TÉCNICO LISBOA

Distributed Programming in Cloud Computing Platforms

Enes UYSAL

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor(s): Prof. José Carlos Martins Delgado

Examination Committee

Chairperson: Prof. Full Name
Supervisor: Prof. José Carlos Martins Delgado
Member of the Committee: Prof. Full Name 3

May 2016

Abstract

In Cloud environment, programming distributed solutions, covering interoperability between heterogeneous systems, They are essentially based on SOA or REST technology, based on HTTP, XML and JSON. The problem is that these technologies were developed in the Web context where the main objective is the integration of existing systems and not programming distributed with efficiency and performance as top concerns. How to compromise these two views in the global environment, on cloud computing?

SOA and Web services are heavy and complex technologies. The popularity of REST is due more to its simplicity than to its mapping the paradigm of services. However, any of them is not very efficient, based on HTTP and languages text-based data description (XML and JSON). The result is that the description of Web Services (WSDL) and languages high-level programming, are inefficient and complex. The binary version of XML is no more than a compression data only for transmission effect (compressed and decompressed on the issue at the reception). A solution used in this work uses binary data format natively, does not require schema and the runtime environment uses Web Sockets, much more efficient than HTTP, while maintaining some compatibility.

Our solution developed for distributed programming in cloud computing environment to develop distributed applications, comparing this solution with the more classical solutions based on SOA and REST. The aim of this work explain this solution and its environment execution. Comparison with the corresponding solutions with technologies based on SOA (Web Services) and REST.

Keywords: SOA, WSDL, REST, XML, JSON, HTTP, Web Sockets

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Distributed applications are required in most of central application sectors including e-commerce, e-banking, e-learning, e-health, telecommunication and transportation[1]. This results that the Internet plays important role in business, administration and our everyday activities. In distributed applications that share information between each other does not need to be built with same technologies. The fundamental problem is the programming of distributed applications with all the basic interoperability problems involving distributed platforms and heterogeneous components.

On the other way, Clouds create a new challenge for distribution. They bring new opportunities such as scalability, dynamic instantiation, location independence, application management and multi-tenancy (several users using the same application independently)[2]. However, they also create distributed platforms and that's why they need to be able to support distributed applications as easily as possible. That's make importance of interoperability in cloud environment because different cloud providers can have different properties and they need to communicate between each other.

Taking as an example the your Android mobile phone, nowadays huge part of the people use smartphones that connected Internet that check news, weathercast or more information like that. Let's say you have an application in your phone that informs you with current weathercast of your city. That application is most probably written in Java since your phone is using Android system and gets current weathercast over Internet that gets information from weathercast provider, so basically that application asks queries to weathercast provider over Internet and gets response data first and then parse that data and display information to your phone screen. During this data exchange happens, both your mobile phone and weathercast provider must understand each other even they don't use the same language because the provider could be working in a Cloud provider and written in C# language. The data is used in C# and Java is not same so they will not understand each other naturally. This interoperability (how to interconnect different programs written in different languages, running in different platforms) issue can be solved with current integration technologies such as SOA or REST using XML or JSON over HTTP and both platforms can communicate between each other. Finally you can get last weathercast information

from your Android phone or your IPhone using same weathercast provider even they use totally different technologies.

The traditional integration technologies, based on either SOA or REST, is that they use the document concept as the foundation, with a data description language as the representation format and schema sharing as the interoperability mechanism (both sides use the same schema such XML Schema) or using previously known data types. Other factors, such as a connectionless protocol (HTTP) and the lack of native support for binary data because the solutions were also based on text (XML or JSON) and contextual information, are limiting for many applications.

Let's explain this with an example, so if we relook the example above that explains that we can have two different technology and they can communicate each other. In case if they use SOA architecture style we need use XML (Extensible Markup Language), which both languages can understand easily since, it is a common language. But still we have some steps to deal such as a stub, which is used by a client application to access a remote service. The stub looks like the service interface[3]; it generates an appropriate XML message and sends it over HTTP then we have XML message that can be read by our mobile phone or weathercast provider and parsed to build a DOM tree to be used. That can be done because they share same schema and they know structure using same schema. There are some problems with using XML because it is text based language, so it can be big and heavy and we need parser to read, write or update this XML message, which brings serious problems. So when you want to see weathercast on your phone, firstly your phone creates query message in XML using stub generator then uses HTTP for message transportation. When message arrives to provider it parses that XML message to own objects to understand it then provide response regarding the query, which is done by your phone. This response again needs to be converted to XML otherwise you phone will not understand it, so this time provider creates an XML message using stub generator and send that response message to your phone and your phone uses XML parsers to understand XML message and uses response data to display information to you. As we have seen in that example it is not very easy to communicate systems that use totally different technologies. In case of REST we can use different message format instead of XML like JSON which easier to parse and less heavy than XML, but in REST both languages needs to know message structure then they can parse and understand.

In case of REST there are some differences. If our weathercast application uses REST to communicate with weathercast provider then we don't need to create XML message to send weathercast provider. We can use other formats to send it using HTTP to server. Lets say we use JSON and your mobile phone creates query in JSON format then it needs to know unique URI of weather cast provider and HTTP verbs (GET, POST, PUT, and DELETE) that need to use, because in REST we can use different HTTP methods with using same URI. When a message arrived to weathercast provider this message will be parsed with DOM parsers to understand query. In that case your phone and weathercast provider must know structure of JSON message than they can both understand each other. After a message

parsed by provider then it creates response message in JSON format to send back your phone and now your phone displays weathercast to you.

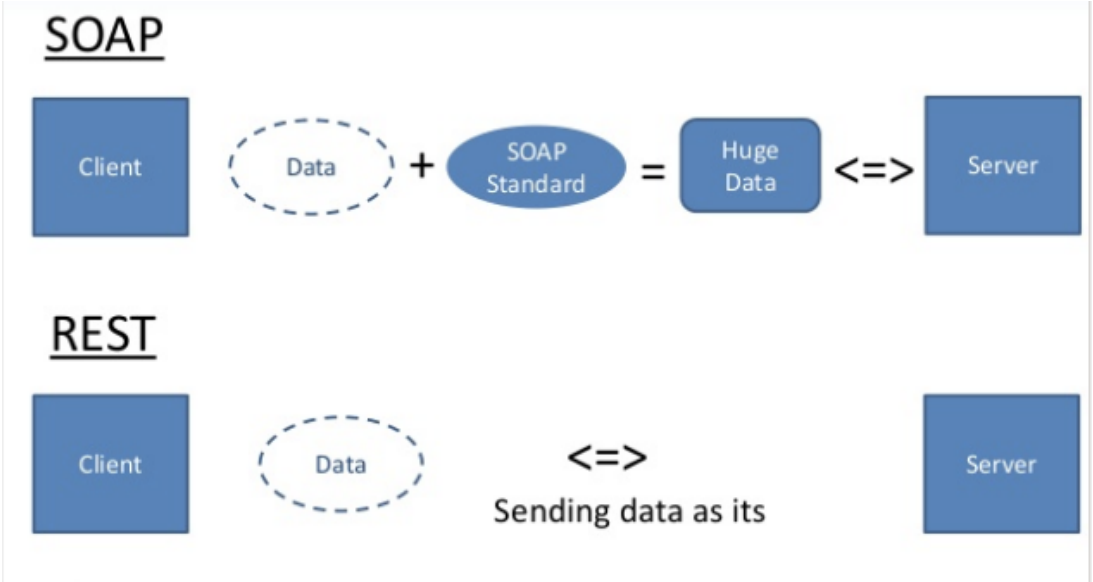

Figure 1.1: SOAP and REST Approach.

As a solution to the problems mentioned above like using XML or JSON based on text which is heavy and hard to parse, now think a new solution way about that problem with using binary directly instead of text for message transportation which reduce complexity and using Web Sockets and HTTP/2 protocol (a binary protocol with small message frames) instead using classical HTTP. The current solutions that are described above use schema sharing to understand message structure and what about using compliance and conformance instead of sharing the same schema.

Following chapters will give more details about these topics. Our solution is aim to show that there can be an alternative solution which combines the best of SOA and REST for document sharing or service invocation between two completely different systems and implementing a distributed application with a client programmed in one language and a remote service programmed in possibly another language.

The main goal of this dissertation is more than just an implementation and the value of this dissertation lie in the demonstration of conclusions, with respect to Web Services and REST. Showing results that if it is a better solution for application interoperability and also if it is easier to implement. We also assess performance to show advantage of using binary, with comparison current solutions.

This remainder of this dissertation is organized as follows:

- State of Art - Chapter 2 details some aspects of existing solutions for distributed systems for cloud environment, namely SOA, REST, text based data (XML and JSON). it also details about new tools

that we will use, namely HTTP/2 and Web Sockets.

- Interoperability - Chapter 3 starts describing interoperability and explaining different perspective from classical solutions to our new solution to overcome interoperability problem.
- Architecture of the solution - Chapter 4 provides some insight on how our system can be used and gives a general overview of how and why it works.
- Implementation - Chapter 5 goes more in depth on the inner workings of our system than the previous chapter and presents one implementation for our system.
- Comparison with existing technologies - Chapter 6 presents the benchmarks used to evaluate our system with current solutions and the results obtained.
- Conclusions - Chapter 7 summarizes the work described in this dissertation, the results achieved, and what are its main contributions. It also presents some possible future improvements to our proposed solution.

Chapter 2

State-of-the-art

2.1 Overview

A detailed description is provided in this section concerning the state of the art technologies existing nowadays in the market. The main integration technologies available today are based on Web technologies, namely HTTP, XML, JSON, Web Services using either SOA or REST architectural styles, with Web Services and REST on HTTP. This section also gives information about new tools such as HTTP/2 and WebSocket technologies that we will use in our solution.

