchronous communication over several protocols such as HTTP, SMTP/POP3, SMS or UDP [24]. This makes the Axis2 web service framework very flexible and so it can be applied to many use cases.

Apache CXF (Java): Apache CXF is also hosted at the Apache Software Foundation. It is a merge of two major web service frameworks: the Celtix framework (which is now hosted at Object Web) and the XFire framework, developed at Codehaus [14, 15]. It focuses on high performance, extensibility and intuitive usage. It supports a variety of JSR (Java Specification Request) standards such as JAX-WS (Java API for XML-Based Web Services) or SAAJ (SOAP with Attachments API for Java). Furthermore it supports several web service specifications concerning quality of service (WS-Reliable Messaging), metadata (WS-Policy, WSDL 1.1), communication security (WS-Security, WS-SecurityPolicy and WS-SecureConversation) and messaging support (WS-Addressing, MTOM and XOP, SOAP 1.1 and SOAP 1.2). Like the Apache Axis2 framework Apache CXF also supports many transport protocols like HTTP, SMTP/POP3 or TCP. The CXF framework also has support for databindings like JAXB 2.x enabling the developer to use Java classes that are mapped to XML representations. Additionally CXF provides a extensibility API to enable more message formats like CORBA (Common Object Request Broker Architecture). CXF applications can be deployed using different servers like Jetty, Tomcat, Spring-based containers, Java Enterprise Edition application servers like Apache Geronimo or Redhat JBoss and even a standalone version. The framework provides several programs to generate code from a WSDL file or to generate a WSDL file from code as well as validation programs [16]. Overall one can say that CXF provides all capabilities to successfully build large, fast and reliable enterprise web service applications fitting into a service oriented architecture.

GlassFish Metro (Java): The Metro web service framework is contained in Oracles Java web server reference implementation, GlassFish. The main development took place at Sun Microsystems which was acquired by Oracle in 2010 [19]. Its most prominent feature is the so called "Web Services Interoperability Technologies" (WSIT) [22] subsystem, which aims to provide interoperability between Metro services and services of one of the most popular non-Java frameworks, namely Microsoft's WCF. It implements many crucial WS-* specifications, such as WS-Security and WS-ReliableMessaging, and enables compatible usage of them when communicating with a service based on WCF. As such Microsoft calls WSIT the "best impl[ementation] of WS-* outside its own" [23]. Additionally Metro is capable of generating server- and client-side code if provided with a WSDL file, which has to be WS-I Basic Profile 1.1 conform. As far as encoding and transport are concerned, Metro offers the same variety of supported protocols as CXF and uses JAXB for databindings. As such the Metro framework ascribes highest importance to interoperability, especially with .NET services, and offers many features which most web service developers can directly utilise.

Windows Communication Foundation (.NET): The Windows Communication Foundation is Microsofts communication framework supporting (amongst others) SOAP and REST web service endpoints (clients as well as servers). It requires the .NET Framework and can be deployed standalone or on the Internet Information Services server. Services can be implemented either in C# or in Visual Basic. Its main focus lies on security, reliability and interoperability. WCF (Windows Communication Foundation) integrates very good with other Microsoft Technologies but can also be used to communicate with other web services frameworks like Glassfish Metro. It supports a variety of different modern industry standards (WS-*), formats (like XML and JSON - mainly used in REST services), protocols (for instance HTTP, TCP or MSMQ) and specifications enabling developers to build large scale services to be used in a Service-oriented architecture (SOA). If WCF is not suitable for a specific task it can be extended. Therefore it provides a number of entry points allowing the developer to customize the behavior of a service. [17]. So using Windows Communication Foundation in a Microsoft / Windows environment is definitely a good choice.