2.2 Web Services

Web services allow you to use two different machines or two different pieces of code that talk each other. Two different applications can talk to each other over the network. They can call methods of each other over the network by using web service technology. The other advantage of using web services is that actually it is a standard technology because it is not really specific to Java or any other language. You can write web services using all other technologies. For example you can write a web service with Java, .Net, Python, C++ or others. The best part of web service standard what is called as Interoperability that is a property of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, without any restricted access or implementation[4].

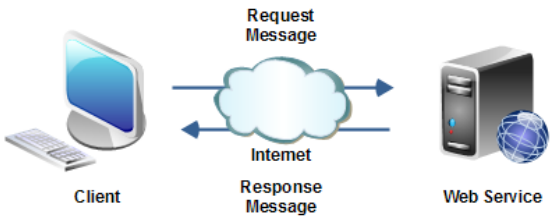

Figure 2.1: Simple Web Service Architecture.

For example let's say you have written web service in Java, let's say you have another web service that written in .Net so what the web service technology allows you to do is, Java can call .Net web service and .Net web service can also call Java web service and it just doesn't have to be web service because it can be an application. Let's say you have another application written in C++, so you can actually have C++ app that call Java web service or .Net web service. You can actually have different applications written in different technologies that communicate with each other during execution time and they can call each other. This actually let us to do that you can actually pick and choose technologies that you want to use let's say you have set of business web services that are implemented in .Net and your Java application can use them.

Web Services are exposed to the Internet for programmatic access. They are online APIs that you can call from your code. When you want to call any API that written by someone else to your Java code, you basically add jar or classes to your class path and executions are done inside of machine or single environment. In the case of web services however you have different pieces of code deployed over different machines and they call methods of each other over the network. For example, you must have seen that different apps or games, which can post to your Facebook wall even these games, not designed by Facebook. So you ask that how they can do that or how they can post to a wall of completely different system or application. Basically they do this by calling online APIs. Companies like Facebook or Twitter publish web services that let other developers call them from their code, so other application developers can actually write code to consume these services and they can post things on Facebook or Twitter. They can read or access data from Facebook or Twitter using the APIs of the web services that Facebook or Twitter has provided.

Web services are similar to web pages. For example Twitter has web side URI called as "www.twitter.com" when you access this URI on your browser you get an HTML response that let you read and write tweets. They have HTML elements for data and also CSS files for styling, this is because web pages that you see are made for human conception. They know that there is actually human is behind of browser on a laptop or devices who was reading these tweets, so they want to make sure about its format properly, thats way it is easy to access and read. Twitter has also other URI as "api.twitter.com" that does a lot of same things as "www.twitter.com" does, but it behaves a bit differently for instance this API gives you response which doesn't have HTML or CSS code. It contains data but it is XML or JSON format and there are specific URIs for different operations this is what the developers can use from their code to read or write to twitter, so this data is actually very easy for parsing and converting then using in their objects and their code for developers. In this case there is no need to have HTML and CSS files.

There are primarily two different types of web services. One of these types called as Simple Object Access Protocol (SOAP) web service and another type called as REpresentational State Transfer(REST) web service. SOAP is older of these two and REST is newer entry to web services world, but both of them are used popular. Next chapter will be focusing on SOAP and REST web services.

2.3 SOAP Web Services

Simple Object Access Protocol, or SOAP as it was the first attempt to standardize a web service interface. It is based on sending an XML message to a service, in a specific XML format, and receiving an XML response in another specific format. The message can be sent across different transports, including HTTP, FTP (File transfer Protocol), SMTP (Simple Mail Transfer Protocol) and more [5]. The specification does not dictate the transport over which the message should be sent, but most implementations send the XML message over HTTP.

I will try to explain shortly SOAP web service by an example. Let's start an example with java application. Let's say we have implementation class and I want to share this implementation class with other developer projects. How I would share this with a consumer class. The best way to share this implementation class would be to contract it with an interface and other consumers would consume this class through this interface. They would call implementation through interface so they get contract and they get the methods, arguments, written types through interface. They actually call the methods of implementation class, so how this works in case of web service, let's say I have web service implementation and I want to share details of this web service to consumers. Is it works with an interface? Probably it will not work because as we discussed before you don't know what technology is consuming it. It might be .Net application or C++ application, so if you have a java web service you might want to give some kind of information that its consumer respects to that technology and that can actually consume. Let's say consumer is .Net and if I give this .Net application a java interface then most probably it will not work because they are different technologies, so the technology that we are going to share with web service consumer has to be technology independent. It should be something that any application or any technology can understand so creators of SOAP web service talked about that problem and what to give format understandable by all technology with all consumers and decided with XML. So what we do is in case of web service, we actually share that contract as an XML document. This XML document is actually called as WSDL(Web Services Description Language)[6].

WSDL (a set of conventions on XML usage) document contains the contract to web service and so that's are the things you have to do when you create the web service and you share WSDL document of that web service to the consumers, so this is not something you would have to do it manually. You would do it manually but there are tools which generate WSDL for the web service. There is a stub generator for every language to create that WSDL document. It is something that you need to share this WSDL to consumers and it is a XML document, so it respective whatever application because applications such as .Net, C++ or Java can all parse this XML[7] and get to know about service information and typically the content of this WSDL is kind of similar to an interface content. It has operations, arguments and types to return that consumer applications will have idea what to call and how to call.

The new question is that how this exchange happens, how you actually send this information, let's

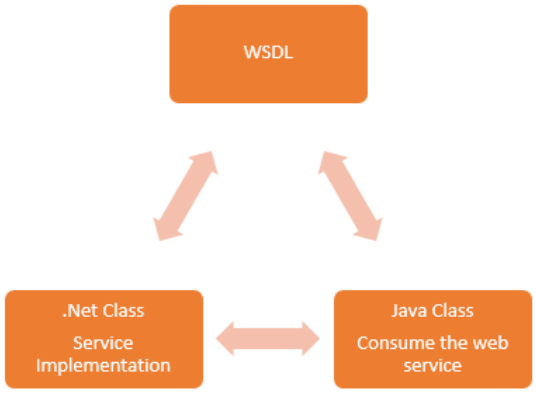

Figure 2.2: WSDL (Web Services Description Language).

say you have a method in your java application and input argument is a string so you have a java string with you and you need to send to web service and let's say output return type is a list, so how you get this information because that could be .Net application and string in Java obviously different from string in .Net. How do you exchange this between client app and web service? When you exchange information input argument or return type you need to exchange it in the format that all different technologies can understand what you are passing and it should be able to send return type back in language that all these technologies can understand. Again this format is XML. When you are sending any information across the network from a client to the web services and return type back to the client, the data has to be in XML format. You are not really sending java string or a list. So it has to be language natural format which is XML. There is specification about how you need to send all these different input type and output argument basically any type needs to be send specific XML format.

It is a protocol that is a way in which both sender and receiver and this XML is called SOAP(Simple Object Access Protocol). It is a way in which these different technologies can access objects can access data it supposedly simple so that a part of the name is called simple object access protocol. So with this protocol all different technologies written in different languages can kind of understand what they all taking about.

Now we know what is the mechanism we know what need to be send and we know how to send which is using SOAP protocol but who does conversion? So for example you have your string object or complex object, so how do you convert it from Java object to a SOAP message? The conversion is actually done with intermedia class so this class takes care of converting all your objects into a SOAP message. The whole method calls itself is actually done by SEI(Service Endpoint Interface)[8]. The SEI access interface to your web service endpoint so you have an interface at your client app to the service endpoint which translate all web service call to a SOAP message and then it makes sure that the other things is able to understand this message. So we don't have to write this class and all the conversion ourselves. We can have it automatically generated for us. When you are making a web service call

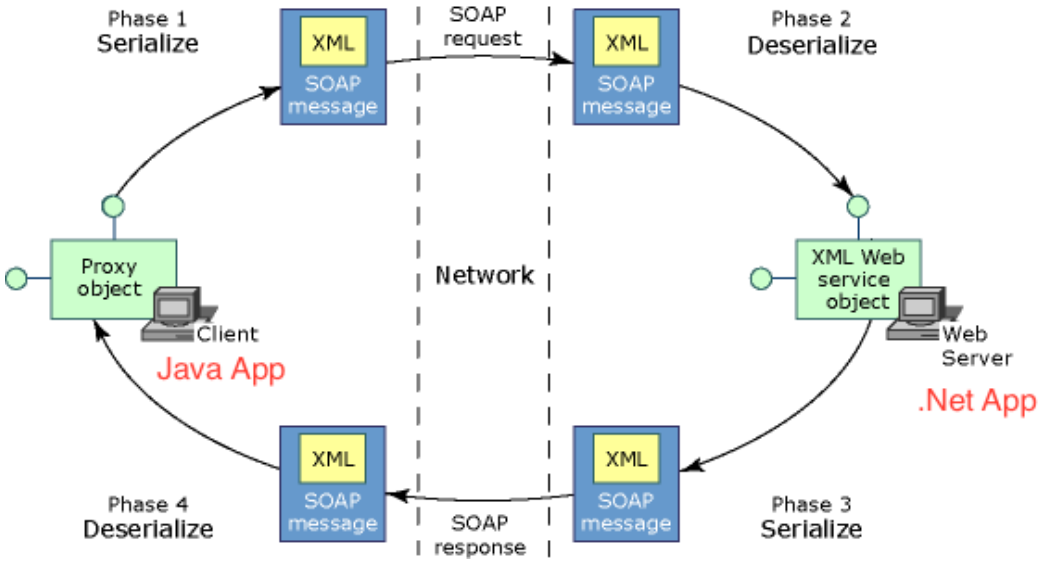

Figure 2.3: SOAP (Simple Object Access Protocol).

you don't worry about where the web service is. When you need to call, all you need to do is have this endpoint interface and good thing about this service endpoint that you can actually have an interface that specific to what you are developing. When you have a java application you will have a specific SEI for java application and it knows to convert Java objects to SOAP message. Let's say your .Net application calling the same web service so you will have SEI for .NET that know to convert .Net objects to SOAP message.

2.4 REST Web Services

REST (Representational State Transfer) was created in 2000 by Roy Fielding[9]. Developed in an academic environment, this protocol embraces the philosophy of the open Web. Instead of using XML to make a request, REST relies on a simple URL in many cases. In some situations you must provide additional information in special ways, but most Web services using REST rely exclusively on obtaining the needed information using the URL approach. REST can use four different HTTP 1.1 verbs (GET, POST, PUT, and DELETE) to perform tasks.

Resources are the fundamental building blocks of web-based systems. A resource is anything we expose to the Web, from a document or video clip to a business processor device. A URI uniquely identifies a web resource, and at the same time makes it addressable, or capable of being manipulated using an application protocol such as HTTP. The relationship between URIs and resources is many-to-one[10]. A URI identifies only one resource, but a resource can have more than one URI. That is, a resource can be identified in more than one way, much as humans can have multiple email addresses or telephone numbers. This fits with our need to identify real world resources in more than one way. Everything in REST web services has URI is unique and standard. For example Facebook, when you open an ac-

count on Facebook, you will get a profile page that obivously dynamically generated pages, so whenever there is a new profile, it basically the same page which does same processing but render different content depending on profile that you are watching. In REST web services we need to think of resources and create unique urls for them. For example you are creating weathercast application and you want to get weather for different cities of Portugal, so your URI needs to be unique for each city as in Table 2.1.

URI	Description
http://weatherapp.com/city/Lisbon	Unique URI for Lisbon city
http://weatherapp.com/city/Porto	Unique URI for Porto city
http://weatherapp.com/city/Algarve	Unique URI for Algarve city

Table 2.1: URI Example.

In REST web services, with same unique URI we can do different actions, for example if you are administrator of weathercast application and you want to get data of Lisbon city or if you want to update data of Lisbon city or deleting data of Lisbon city you can use the same unique URI for all these methods using HTTP methods. Only a single concrete URI, a single URI template, and four HTTP verbs. In fact, it's so compact that we can provide an overview in just a few lines, as shown in Table 2.2.

Verb	URI or template	Use
POST	/city	Create a new city
GET	/city/cityName	Request the data of city specified by the URI.
PUT	/city/cityName	Update an city data at the given URI with new information.
DELETE	/city/cityName	Logically remove the city identified by the given URI.

Table 2.2: HTTP Methods.

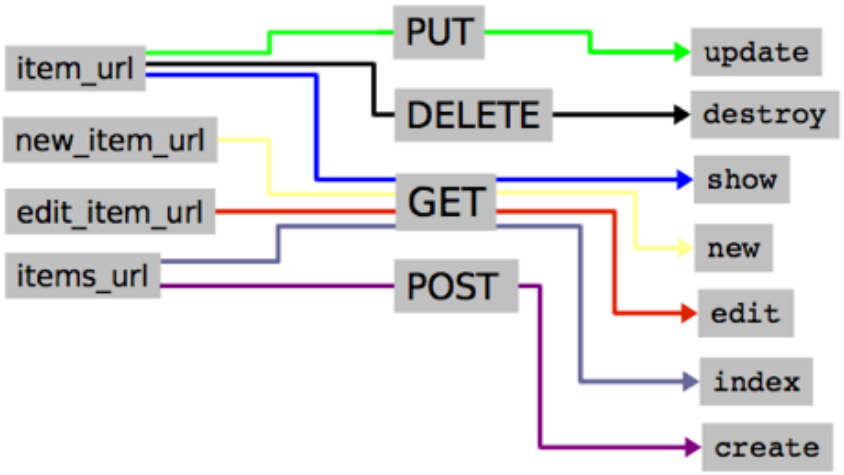

Figure 2.4: REST.

We've looked at requests so far, and understood what resource URIs and HTTP methods. We make request to resource URIs with using HTTP methods. Let's switch to responses now. After making re-

quest, RESTful web service will be responded. It is important format of response because client needs to write code to handle that response. if that were a web application, the response would be HTML page with styling formatting. When it comes RESTful Web service you don't need all this styling because it can be basically XML or JSON. Following section will give detailed information about XML or JSON.

2.5 Serialization formats (XML, JSON and Binary)

The contents of messages in distrubuted systems need to be serialized to be sent over the channel and the format to do so (such as XML or JSON). A schema is used to transform the internal data structures into serial messages and vice-versa. The serialization formats used in the Web (e.g., XML, JSON) are text-based and thus verbose and costly in communications. Technologies have been developed to compress text-based documents [11], such as EXI (Efficient XML Interchange), BSON and others [12]. However, these are compression technologies, which need text parsing after decompression.

```
1   public class Person {
2     public String name;
3     public int age;
4     public String eyeColor;
5     }
6
7   Person p = new Person();
8   p.name = "Peter";
9   p.age = 34;
10  p.eyeColor = "blue";
```

Listing 2.1: Example Person Object Class

XML, which maintained the text markup style of HTML, made computer based clients easier and, together with HTTP, became the cornerstone of Web Services technologies. This evolutionary transition is perfectly understandable in market and standardization terms, but still constitutes a mismatch towards both humans and applications. XML is text based and support schemas. XML retains the look and feel of HTML, with text markup based on start and end tags as seen example in in Listing 2.2 of in Listing 2.1 Person Object. XML has generalized HTML, separating data from formatting and introducing self-description with a schema, but retained much of its look and feel, still with a data document nature (just data, instead of a more complete service nature, by including code) and text with markup (lacks native binary support). There are technologies such as EXI, the binary XML format, which offers compression at less cost than using Gzip, and save processing otherwise needed to decompress compressed XML[13].

```
1   <Person>
2     <name>Peter</name>
3     <age>34</age>
```

```
4       <eyeColor>blue</eyeColor>
5     </Person>
```

Listing 2.2: JSON presentation of Person object

JSON is much more populer than XML[14], as a simpler alternative to XML because it is more compact especially you have large amount of data and also when client is browser which is piece of javascript code running on browser so sending response data in JSON can be easily parsed by client. JSON delimits data with syntax tokens, with a simple grammar that bears some similarity with data structures in C-like languages as seen example in in Listing 2.3 of in Listing 2.1 Person Object. XML has targeted flexibility and generality, whereas JSON has emphasized simplicity, which is after all the secret of its popularity. There is also BSON, the binary JSON format devised by 10gen and used in their MongoDB NoSQL database, instead of JSON [15]. As we described before these are compression technologies, which need text parsing after decompression.

```
1   {
2   "Person":
3     {
4     "name": "Peter",
5     "age": "34",
6     "eyeColor": "blue"
7     }
8   }
```

Listing 2.3: JSON presentation of Person object

In spite of the differences, both suffer from drawbacks and limitations:

They are text based, which means inefficient parsing and data traversal where all characters of a component need to be parsed to reach the next one, high memory consumption, relevant message transmission times and poor support for binary data. The serialization format should be as close as possible to the data structure to minimize the conversion effort in serialization and deserialization. Text can be thought as human readable and therefore advantageous over binary, but this is true only for very simple documents, especially when using XML. Binary compression mechanisms exist [11][12], but this does not reduce the parsing time, since text need to be recovered. It would be better to follow the old example of programming languages, by using a source format for humans, a binary format for computers and a compiler to synchronize them.[14]

2.6 HTTP/2

HTTP/2 is the second major version of the HTTP network protocol used by the World Wide Web. HTTP/2 will make our applications faster, simpler, and more powerful. The primary goals for HTTP/2 are to re-

duce latency by enabling full request and response multiplexing, minimize protocol overhead via efficient compression of HTTP header fields, and add support for request prioritization and server push[16]

The first important thing to notice about HTTP/2 is that it is not a replacement for all of HTTP. The verbs, status codes and most of the headers will remain the same as today. HTTP/2 is about becoming more efficient in the way data is being transferred on the wire[17].

HTTP/2 breaks down the HTTP protocol communication into an exchange of binary-encoded frames, which are then mapped to messages that belong to a particular stream, and all of which are multiplexed within a single TCP connection. HTTP/2 communication is split into smaller messages and frames, each of which is encoded in binary format. This is the foundation that enables all other features and performance optimizations provided by the HTTP/2 protocol.

HTTP/2 uses header compression to reduce overhead. Typical header sizes of 1KB are common mainly because of the cookies that we all have to accept for a smooth user experience. Transferring 1KB can take several network round trips just to exchange headers, and those headers are being resent every time because of the stateless nature of HTTP

HTTP/2 Server Push allows servers to proactively send responses into client caches. In a typical HTTP workflow, the browser requests a page, the server sends the HTML in the response, and then needs to wait for the browser to parse the response and issue additional requests to fetch the additional embedded assets (JavaScript, CSS, etc.). Server push allows the server to speculatively start sending resources to the client. Here, the browser does not have to parse the HTML page and find out which other resources to load; instead the server can start sending them immediately.

2.7 Web Sockets

WebSockets[18] are a relatively new technology which promises to make websites more reactive by allowing lower latency interaction between users and the server. WebSockets circumvent some of the limitations of HTTP which removes this restriction, adds binary support and increases performance.

WebSocket is a protocol which allows for communication between the client and the server/endpoint using a single TCP connection. it sounds a bit like HTTP. The advantage WebSocket has over HTTP is that the protocol is full-duplex (allows for simultaneous two-way communcation) and it's header is much smaller than that of a HTTP header, allowing for more efficient communcation even over small packets of data.