gSOAP (C/C++): gSOAP was introduced in 2002 but the framework is still actively maintained and developed. It supports ANSI C as well as mixed C/C++ application development. Its main focus lies on speed, reliability and flexibility. gSOAP is widely used in the industry, for example in the Cisco Unity Connection or the IBM DB2 Content Manager [20]. It also can be adopted to be used in embedded and mobile systems. It supports HTTP(S), TCP and UDP. Using plugins it can be extended to support virtually every transport protocol available. gSOAP can be deployed either standalone or by using web servers, for example Microsoft Internet Information Services, Apache_mod, CGI or FastCGI containers. Since development and testing took place since 2001, gSOAP is one of the most stable web services frameworks available. It also supports many standards such as WS-SecurityPolicy 1.2, WS-ReliableMessaging or WS-Addressing. gSOAP delivers programs to generate C or C++ code from a WSDL file [21]. When searching for C/C++ web services frameworks gSOAP seems to be the only framework providing such a broad support for industry standards. Overall gSOAP is a very good possibility to expose functionality (written in C/C++ code) to a SOA.

PHP: PHP provides support for SOAP web services (clients as well as servers) out of the box. SOAP support is integrated in the PHP programming language (since PHP 5). So there is no need to install any third party extension. Since PHP is very easy to learn and can be written very fast, writing a PHP web service can also be accomplished by beginners. Even experts can use it for rapid prototyping or to reduce the time to market (TTM) for simple services.

3.2.2 Feature comparison

As we now know about the frameworks that we chose to implement, we can compare them by examining which WS-* specifications they support. Thus we can tell beforehand which framework is usable in which context, e.g. if WS-Security is indispensable for an application, we can see which frameworks fullfil those specific needs. This comparison is interesting for developers as they do not need a web service with good performance that does not meet their

demands. Note that PHP does natively not support any of those specifications, so it is not included in the following table.

Specifications	Axis2	CXF	Metro	gSOAP	WCF
WS-Addressing	✓	✓	✓	✓	✓
WS-AtomicTransaction	✓		✓		✓
WS-BusinessActivity	✓				✓
WS-Coordination	✓		✓		✓
WS-Discovery				✓	✓
WS-Enumeration				✓	
WS-Eventing	✓			✓	✓
WS-MetadataExchange	✓	✓	✓	✓	✓
WS-Notification	✓	✓	✓		
WS-ReliableMessaging	✓	✓	✓	✓	✓
WS-Policy	✓	✓	✓	✓	✓
WS-SecureConversation	✓	✓	✓	✓	✓
WS-SecurityPolicy	✓	✓	✓	✓	✓
WS-Security	✓	✓	✓	✓	✓
WS-Trust	✓	✓	✓		✓
WS-Transfer	✓			✓	✓

Table 2. Supported WS-* Specifications

Additionally we may look upon the different supported protocols for message transporting, which might play a big role if interoperability with already existing systems is a decisive factor.

Protocol	Axis2	CXF	Metro	gSOAP	WCF
CORBA	✓	✓	✓	✓	
HTTP	✓	✓	✓	✓	✓
JMS	✓	✓	✓		
Jabber	✓	✓	✓		
SMTP/POP3	✓	✓	✓		
TCP	✓	✓	✓	✓	✓
UDP	✓	✓		✓	✓

Table 3. Supported Transport Protocols

3.3 Implementation

In the following part of this article we describe the implementation of the mentioned services. This mirrors our personal experience when implementing the user service described in section 3.1. The implementation of the user service core functionality in the three Java frameworks is the same.
We decided to use the default deployment for the services without applying special settings that can possibly improve the performance.
To ensure that all implementations provide the same functionality we tested them with JUnit tests using the Axis2 generated client. All implementations passed the tests.

Apache Axis2

Creating a web service with the Axis2 framework is a pretty straight-forward process, save for a few minor exceptions. When first taking a look at the plentiful documentation at the framework's website, one can easily see that all of its functionality is explained briefly and even enhanced with examples. As such we just follow the quickstart guide, but soon find that the documentation is not updated for the latest Axis2 version, e.g. most of the given filepaths simply do not exist in the Axis2 framework as it is stated in the documentation. Some paths are just minorly renamed while others have been moved to completely different locations. But nonetheless when searching the web for help, we find many people having the same difficulties and thus we can find answers with relative ease. As a matter of fact building a contract-first web service with Axis2 is as easy as utilising the "wsdl2java" script, which is shipped with the framework. Depending on whether we want a server- or client-side skeleton, we just have to call the script with different parameters. After implementing the needed functions in the server skeleton, we need to run an ant script, which also has been generated by the wsdl2java command. This bundles the generated java file to an ".aar"-file, a container file used for Axis2 services. After that the only thing left to do is moving the container file in the correct folder and the service is deployed when starting the server shipped with Axis2.