Web Sockets are also fundamental in the efficient support for binary data and increases performance.

Example of WebSocket life cycle:[19]

1.Client sends the Server a handshake request in the form of a HTTP upgrade header with data about the WebSocket it's attempting to connect to.

2.The Server responds to the request with another HTTP header, this is the last time a HTTP header gets used in the WebSocket connection. If the handshake was successful, they server sends a HTTP

header telling the client it's switching to the WebSocket protocol.
3.When connection is opened and the client and server can send any number of messages to each other until the connection is closed. These messages have very low bytes of overhead.

Now, if we compare HTTP/2 against WebSocket[17], we can see a lot of similarities in Table 2.3:

	HTTP/2	WebSocket
Headers	Compressed (HPACK)	None
Binary	Yes	Binary or Textual
Multiplexing	Yes	Yes
Prioritization	Yes	No
Compression	Yes	Yes
Direction	Client/Server + Server Push	Bidirectional
Full-duplex	Yes	Yes

Table 2.3: HTTP/2 and WebSocket.

As explained above they both support for binary data and increases performance. Instead of using classical HTTP, using these technologies will improve system performance.

Chapter 3

Interoperability

3.1 Overview

Resources need to interact to accomplish collaboration, either designed or emergent. This necessarily entails some form of mutual knowledge and understanding, but this creates dependencies on other resources that may hamper resource evolution and that's why Interoperability, a necessary condition to achieve integration between different systems. Another important factor for integration between systems is decoupling which says that resources need to be independent to evolve freely and dynamically. Unfortunately, independent resources do not understand each other and are not able to interact. Therefore, the fundamental problem of resource integration is to provide the maximum decoupling possible while ensuring the minimum interoperability requirements.

3.2 Interoperability in SOA and REST

Currently, enterprise integration is based on SOA with Web Services and REST with HTTP; the two most used architectural styles for distributed interoperability. These styles use symmetric arrangement in which a sender produces a message according to some schema and the receiver uses the same schema to validate and to get the contents of the message. The message is sent over a channel between sender and receiver. In SOA web services, schema is usual in XML Schema and WSDL. In the REST world, schemas are known as media types but perform the same role. The difference is that, instead of being declared in a separate document referenced by messages, they need to be previously known to the interaction resources, either by being standard or by having been previously agreed. In any case, the schema or media type must be the same at both sender and receiver and therefore imposes coupling between the resources for all its possible values, even if only a few are actually used. In either case, data types need to be fully known by both interacting resources.

As described above, resources send requests to each other to invoke a given operation with a SOA or REST approach. These requests and their responses usually contain data, which are serialized, sent and reconstructed upon reception of the corresponding message. The sender and receiver need to

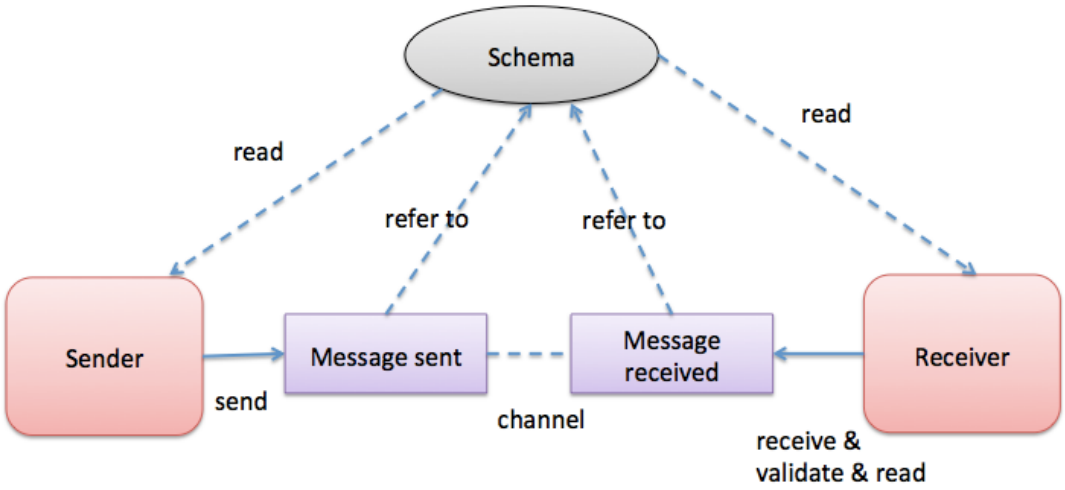

Figure 3.1: Interoperability in SOA and REST.

interpret those data in a compatible way, which means they need schema. These schemas can be XML Schema and JSON Schema. JSON is a very popular serialization format, as a simpler alternative to XML. JSON Schema is currently just an IETF draft[20], but is raising interest as a simpler alternative to XML Schema. Both XML and JSON are text based which means inefficient parsing and all characters of a component need to be parsed to reach the next one. Let's explain this with an example, so if we relook the example in Chapter 1 that explain interaction between Android mobile phone and .Net weathercast provider. In case of REST approach I need to create an XML or JSON message to send server. Lets see how we can create an XML message with Android phone to send it weathercast provider. The code in Listing 3.2 explains how to create an XML message in Listing 3.1 in Java.

```
1
2   <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
3   <weather>
4       <country>Portugal</country>
5         <city>Lisbon</city>
6         <date>30.03.2016</date>
7         <weatherResult></weatherResult>
8   </weather>
```

Listing 3.1: Example XML message to send weathercast provider

```
1
2         DocumentBuilderFactory docFactory =
              DocumentBuilderFactory.newInstance();
3         DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
4
5         // root elements
6         Document doc = docBuilder.newDocument();
```

```
7         Element rootElement = doc.createElement("weather");
8         doc.appendChild(rootElement);
9
10        // country elements
11        Element country = doc.createElement("country");
12        rootElement.appendChild(country);
13
14        // city elements
15        Element city = doc.createElement("city");
16        city.appendChild(doc.createTextNode(cityInfo));
17        rootElement.appendChild(city);
18
19        // date elements
20        Element date = doc.createElement("date");
21        date.appendChild(doc.createTextNode(new Date().toString()));
22        rootElement.appendChild(date);
23
24        // weatherResult elements
25        Element nickname = doc.createElement("weatherResult");
26        nickname.appendChild(doc.createTextNode(""));
27        rootElement.appendChild(nickname);
28
29        TransformerFactory transformerFactory =
              TransformerFactory.newInstance();
30        Transformer transformer = transformerFactory.newTransformer();
31        DOMSource source = new DOMSource(doc); // XML Message
```

Listing 3.2: XML message creation with Java Code

After creation of XML message in Java, this message will be sent with suitable message level protocol (e.g., HTTP) and will be received by the provider. Weathercast provider recover the message, using the provider's protocol. Now .Net provider need to parse this message to understand query. The code in Listing 3.3 explains how to parse an XML message in .Net.

```
1
2   string Country;
3   string City;
4   string date;
5   // Create an XmlReader
6   using (XmlReader reader = XmlReader.Create(new StringReader(xmlString)))
7   {
8   reader.ReadToFollowing("weather");
9   reader.ReadToFollowing("country");
10  Country = reader.ReadElementContentAsString();
```

```
11    reader.ReadToFollowing("city");
12    City = reader.ReadElementContentAsString();
13    reader.ReadToFollowing("date");
14    date = reader.ReadElementContentAsString();
15    }
```

Listing 3.3: XML message parsing with .Net Code

After parsing XML message .Net weathercast provider, it builds response data as XML message and then send to your mobile phone. You mobile phone need to parse that XML response message and display you the result. Here in this example we assumed that both your mobile and weathercast provider know the schema and they both can create same structure of XML message otherwise the componenent can not parse XML correctly and can not communicate each other.

The same example can be done with SOA approach. SOA leads to an architectural style that is an evolution of the RPC (Remote Procedure Call) style, by declaring and invoking operations in a standard and language independent way. Service description (WSDL document) must be obtained first by your mobile phone and weathercast provider, so that a contract can be established between that resource (the provider) and the resource that uses it (the consumer). A concrete schema, with the names used, must be specified and be compatible on both sides, otherwise a representation returned by a resource, for example, will not be able to be analyzed and understood.

We will have a web service in .NET for weather cast provider such as in Listing 3.4 and also following WSDL document will be created automatically for that web service by .NET as seen in Listing 3.5.

```
1
2   [WebService(Namespace = "http://tempuri.org/")]
3   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
4
5   public class WebService : System.Web.Services.WebService {
6
7       public WebService () {
8       }
9
10      [WebMethod]
11      public Weather GetWeatherCast(Weather weather) {
12          weather.weatherResult =
                WeatherCastProvider.GetWeather(weather.country, weather.city,
                weather.date);
13          return weather;
14      }
15
16  }
```

Listing 3.4: SOAP Web Service with .Net

So if your mobile application knows that WSDL document in Listing 3.5 it can call a method as in in Listing 3.6 from weathercast provider by RPC (Remote Procedure Call) style to get current weathercast info for requested city.

```
1
2   <?xml version="1.0" encoding="utf-8"?>
3   <wsdl:definitions xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:tns="http://tempuri.org/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:s="http://www.w3.org/2001/XMLSchema"
        xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        targetNamespace="http://tempuri.org/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
4     <wsdl:types>
5       <s:schema elementFormDefault="qualified"
            targetNamespace="http://tempuri.org/">
6         <s:element name="GetWeatherCast">
7           <s:complexType>
8             <s:sequence>
9               <s:element minOccurs="0" maxOccurs="1" name="weather"
                    type="tns:Weather" />
10            </s:sequence>
11          </s:complexType>
12        </s:element>
13        <s:complexType name="Weather">
14          <s:sequence>
15            <s:element minOccurs="0" maxOccurs="1" name="country"
                  type="s:string" />
16            <s:element minOccurs="0" maxOccurs="1" name="city" type="s:string"
                  />
17            <s:element minOccurs="1" maxOccurs="1" name="date"
                  type="s:dateTime" />
18            <s:element minOccurs="0" maxOccurs="1" name="weatherResult"
                  type="s:string" />
19          </s:sequence>
20        </s:complexType>
21        <s:element name="GetWeatherCastResponse">
22          <s:complexType>
23            <s:sequence>
24              <s:element minOccurs="0" maxOccurs="1"
```

```
                    name="GetWeatherCastResult" type="tns:Weather" />
25              </s:sequence>
26            </s:complexType>
27          </s:element>
28        </s:schema>
29      </wsdl:types>
30      <wsdl:message name="GetWeatherCastSoapIn">
31        <wsdl:part name="parameters" element="tns:GetWeatherCast" />
32      </wsdl:message>
33      <wsdl:message name="GetWeatherCastSoapOut">
34        <wsdl:part name="parameters" element="tns:GetWeatherCastResponse" />
35      </wsdl:message>
36      <wsdl:portType name="WebServiceSoap">
37        <wsdl:operation name="GetWeatherCast">
38          <wsdl:input message="tns:GetWeatherCastSoapIn" />
39          <wsdl:output message="tns:GetWeatherCastSoapOut" />
40        </wsdl:operation>
41      </wsdl:portType>
42      <wsdl:binding name="WebServiceSoap" type="tns:WebServiceSoap">
43        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
44        <wsdl:operation name="GetWeatherCast">
45          <soap:operation soapAction="http://tempuri.org/GetWeatherCast"
              style="document" />
46          <wsdl:input>
47            <soap:body use="literal" />
48          </wsdl:input>
49          <wsdl:output>
50            <soap:body use="literal" />
51          </wsdl:output>
52        </wsdl:operation>
53      </wsdl:binding>
54      <wsdl:binding name="WebServiceSoap12" type="tns:WebServiceSoap">
55        <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
56        <wsdl:operation name="GetWeatherCast">
57          <soap12:operation soapAction="http://tempuri.org/GetWeatherCast"
              style="document" />
58          <wsdl:input>
59            <soap12:body use="literal" />
60          </wsdl:input>
61          <wsdl:output>
62            <soap12:body use="literal" />
63          </wsdl:output>
```

```
64        </wsdl:operation>
65      </wsdl:binding>
66      <wsdl:service name="WebService">
67        <wsdl:port name="WebServiceSoap" binding="tns:WebServiceSoap">
68          <soap:address location="http://localhost:5270/WebService.asmx" />
69        </wsdl:port>
70        <wsdl:port name="WebServiceSoap12" binding="tns:WebServiceSoap12">
71          <soap12:address location="http://localhost:5270/WebService.asmx" />
72        </wsdl:port>
73      </wsdl:service>
74    </wsdl:definitions>
```

Listing 3.5: WSDL Document for Weathercast Provider

```
1
2   import CSharpWebService.Weather;
3   import CSharpWebService.WebService;
4   import CSharpWebService.WebServiceSoap;
5
6   public class Run {
7
8       public Weather GetWeatherCast() {
9
10          Weather  w = new Weather();
11          w.setCountry("Portugal");
12          w.setCity("Lisbon");
13          w.setDate(null);
14          WebService service  = new WebService();
15          WebServiceSoap port = service .getWebServiceSoap12();
16          return (port.getWeatherCast(w));
17      }
18  }
```

Listing 3.6: SOAP Web Service Client with Java

As seen in Listing 3.5 for very basic example we need a big WSDL document in case if we use SOA architecture.

3.3 Asymmetric interoperability

In our solution, we show how interaction is stil possible with only a partial knowledge of types, as long as the characteristics actually used are included (partial interoperability). This is a way of getting closer to solving the fundamental integration problem, by reducing coupling to what is actually required. Coupling

is thus a necessary bad, without it no interaction is possible. Our goal is to minimize it as much as possible, down to the minimum level that ensures the level of interaction required by the resources to integrate.

In this solutution we will show different approach, based on compliance. Messages do not obey some external schema. Each message has one specific value which are structured or primitive with its own exclusive schema that is nothing more than a self-description, without the value variability that a type exhibits. This value and its description can be validated against an infinite number of schemas, those that have this particular value included in the set of their instances.

The receiver in Figure 3.2 exposes a schema that defines the values it is willing to accept. When a

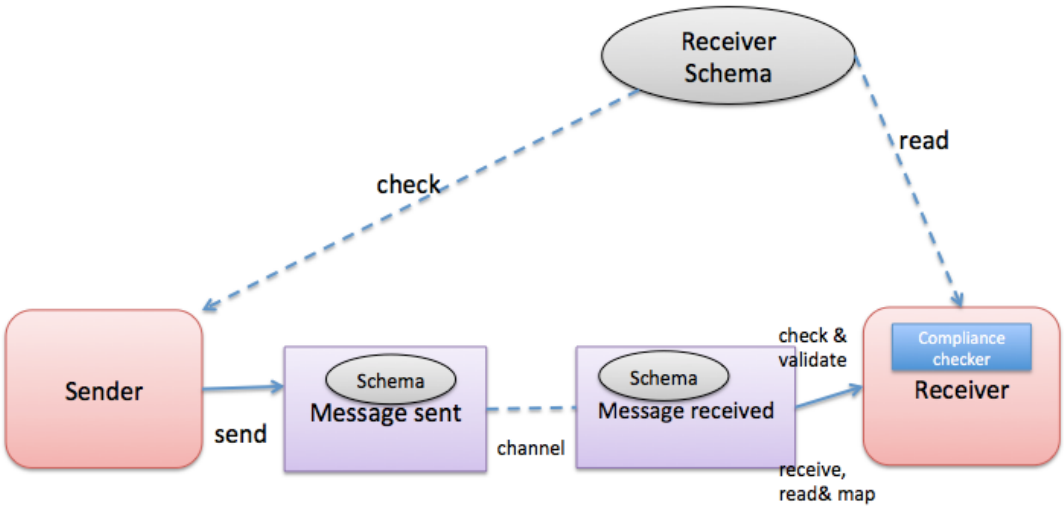

Figure 3.2: Asymmetric interoperability.

message is received, its internal schema is checked against the receiver's own schema. If it complies which means satisfies all the requirements of the receiver's schema, the message can be accepted and processed. The advantage of this is that a resource can send a message to all the resources with schemas that the message complies with and, conversely, a resource can receive messages from any resource that sends messages compliant with its receiving schema. In other words, coupling occurs only in the characteristics actually used by messages and not in all the characteristics of the schemas used to generate the message or to describe the service of the receiving resource. Since the schemas of the message and of the receiver are not agreed beforehand, they need to be checked structurally. Resources of primitive types have predefined compliance rules. Structured resources are compared by the names of components (regardless of order of declaration or appearance) and (recursively) by the compliance between structured resources with matching names. Since the order of appearance of named component resources may different in the message and in the receiving schema, there is the need to map one onto the other. This is a form of polymorphism that increases the range of applicability of both sender and receiver, constituting a means to reduce coupling to only what is actually used. Sender and receiver no longer need to be designed for each other but, as long as compliance is ensured, one resource can replace another. In this case, what is involved is conformance between the replacement

and the resource replaced, stating that the former supports all the characteristics supported by the latter. When a resource is able to interact with another, although not entirely interoperable with it, we say that have partial interoperability.

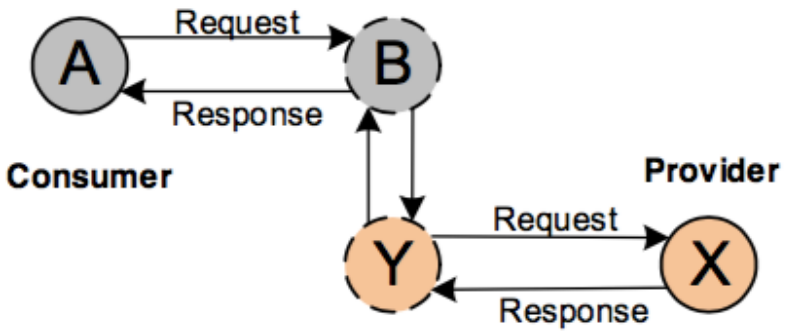

Figure 3.3: Partial interoperability, based on compliance and conformance.

Figure 3.3 illustrates these concepts and differentiates compliance from conformance. It also describes complete transactions, with both request and response. The interacting resources now perform the roles of consumer who makes the request and provider, with sender and receiver roles reversed from request to response. Interoperability of a consumer with a provider is possible by satisfying the following properties:

Compliance[21] which means that the consumer must satisfy (comply with) the requirements established by the provider to accept requests sent to it, without which these cannot be honored.

Conformance[22] which means that the provider must fulfill the expectations of the consumer regarding the effects of a request (including eventual responses), therefore being able to take the form of (to conform to) whatever the consumer expects it to be.