3.3.1 Apache CXF

CXF offers almost any feature a web service framework can have with the consequence that the documentation is very large. Almost all examples are built using Maven. This is a great feature for medium and large scale projects, but using it for building a simple user service would imply overhead from our point of view. So we implement the web service without using Maven. There are tools to generate the service skeleton from the WSDL contract and it is easy to fill in the business logic. The CXF binary distribution provides all tools and Java archive (JAR) libraries to develop and deploy a standalone web service. The script can also be used to generate client code which can be used very intuitively.

Glassfish Metro

For the implementation of the Metro web service, we need to take a rather difficult approach. This may be the case because Metro is much more aimed at corporate usage than minor projects like ours. Even though the documentation is rather detailed, we can not use its example for creating a contract-first service. The examples provided uses much more than just a WSDL-file for code generation. Thus we first generate Java files from our WSDL by using Metro's "wsimport" script and add the user service functionality by implementing the created interface with the methods specified in the WSDL. The Eclipse IDE provides a good integration of Glassfish development tools. So after installing the extensions required to connect the IDE to the Glassfish server, deploying the metro application was rather easy.

3.3.2 PHP

For the PHP implementation we use a "wsdl2php" script which has the location of the wsdl file as input parameter and creates the corresponding SOAP client code. The generated client is very simple to use. There are no scripts generating the server stubs. This is no problem because PHP provides an easy way to provide a class object as service implementation. The following code illustrates this approach:

```
class UserServicePortTypeImpl {
  public function CreateUser($userCreateRequest) {
    return new UserCreationResponse(create_user());
  }
  // ... and all other methods
  // described by the WSDL contract
}

$service = new UserServicePortTypeImpl();
$server = new SoapServer("userservice.wsdl");
$server->setObject($service);
$server->handle();
```

A disadvantage of PHP is that it is not possible to store a user list and a group list in memory because the script ends when the web service call is handled. This way we have to implement the PHP

service using a database. This must be taken into account when analyzing the results of the performance benchmark. The database connection is implemented with the PDO abstraction layer. This way the service is compatible to different database servers. For our tests we used a MySQL server to store the user and group information. We uses an Apache2 web server with a *mod_php* extension (compiled with SOAP support enabled). Overall we can say that implementing the PHP web service communication is very intuitive.

3.3.3 gSOAP

gSOAP provides programs to generate service stubs from a WSDL file, too. The first program ("wsdl2h") generates a header file from the WSDL input file. The second program ("soapcpp2") generates service or client code (depending on the given command line flags). gSOAP has a large documentation describing the different features and leading through the creation of web service servers and clients. gSOAP can be deployed standalone or using web servers. We decide to use the single-threaded approach described in the documentation as we experienced that due to mutex locking synchronizing data access (user and group information is only held in the heap memory) a multi-threaded version does not have a higher performance than a single-threaded one.

3.3.4 Windows Communication Foundation

We decide to implement the WCF web service in C# (the alternative would have been Visual Basic). Looking for helper programs to generate C# web service code from the WSDL contract we discover that Microsoft does not provide an official program that can generate server code from a WSDL contract. The program *WSCF.blue* [27] seems to be a widely used program that provides support for developing WCF web services utilizing the contract-first approach. This tool only works with the *Visual Studio Ultimate* edition, not with the free *Express* version. After replacing the stubs generated by *WSCF.blue* with our service logic, the standalone service can be started.

3.4 Tests

3.4.1 Performance test

The performance tests try to measure the performance of the different web services in the average response time per request. Furthermore we measure the heap (RAM) and the CPU usage of the different services.

Since we did not implement any special caching strategy, it is up to the service to cache request responses (using more heap memory) or not (using more CPU power - especially for information requests like the "Get information about a specified user"-request listing all groups).