In full interoperability, the consumer can use all the characteristics that the provider exposes. This is what happens when schemas are shared. In partial interoperability, the consumer uses only a subset of those characteristics, which means that compliance and conformance need only be ensured for that subset. These properties are not commutative (e.g., if P complies with Q, Q does not necessarily comply with P), since the roles of consumer and provider are different and asymmetric by nature, but are transitive (e.g., if P complies with Q and Q complies with R, then P complies with R)

In Figure 3.3 , a resource A, in the role of consumer, has been designed for full interoperability with resource B, in the role of provider. A uses only the characteristics that B offers and B offers only the characteristics that A uses. Let us assume that we want to change the provider of A to resource X, which has been designed for full interoperability with resource Y, in the role of consumer. The problem is that A was designed to interact with provider B and X was designed to expect consumer Y. This means that, if we use resource X as a provider of A, B is how A views provider X and Y is how X views consumer A. Ensuring that A is interoperable with X requires two conditions such as Compliance and Conformance. For Compliance; B must comply with Y. Since A complies with B and Y complies with X, this means that A complies with X and, therefore, A can use X as if it were B, as it was designed for; For Conformance:

Y must conform to B. Since X conforms to Y and B conforms to A, this means that X conforms to A and, therefore, X can replace (take the form of) B without A noticing it. Partial interoperability has been achieved by subsumption, with the set of characteristics that A uses as a subset of the set of characteristics offered by X. This inclusion relationship, without changing characteristics, is similar in nature to the inheritance-based polymorphism supported by many programming languages, but here it applies to a distributed context. It constitutes the basis for transitivity in compliance and conformance, as well as the mechanism to reduce coupling between two resources to the minimum required by the application.

Chapter 4

Architecture of the solution

4.1 Overview

As we discussed before, The idea is to compare current technologies (Web Services, REST, XML, JSON) with our solution. This solution is a new programming technology alternative current solutions. Basically, this is an alternative to text-based data description (XML and JSON) technologies, for document sharing or service invocation, between two completely different systems. Following sections will explain about how we implemented the interoperability, our primitive data formats and solution of binary level compliance and comformance.

4.2 Binary Format

The one of ideas in this technology is using binary format instead of using text or other formats. Because binary is faster to write, communicate and read. When we compare serialization or deserialization performance with binary, XML and JSON, we easily see that binary format gives faster speed than the others especially with large data[23].

On the other hand, using text is far more flexible. Textual representation leads parsing overheads. Messages received are parsed directly in binary, much faster than text parsing. The binary representation provides native support for binary data, has a smaller length and is faster to parse.

We define a binary format to which messages are serialized on send and recovered on reception. For binary format using TLV (Tag, Length and Value) binary markup[24], we use an array of bytes with each resource serialized in a tag (a byte codifying each resource type), size, name (only on structure and resource components) and value (the actual sequence of bytes resulting from serializing the resource).

The binary format used to serialize the resources with TLV (Tag, Length and Value) binary markup as in Table 4.1. This supports the direct integration of binary information but also facilitates parsing, since

Format	Definition
Type	A binary code, which indicates the kind of field that this part of the message represents.
Length	The size of the value field in bytes
Value	Variable-sized series of bytes which contains data for this part of the message.

Table 4.1: TLV format (Type-length-value).

each resource, primitive or structured. Binary message format resulting from compilation of the source program and that uses self-description information and only when needed. This allow us maintaining all the information necessary to communicate in a standard and platform independent way.

As long as we control the serialization format, we can perform the serialization in one language and the deserialization in another. The binary format is always the result of serializing data in each language, with a tag, the number of bytes that follow and the serialized content. Recovering the serialized data is simply testing the tag to find the data type and then using the number of bytes and the serialized content.

Let's demonstrate this with an example, so if we relook the example in Chapter 3 that explain interaction between Android mobile phone and .Net weathercast provider. Your phone has an application that written in Java that send query message to .Net weathercast provider and displaying result regarding the respond from .Net weathercast provider.

We start by serializing that query to the binary format in Java. In our case Application in your phone will create binary message instead of XML or JSON to send weathercast provider as in Listing 4.1

```
1
2   public static void main(String[] args)  {
3
4         Weather w = new Weather();
5         w.country="Portugal";
6         w.city="Lisbon";
7         Message msg = new Message(w);
8         byte[] msgToSend =  msg.SeriliazeBinary();
9         string response = clientEndPoint.sendMessage(msgToSend);
10    }
```

Listing 4.1: Creating serialized binary Message

Creating a binary message will be similar to TLV (Tag, Length and Value) binary markup idea. Reagarding the code in Listing 4.1, the provided binary will be as in Listing 4.2.

```
1
```

```
2 [1, 3, 0, 0, 0, 7, 0, 0, 0, 7, 0, 0, 0, 8, 0, 0, 0, 8, 99, 111, 117, 110, 116,
    114, 121, 80, 111, 114, 116, 117, 103, 97, 108, 1, 3, 0, 0, 0, 4, 0, 0, 0,
    4, 0, 0, 0, 6, 0, 0, 0, 6, 99, 105, 116, 121, 76, 105, 115, 98, 111, 110,
    1, 3, 0, 0, 0, 4, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 100, 97, 116, 101]
```

Listing 4.2: Creating serialized binary Message

Weather object in Listing 4.1 is serialized by its primitive elements. So in Figure 4.1, we can see that how first primitive element of weather object, which is, country element is been serialized to binary.

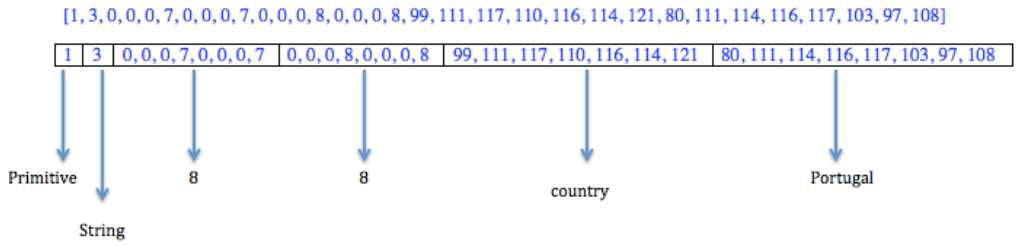

Figure 4.1: Binary representation of primitive resource.

Equation 4.1a informs that serialization starts with checking if the object is primitive or structure. After that equation 4.1b informs type of primitive object. In that case it is String object and then next part of binary (equation 4.1c) informs about primitive field name length, so here, name of the primitive field is “country” and its size is 7. Next part of binary (equation 4.1d) is field value length, which is “Portugal”, and its size is 8. Since we now size of field name and value we can convert them to binary. Equation 4.1e and equation 4.1f inform us name of the field and value of field.

$$1 \tag{4.1a}$$

$$3 \tag{4.1b}$$

$$0, 0, 0, 7, 0, 0, 0, 7 \tag{4.1c}$$

$$0, 0, 0, 8, 0, 0, 0, 8 \tag{4.1d}$$

$$99, 111, 117, 110, 116, 114, 121 \tag{4.1e}$$

$$80, 111, 114, 116, 117, 103, 97, 108 \tag{4.1f}$$

Creating binary is implemented with following rules:

- An integer is always 64 bits, with each byte serialized in sequence. The receiver will recover the integer in the same way.

- Booleans can use just two different tags. There is no size or content, since the tag says it all needed information.
- Strings use a UTF-8 encoding, since it is already byte oriented. We decided which byte of each character goes first, if it has more than one byte.
- Structures. Sequence of fields, in which, for each field, you should include: name (a string) and the component proper (serialized according to its type of resource)

Objects with variables should be serialized to a compound data (composed of inner data). This means having a compound data type, with its own tag, and inner components serialized according to their own data type (composition can be recursive). This is always recoverable at the receiver, with advantage of the tag.

4.3 Compliance and Conformance

The other idea in this technology is using compliance and conformance instead of sharing the same schema. There are no schemas or data types that need to be shared or agreed upon beforehand. This means that two partners will be able to communicate as long as the sender complies with receiver which follow what the receiver requires and the receiver conforms to what the sender expects (supports all the features that the sender requires), just in the features actually invoked by the sender.

When a resource receives a message, it tries to match it against the type of the input parameter of each of its operations. If one is found, the message received is partially assigned to that argument and the operation invoked. Partial assignment means that the input parameter is created first, then each matching component (by name) of the message is assigned to the parameter. This means that some components of the message can be ignored and some components of the argument may stay unassigned (these are the components that do not match). If all non-optional components in the argument are matched, there is compliance, the matching components of the message are assigned to those of the argument with the same name and those that do not match are ignored. Thus the designation partial assignment.

Using the simple matching that we described above to avoid complexity in the implementation and still achieve interoperability based on compliance and show that two different messages can still match a given operation's argument.

When we try to explain this with our example. After message is created by your phone then transferring the array of bytes from sender to receiver requires a binary channel (Web Sockets or HTTP2). We still can use HTTP but we need to use BASE64 to overcome the limitation of HTTP of not supporting binary data. we can serialize everything to the binary, then encode in BASE64 and decode at the receiver to recover the binary data. When message arrives the receiver(weathercast provider) then it tries to match against the type of the input parameter of each of its operations. If one is found, the message

received is partially assigned to that argument and the operation invoked to create response message. That response will be in binary format and send back to mobile phone via binary channel. A application will display that message to user.

4.4 No static data binding

The other idea is eliminating the need for static data binding (generation of stubs based on a schema). The receiver has always a default value for the message, and only the message components that comply with what the receiver expects are assigned to the matching receiver argument's components. Matching is done by name. This means that the argument is either primitive (int, bool, string) or structured, with named components. Matching is either the same type (primitive types) or name (if the argument is a structured type). Only the components that match are assigned to the formal argument of the operation.

As we seen provious section, we dont need WSDL file or generating a interface based on schema in our implementation. Messages do not obey some external schema. Each message has one specific value which are structured or primitive with its own exclusive schema that is nothing more than a self-description, without the value variability that a type exhibits. This value and its description can be validated against an infinite number of schemas, those that have this particular value included in the set of their instances.

4.5 Anotations

To make compliance possible, we should also serialize the formal argument of each operation. These should also support optional components which use the formal argument component if none in the message matches it. Therefore, the serialization methods in the static serialization class should include whether it is optional. The best is for each primitive data type to have two tags, one for mandatory and another for optional. Annotations allow us define tags for primitive data type. Messages sent use only mandatory values. Serialized formal arguments can use both mandatory and optional.