We implemented the service on the same way with all web service frameworks. If the implementations had differed too much (for example by using very efficient datastructures in one service implementation and less efficient in the other one) the test results would have become useless.

During the tests all the web services are run on the servers, which are shipped with them. No additional software to enhance performance outside of those provided in the standard editions are used. This means Metro runs on Glassfish, CXF runs on Jetty, Axis2 on its own server and gSOAP's server is created like it is specified in its documentation [26].

The performance of the web service implementations are measured using the following test:

- Testing duration is one minute. During a test we send a fixed number of requests per second to the web service. The number of requests is incremented steadily until the service is uncapable of responding to all issued requests and starts discarding additional requests.
- At first we randomly create users and group and add users to groups. This part shows how the different web services perform in reading (deserializing) XML requests.
- Then we retrieve the user and group information. Here we see how creating and sending the XML responses (serializing internal datatypes to XML) works for the services and how long this takes.
- Finally the groups and users are deleted and every deletion request is answered with a response, so we can once again test operating time of the requests and speed of responses.

As such we can see how much load the web services generated by the different frameworks can take before collapsing. It also enables us to compare them regarding their response times under increasing load.

3.4.2 Interoperability test

The interoperability test tries to check whether the web services can communicate with clients generated from other frameworks using the same WSDL file. The application used for this test remains the same application used in the performance test (user group management service). So the server implementations can be reused for this tests. Here we only compare the different servers and clients on the base of user and group information requests, since those two contain the most complex datatypes of our user service application and if these work, the other requests would be trivial to handle for the servers and clients. Thus we can identify which framework generates clients based on the contract-first approach and as such is interoperable with the other frameworks.

3.5 Results

3.5.1 Results of the performance test

We realized the previously described perfomance test by running the servers of the frameworks except for WCF on a machine with the following specifications:

- CPU: Intel(R) Core(TM)2 Duo CPU P8600 @ 2.40GHz
- CPU Cache: 3072 KB Cache per core
- RAM: 4GB DDR-3 RAM
- OS: Ubuntu 11.10

For WCF we had to fall back to using another machine (AMD Phenom(tm) II X4 945 (3 CPUs), 12GB DDR-3 RAM, Windows 7) because it requires Windows for running the server. The main focus of these tests are average response time per request, CPU and RAM usage. Creation of the testcases is done using soapUI [28]. They are then bundled to one "load-test" and executed with loadUI [29]. As such one execution of the test suite in loadUI (one request) results in actual eight consecutive requests on the server (create user, create group, get user info, add user to group, get group info, remove user from group, delete group, delete user). This should be kept in mind when reading the upcoming evaluation of the performance test, in which one request means one loadUI request. So let us first have a look at the results concerning the average response time per request.

As we can see in figure 4 all of the frameworks except for PHP can handle sufficient number of requests without any major increases in response time for up to about 250 requests per second. At this point gSOAP can not handle incoming requests fast enough and a queue with untreated requests starts to form which in turn significantly increases the average response time. Once the queue reaches its maximum capacity (1000 requests) any further requests

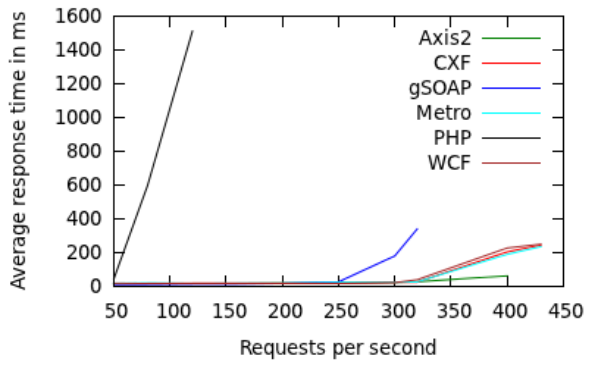

Figure 4. Average response time per requests

are discarded and we stop increasing the requests per second any further. Thus we can see that the three java frameworks all work equally well up to about 330 requests per second, when Axis2 is the only framework to not increase its response time. Now we can see that while Axis2 cannot handle as many requests as Metro and CXF while not succumbing under the increasing incoming requests, its request time is the only one which stays constantly low throughout the test case. On the other hand Metro and CXF can handle more requests than Axis2 with the price of an increasing response time when approaching their limits. The PHP implementation can only handle about 120 requests per second with majorly higher response times than the other frameworks. gSOAP can only keep up with the Java frameworks up to a certain point with equally good results, but then starts to fail rather quickly with more requests being issued. Thus we can conclude that up to now the Java frameworks are leading, with Axis2 having a more constant low response time overall, but not being able to process as many requests as Metro and CXF. So let us look at the CPU usage now.

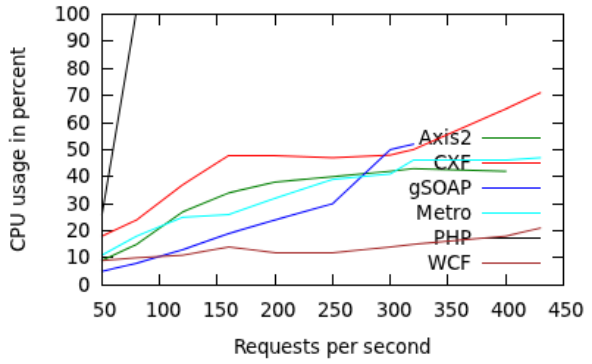

Figure 5. Average CPU usage

As we can see, PHP once again can not compete with the other frameworks with up to 4 times the CPU usage of the others at the time of succumbing. The Metro and Axis2 implementations are about equal while CXF is arguably worse throughout the tests. gSOAP on the other hand is clearly the least CPU straining framework during the tests up to the point of being overwhelmed by too many requests. But all four frameworks show about the same trend of CPU load under increasing a growing numer of requests - something which does not come unexpected, but still noteworthy because they don't increase exponentially like the PHP implementation, but steady linear increasements. As such we can conclude that while working under no pressure, gSOAP can respond as fast as the Java frameworks and use less CPU power, it might be better suited for less busy web services, not too big applications. But we need to keep in mind, that RAM is also often a decisive factor.

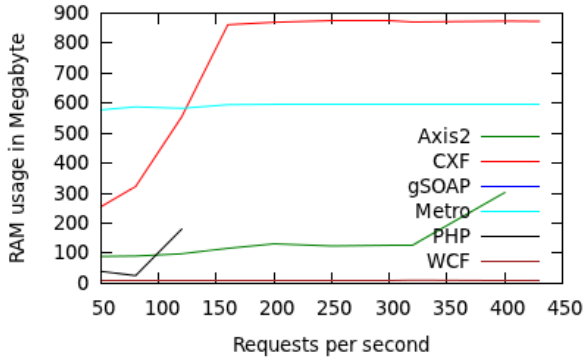

Figure 6. Average RAM usage

Although gSOAP's RAM usage is not observable in the above graphic, its line is directly above the x-axis with a steady 272 kilobytes of used RAM throughout all the tests. To put that into perspective the next best framework is Axis2 with about 90 megabytes of RAM at 300 requests per second. The next best would be Metro with a constant about 580 MB of RAM used and then CXF with by far the highest RAM usage at higher load. Also notable is the PHP implementation's low RAM usage which is in stark contrast to its other results.

Thus we come to the conclusion that even though it is possible to create web services with PHP rather easily it is nowhere near as feasible as the other specialized web service frameworks regarding performance. By the same token the single threaded gSOAP standalone implementation tested by us is not meant to be used in big corporate level applications because even though it works really well up to specific point - in fact better than the other frameworks - it just can not handle as many requests as the Java frameworks. Out of those all performed more or less equally good, with the exception of RAM usage where Axis2 is the clear leader. So we can come to the conclusion that Axis2 has a slight advantage in overall performance, with Metro and CXF not too far behind. gSOAP is usable but only to certain extent but really shines with its low RAM usage which makes it very applicable for mobile and embedded devices.