Chapter 5

Implementation

5.1 Overview

In this section we describe the implementation of the Binary Level Serialization, Central Serializer, Receiver, Message Transportation and Deployment.

5.2 Binary Level Serialization

When we try to work with different computer language and their object types we had problem with serialization. We can not use standard Java or .Net(C#) object serialization, because different program languages uses different algoritms for serialization that's why we'll run into the problem. for example in our case Java and .Net does'nt use same serialization algoritm if you serialize an object in Java to binary format and then if you try to deserialize it in .Net language you will have problem because .Net will not recognize that binary format while deserializing.For example Person object binary representation is not the same in .Net(Table 5.1) and Java(Table 5.2)

```
1
2   public class Person{
3         public String name;
4         public int age;
5   }
```

Listing 5.1: Person Object

In this case we will use our algoritms instead of using stardard serializer for Java or .Net that both languages could understand and easily serialize or deserialize.

The binary format is always the result of serializing data in each language, with a tag, the number of bytes that follow and the serialized content. Recovering the serialized data is simply testing the tag to find the data type and then using the number of bytes and the serialized content. So instead of using

0	1	0	0	0	255	255	255	255	1	0	0	0	0	0
0	0	12	2	0	0	0	73	83	101	114	105	97	108	105
122	97	116	105	111	110	32	116	101	115	116	44	32	86	101
114	115	105	111	110	61	49	46	48	46	48	46	48	44	32
67	117	108	116	117	114	101	61	110	101	117	116	114	97	108
44	32	80	117	98	108	105	99	75	101	121	84	111	107	101
110	61	110	117	108	108	5	1	0	0	0	25	83	101	114
105	97	108	105	122	97	116	105	111	110	95	116	101	115	116
46	80	101	114	115	111	110	2	0	0	0	21	60	110	97
109	101	62	107	95	95	66	97	99	107	105	110	103	70	105
101	108	100	20	60	97	103	101	62	107	95	95	66	97	99
107	105	110	103	70	105	101	108	100	1	0	8	2	0	0
0	6	3	0	0	0	4	74	111	104	110	32	0	0	0
11														

Table 5.1: .Net Serialization Person Object

-84	-19	0	5	115	114	0	23	106	97	118	97	97	112	112
108	105	99	97	116	105	111	110	55	46	80	101	114	115	111
110	79	-70	94	85	-31	-32	-110	90	2	0	2	73	0	3
97	103	101	76	0	4	110	97	109	101	116	0	18	76	106
97	118	97	47	108	97	110	103	47	83	116	114	105	110	103
59	120	112	0	0	0	32	116	0	4	74	111	104	110	

Table 5.2: Java Serialization Person Object

stardard serializer with this algoritm we could serialize object in a language and deserialize with another language.

Objects with variables should be serialized to a compound data (composed of inner data). This means having a compound data type, with its own tag, and inner components serialized according to their own data type (composition can be recursive). This is always recoverable at the receiver, thanks to the tag.

In following section CentralSerializer class will help us to seriliaze objects to binary level using primitive type object serialization.

5.3 Central Serializer

A message to be sent should be an object (in Java or .Net) that should provide a serialization method, which basically builds a serialized message (an array of bytes) by sucessively adding each of its components, serialized. This is done by invoking the methods of the serialization class for primitive data (ints, bools, etc), or by recursively invoking the serialization method of structured objects that constitute the message.

The idea of centralizing the “object of primitive type to a sequence of bytes” and vice-versa methods

in a single class is to avoid the need for every class to have these methods. Since they are static (they receive an object and return bytes, or vice versa), they can simply be concentrated in a static class (no instances) and invoked from anywhere.

The methods to serialize can receive a primitive object (integer, Boolean, etc) and an array of bytes, returning the array of bytes with the serialized object's bytes appended. Therefore, each serialization method grows the byte array. When the user wants to send a message, that message needs to be an instance of a class that has a method that knows how to serialize it, by invoking the serialization methods of the static class for each of its variables. This needs to be programmed by the use and is similar to the programming language's serialization methods.

We will show some examples of primitive type serialization.

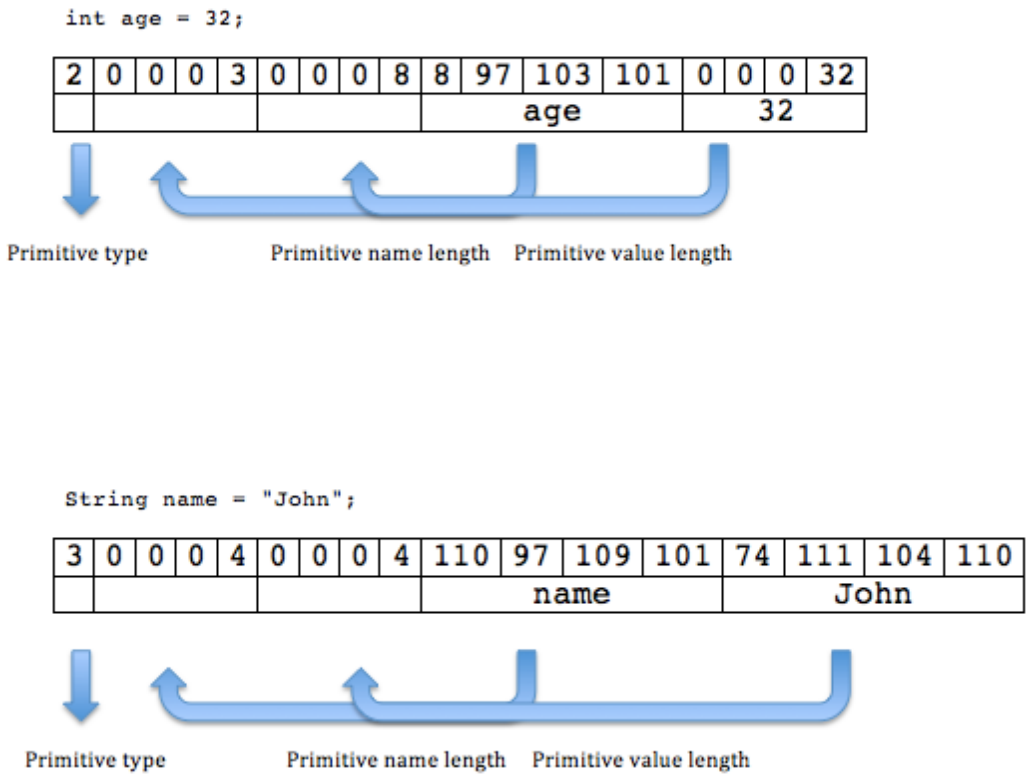

Figure 5.1: Examples of primitive type serialization.

5.4 Message Transportation

Transferring the array of bytes from sender to receiver requires a binary channel like Web Sockets(HTTP2) or more classical way by encoding and decoding the binary array with BASE64 and then use typical HTTP-based solutions (Web Services or REST). The classical solution is of course non-optimal com-

pared to the other ones, but it can easier to implement given existent tools.

Message Transportation in this solution is done with essentially JavaScript and WebSockets [14], to circumvent some of the limitations of HTTP. The most important issue is that finally humans are also becoming direct Web producers and not merely clients of servers.

Web Sockets [44] are fundamental in the efficient support for binary data removes this restriction, increases performance. They use the protocol upgrade feature of HTTP and provide a substantial degree of compatibility with existing systems. Part of the HTML5 world, servers and firewalls are increasingly supporting them and removes this restriction, adds binary support and increases performance.

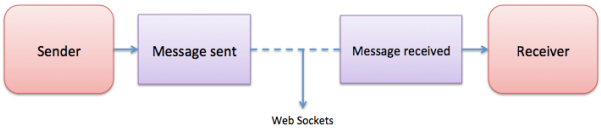

Figure 5.2: Message Transportation.

5.5 Receiver

The Receving platform makes compliance tests in binary because binary format to be able to traverse both the received message and each operation's formal argument, previously serialized. Only the components that match are assigned to the formal argument of the operation.

Two partners will be able to communicate as long as the sender complies with receiver and the receiver conforms to what the sender expects and it supports all the features that the sender requires, again just in the features actually invoked by the sender

When the receiver finds a suitable operation, it builds another array of bytes from both the message and the formal argument then receiver matches components in the formal argument, if it matches it uses the message ones but if not matched by any in the message, use the ones already in the formal argument

The receiver has always a default value for the message, and only the message components that comply with what the receiver expects are assigned to the matching receiver argumentís components. Matching in your case is done by name. This means that the argument (just one) is either primitive (int, bool, string) or structured, with named components. Matching is either the same type (primitive types) or name by name (if the argument is a structured type).

Then, the platform invokes the operation after generating the argument's object from array of bytes. This can be done either by reflection or by invoking a constructor defined by the user in the class of the formal argument, which knows how to build itself from an array of bytes that respects its components (the new array of bytes respects this because it was created by copying the original array of bytes of the serialized formal arguments, replacing only the components that matched).

5.6 Compliance

Compliance is done at the binary level, with primitive components, as we described in previous section. Only the components that match are assigned to the formal argument of the operation.

We should also serialize the formal argument of each operation to make compliance possible. These should also support optional components which use the formal argument component if none in the message matches it. Therefore, the serialization methods in the static serialization class should include the name of the component, whether it is optional (a boolean), the type (encoded in the tag) and the value. For each (primitive) data type can have mandatory annotation. Messages sent use only mandatory values. Serialized formal arguments can use both mandatory and non-mandatory.