Overall one must say that our results can not be transferred directly to enterprise usage since we always use the default settings with the default (standalone) deployment. Especially CXF which is deployed using Jetty would probably not be used this way in the industry. Also gSOAP which is implemented using a single threaded approach would certainly perform better when being used to its full potential with multiple threads to use all CPU cores which are available in todays hardware. Nevertheless the results point out a trend of the performance of the web service frameworks.

3.5.2 Results of the interoperability test

For the interoperability test we decide to implement web service clients using the Axis2, WCF, CXF and SOAP4R (a SOAP framework for Ruby) and check whether the communication between these clients and the web services (already implemented for the performance comparison) works. The example application (and the corresponding WSDL contract) stays the same.

As we can see in table 4 the frameworks we tested here are all interoperable. All vendors generate code that is compliant with the WSDL file. Here we can see that interoperability problems can be avoided by applying the contract first approach - at least in our test case.

		Server						
		Axis2	WCF	CXF	Metro	PHP	gSOAP	SOAP4R
Client	Axis2	✓	✓	✓	✓	✓	✓	✓
	WCF	✓	✓	✓	✓	✓	✓	✓
	CXF	✓	✓	✓	✓	✓	✓	✓
	SOAP4R	✓	✓	✓	✓	✓	✓	✓

Table 4. Results of the interoperability test

4. Related work

When comparing our results to those of Steve Sosnoski in his article "Java web services: CXF performance comparison" [25] in which CXF, Axis2 and Metro are compared, we can observe that there are major differences between the two performance tests. While we have come to the conclusion that Axis2 has the fastest response time, it is stated in the article that Metro is by far (25% difference) the fastest service of the three. Those differences can easily be explained by investigating how the services have been deployed. We see that all services have been run a Tomcat server while our tests ran all services on their native servers shipped with the framework, hence the discrepancies between the two results. As such we can see that the art of deployment plays a major role when doing such a comparison.

5. Conclusion

In this paper we wanted to have look at the interoperability and the performance of web service implementation technologies. We tried to identify reasons for interoperability problems and to acquire a benchmark for web service frameworks.

As the implementation approach that is used when developing a web service has a major influence on the interoperability, we analyzed the two common implementation approaches *Code-* and *Contract-First*. Doing so, we learned that *Contract-First* is the approach reducing the risk of interoperability problems while *Code-First* provokes them.

Furthermore we implemented different web service frameworks and analyzed them with regard to their performance. We discovered that all the services except the one implemented in PHP behave similar in terms of the average response time per request until 250 loadUI requests per second. We also could see that some of the services (CXF, Metro, WCF) were able to handle more requests per second than others (PHP, gSOAP) and that the Axis2 service had the lowest average response time until the point at which it is no longer able to handle the amount of requests. Additionally, we made the experience that the services lead to significant different CPU usages. In this context the WCF service dominates the other services while the PHP service is the one with the worst performance, again. Furthermore, the performance test showed that running the services result in great differing RAM usages. While running the CXF service results in a very high RAM usage, the gSOAP and the WCF service constantly require a very low one.

In an interoperability test we checked whether the implemented web services and a subset of their clients are interoperable. We saw that this was the case which confirms the *Contract-First* approach to reduce the risk of problems with interoperability.

After all, the selection of an implementation approach and of a web service framework depends on the web service that is intended to be created. If interoperability is essential not *Code-* but *Contract-First* should be used. Regarding the web service frameworks one can say that the one with highest performance is not stringently the one a developer should decide on. As different frameworks have different pros and cons, as they support different standards, developers should firstly determine the frameworks that fulfill their needs and secondly consider the frameworks' performance.

A. Appendix

For a better comprehension of our work we want to provide

- the WSDL-file,
- the implemented web services,
- the implemented clients
- and the test results.

All of them can be downloaded from `https://github.com/felixguendling/web-service-interop-and-performance`.

References

[1] Institute of Electrical and Electronics Engineers (1990). *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York.

[2] O'Brien, J., Marakas, G.. *Introduction to Information Systems*. McGraw-Hill/Irwin.

[3] Frotscher, T. et al (2007). *Java Web Services mit Apache Axis2*. Frankfurt am Main: Software & Support Media GmbH.

[4] Christensen, E. et al (2001, March 15). *Web Services Description Language (WSDL) 1.1*. Available from `http://www.w3.org/TR/wsdl` (2012, May 21).

[5] Hardley, M. (2009, August 31). *Web Application Description Language*. Available from `http://www.w3.org/Submission/wadl/` (2012, May 21).

[6] Sperberg-McQueen, C.M., Thompson, H. (2011, November 2). *XML Schema*. Available from `http://www.w3.org/XML/Schema` (2012, May 22).

[7] *WS-Interoperability Organization (WS-I)*. Available from `http://www.ws-i.org/` (2012, May 22).

[8] Chumbley, R. et al (2010, November 9). *Basic Profile Version 2.0*. Available from `http://ws-i.org/Profiles/BasicProfile-2.0-2010-11-09.html` (2012, May 22).

[9] Ferris, C. (2002, October 1). *First look at the WS-I Basic Profile 1.0*. Available from `http://www.ibm.com/developerworks/webservices/library/ws-basicprof/index.html` (2012, June 29).

[10] (2012, April 17). *Axis2 Quick Start Guide*. Available from `http://axis.apache.org/axis2/java/core/docs/quickstartguide.html` (2012, May 21).

[11] *Axis2 - The Future of Web Services*. Available from `http://www.jaxmag.com/itr/online_artikel/psecom,id,747,nodeid,147.html` (2012, June 29).

[12] (2012 April 17). *Apache Axis2/Java Version 1.6.2 Documentation Index*. Available from `http://axis.apache.org/axis2/java/core/docs/contents.html` (2012, June 29).

[13] (2012 April 17). *Welcome to Apache Axis2/Java*. Available from `http://axis.apache.org/axis2/java/core/` (2012, June 29).

[14] *Apache CXF: CXF User's Guide*. Available from `http://cxf.apache.org/docs/index.html` (2012, May 21).

[15] *XFire and Celtix Merge*. Available from `http://xfire.codehaus.org/XFire+and+Celtix+Merge` (2012, June 4).

[16] *Apache CXF: An Open-Source Services Framework*. Available from `http://cxf.apache.org/` (2012, June 29).

[17] *What Is Windows Communication Foundation*. Available from `http://msdn.microsoft.com/en-us/library/ms731082.aspx` (2012, June 4).

[18] (2011 June 25). *Windows Communication Foundation Getting Started Tutorial*. Available from `http://msdn.microsoft.com/en-us/library/ms734712.aspx` (2012, July 1).

[19] *Introduction to Metro*. Available from `http://metro.java.net/guide/ch01.html` (2012, June 4).

[20] *gSOAP Success Stories*. Available from `http://www.cs.fsu.edu/~engelen/soapstories.html` (2012, June 30).

[21] *The gSOAP Toolkit for SOAP Web Services and XML-Based Applications*. Available from `http://www.cs.fsu.edu/~engelen/soap.html` (2012, June 30).

[22] *Web Services Interoperability Technologies*. Available from `http://wsit.java.net/` (2012, June 29).

[23] *WSIT Endorsements*. Available from `http://wsit.java.net/endorsements.html` (2012, June 29).

[24] (2011, November 15). *StackComparison*. Available from `http://wiki.apache.org/ws/StackComparison` (2012, June 29).

[25] Sosnoski, D. (2010, April 27). *Java web services: CXF performance comparison*. Available from `http://www.ibm.com/developerworks/java/library/j-jws14/index.html` (2012, July 2).

[26] *How to create a gSOAP stand-alone server*. Available from `http://www.cs.fsu.edu/~engelen/soapdoc2.html#tth_sEc7.2.3` (2012, June 30).

[27] (2012, November 19). *Thinktecture WSCF.blue*. Available from `http://wscfblue.codeplex.com/` (2012, June 30).

[28] *soapUI - The Home of Functional Testing*. Available from `http://www.soapui.org/` (2012, July 2).

[29] *loadUI - The Home of Load Testing*. Available from `http://www.loadui.org/` (2012, July 2).