To clarify the compliance, the idea is to serialize the argument (only one, but it can be structured) of each operation of a service (receiving object). This acts like a default value, against which the message received is matched. A service receiving a message matches it against the argument of each operation, until it finds one, assigns the message to that argument (partial assignment, see below) and runs the operation, returning eventually a result.

When checking for compliance, a component in the message with the same name as a component in the operation's argument matches it and can be assigned to it (if the message complies with the argument, as a whole). If not, it is ignored. If a component in the argument is not matched by any of the message's components and is optional , retains the argument's value (which acts as a default value). If all non-optional components in the argument are matched, there is compliance, the matching components of the message are assigned to those of the argument with the same name and those that do not match are ignored. Thus the designation partial assignment.

5.7 Deployment

The solution is developed and in two different language. They are .NET and Java and deployed Microsoft Azure Cloud. That's way we can have 2 different provider and .Net client can send message to Java provider over the cloud by using WebSockets and also Java client can send message to .Net provider. Using Microsoft Azure Cloud allowed us to test project in cloud enviroment

Chapter 6

Comparison with existing technologies

6.1 Overview

Current integration technologies for distributed systems in Cloud environment are generally supported on textual data description languages (XML and JSON) and HTTP protocol. These technologies especially designed for human-level interaction and that creates integration problems. In this section we will do comparison qualitatively and quantitevely with these current technologies and our new approach.

6.2 Qualitatively Comparison

6.2.1 Textual or binary representation

Current integration technologies for distributed systems in Cloud environment are generally supported on textual data description languages (XML and JSON) and HTTP protocol. These technologies especially designed for human-level interaction and that creates integration problems. In this section we will do comparison with these current technologies and our new approach.

SOA is usually implemented by Web Services with WSDL, which is a set of conventions on XML usage to describe services at the interface level and SOAP as a message protocol, which is again based on XML.

Many developers found SOAP cumbersome and hard to use. For example, working with SOAP in JavaScript means writing a ton of code to perform extremely simple tasks because you must create the required XML structure absolutely every time. one perceived disadvantage is the use of XML because of the verboseness of it and the time it takes to parse.

REST also requires that data types, which are usually, called media types and standardized or previously agreed, when they are application specific. REST doesn't have to use XML to provide the response. You can find REST-based Web services that output the data in Command Separated Value (CSV), JavaScript Object Notation (JSON) and Really Simple Syndication (RSS). The point is that you can obtain the output you need in a form that's easy to parse within the language you need for your application. While this may seem like it adds complexity to REST because you need to handle multiple formats, JSON usually is a better fit for data and parses much faster. REST allows better support for browser clients due to its support for JSON.

SOA and REST use textual representation but textual representation brings parsing expenses and poor support for binary data. Instead of using textual representation, the binary representation provides native support for binary data, has a smaller length and is faster to parse.

Our approach is designed to work with binary representation. The binary representation uses a modified version of the TLV format (Tag, Length and Value) used by ASN.1 [24]. This not only supports the direct integration of binary information but also facilitates parsing, since each resource, primitive or structured.

6.2.2 Message protocol

SOAP is language, platform, and transport independent SOAP can use almost any transport to send the request, using everything from the afore mentioned to SMTP (Simple Mail Transfer Protocol) and even JMS (Java Messaging Service), but REST requires use of HTTP/HTTPS.
Our approach does not depend on any particular transport protocol, relying only on message delivery. Any existing server can be used, based on HTTP, WebSockets or any other protocol. In fact, several servers can be used simultaneously, receiving messages that are handed over to the message handlers that are able to process them.
New protocols, such as WebSockets, reduce some of the problems. Web Sockets, now part of the HTML5 world, removes this restriction, adds binary support and increases performance. XML is verbose and complex, has limited support for binary formats, is inefficient in computer terms due to parsing and exhibits symmetric interoperability, based on both sender and receiver using the same schema, which constitutes a relevant coupling problem.

6.2.3 The interoperability problem

In both SOA and REST, interoperability is achieved by using common data types (usually structured), either by sharing a schema (i.e., WSDL files) or by using a previously agreed data type (typical in RESTful applications). There is usually no support for partial interoperability and polymorphism in distributed systems.

The basis of data interoperability with XML and JSON is schema sharing(at runtime or with previously standardized or agreed upon internet media types). Both the producer and consumer (reader) of a document should use the same schema, to ensure that any document produced (with that schema) can be read by the consumer. This means that both producer and consumer will be coupled by this schema. Schemas must be shared between interacting Web Services, establishing coupling for all the possible values satisfying each schema, even if they are not actually used. In this case, a reference to a schema acts like its name.
REST also requires that data types (usually called media types) must have been previously agreed, either standardized or application specific.
Searching for an interoperable Web Service is usually done by schema matching with similarity algorithms [25] and ontology matching and mapping [18]. This does not ensure interoperability and manual adaptations are usually inevitable.
In our solution, we propose to use partial interoperability, based on the concepts of compliance and conformance. It introduces a different perspective, stronger than similarity but weaker than commonality (sharing). The trick is to allow partial interoperability, by considering only the intersection between what the consumer needs and what the provider offers. It allow us for increased interoperability,adaptability and changeability, without the need to have resource types necessarily shared or previously agreed. Building interoperability on compliance and conformance avoids the problem of having to define schemas as separate documents and to agree upon them beforehand. As long as compliance and conformance hold, any two resources can interoperate, even if they were designed unawares to each other.

6.3 Quantatively Comparison

TO BE COMPLETED

Chapter 7

Conclusions

TO BE COMPLETED

7.1 Future Work

TO BE COMPLETED

Bibliography

[1] A. L. Zielinski, Kurt Geihs. *New Developments in Distributed Applications and Interoperable Systems*, page 327. Springer, 2006.

[2] M. Villari. *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice: Theory and Practice*, pages 58,91,119,269. IGI Global, 2012.

[3] R. Englander. *Java and SOAP*, page 213. O'Reilly Media, Inc., 2002.

[4] Definition of Interoperability. `http://interoperability-definition.info/en/`, 2016. [Online; accessed 24/05/2016].

[5] L. G. Nikos Antonopoulos. *Cloud Computing: Principles, Systems and Applications*, page 15. Springer Science, Business Media, 2010.

[6] N. M. Josuttis. *SOA in Practice: The Art of Distributed System Design*, page 221. O'Reilly Media, Inc, 2007.

[7] D. T. L. Deborah Nolan. *XML and Web Technologies for Data Sciences with R*, page 73. Springer Science - Business Media, 2013.

[8] M. D. Hansen. *SOA Using Java Web Services*, page 34. Pearson Education, 2007.

[9] S. P. Jim Webber and I. Robinson. *REST in Practice: Hypermedia and Systems Architecture*, page 12. O'Reilly Media, 1^{st} edition, 2010. ISBN-13: 978-0596805821.

[10] S. P. Jim Webber and I. Robinson. *REST in Practice: Hypermedia and Systems Architecture*, page 5. O'Reilly Media, 1^{st} edition, 2010. ISBN-13: 978-0596805821.

[11] S. Sakr. Xml compression techniques: A survey and comparison. *J Comput Syst Sci*, pages 303–322, 2008.

[12] S. Sakr. Encoding and compression for the devices profile for web services. *Proc. 24th International Conf. on Advanced Information Networking and Applications Workshops*, pages 514 – 519, 2010.

[13] Efficient XML Interchange Working Group. `https://www.w3.org/XML/EXI/`, 2016. [Online; accessed 24/05/2016].

[14] J. Delgado. *Service Interoperability in the Internet of Things*, pages pp 51–87. Springer Berlin Heidelberg, 2013.

[15] A. Binstock. After XML, JSON: Then What? http://www.drdobbs.com/web-development/after-xml-json-then-what/240151851, 2013. [Online; accessed 24/05/2016].

[16] I. Grigorik. High Performance Browser Networking. http://chimera.labs.oreilly.com/books/1230000000545/ch12.html, 2013. [Online; accessed 24/05/2016].

[17] A. Denis. Will WebSocket survive HTTP/2? http://www.infoq.com/articles/websocket-and-http2-coexist, 2016. [Online; accessed 30/05/2016].

[18] S. P. Euzenat J. *Ontology matching*. Berlin. Springer, 2^{nd} edition, 2007. ISBN: 978-3-642-38720-3.

[19] WebSockets – A Quick Introduction and a Sample Application. https://blog.idrsolutions.com/2013/12/websockets-an-introduction/, 2013. [Online; accessed 24/05/2016].

[20] C. G. Zyp K. A JSON Media Type for Describing the Structure and Meaning of JSON Documents. http://tools.ietf.org/html/draft-zyp-json-schema-03, 2011. [Online; accessed 24/05/2016].

[21] F. A. Natallia Kokash. Formal behavioral modeling and compliance analysis for service-oriented systems. *Springer Berlin Heidelberg*, 3(9-10):21—41, Oct. 2009. doi:10.1007 978-3-642-04167-9 2.

[22] W. S. Dae-Kyoo Kim. An approach to evaluating structural pattern conformance of uml models. *Conference: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea*, 2007. DOI: 10.1145 1244002.1244305.

[23] Serialization Performance comparison(XML,Binary,JSON,P...). http://maxondev.com/serialization-performance-comparison-c-net-formats-frameworks-xmldatacontractserializer-xmlseria 2016. [Online; accessed 24/05/2016].

[24] Olivier.Dubuisson. *Communication Between Heterogeneous Systems*. OSS Nokalva, 2^{nd} edition, 2000. ISBN 978-3-642-38721-0.

[25] C. H. L. J. Jeong B, Lee D. A novel method for measuring semantic similarity for xml schema matching. *Expert Syst with Appl*, 34:1651—1658, 2008. doi:10.1016/j.eswa.2007.01.025.