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. Rui António dos Santos Cruz

May 2016

Abstract

In Cloud environment, distributed programming solutions which cover interoperability between heterogeneous systems are essentially based on SOA or REST technology that uses 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 more due to its simplicity than due to its mapping the paradigm of services. However, both of them is not very efficient because they are based on HTTP or text-based data description languages (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 and does not require schema and uses Web Sockets as the runtime environment and these Web Sockets are much more efficient than HTTP, while maintaining some compatibility.

New solution is developed for distributed programming in cloud computing environment to develop distributed applications. While this solution compared with classical solution which is based on SOA and REST, this solution uses asymmetric interoperability which is based on compliance and conformance. The aim of this work is to explain this solution and its execution environment and also to compare it with the corresponding solution technologies which are based on SOA (Web Services) and REST.

Contents

List of Tables

List of Figures

x

Chapter 1

Introduction

1.1 Overview

This chapter gives information about the subject of the thesis, the area of interest with that subject and also general information about current solutions related that topic.

1.2 Context

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 an 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 involves distributed platforms and heterogeneous components.

On the other hand, 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.

The subject of thesis research current solutions of distributed applications and their implementation in cloud environment and propose the design of the alternative solution that is both simpler and more effective than existing ones.

Taking your Android mobile phone as an example, nowadays a huge part of the people use smartphones that are 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 the 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.

1.3 The problem

Distributed applications need to interact among themselves. In this respect, they need to solve the interoperability problems as any two systems that need to understand each other to achieve meaningful and useful collaboration. The traditional integration technologies, based on either SOA or REST as seen in Figure 1.1, 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 connectionless protocol (HTTP) and XML, JSON (and by extension SOA and REST) that do not support binary in its main features, because the solutions were also based on text (XML or JSON) and contextual information, are limited for many applications.

The main problem of the traditional integration technologies creates coupling between the provider and the consumer, because both customer and provider are forced to implement full interoperability (for example, sharing a XML schema). This leads to a greater coupling than needed.

While solving data interoperability, the problem of using XML or JSON is that XML or JSON is inefficient in computer terms due to parsing data to build a DOM tree and validating schemas to validate structure of message.

The problem can be explained with an example, as described in the previous example in section1.2 that you can have two different technologies and they can communicate each other. In case if they use SOA architecture style (namely, SOAP Web Services) you need to use XML (Extensible Markup Language), which both languages can understand easily since, it is a common language. But still you 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 you have XML message that can be read by your mobile phone or weathercast provider and parsed to build a DOM tree to be used. That can be done with sharing same schema to know structure of message. There are some problems with sharing schema which brings coupling problem between systems. 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. The created
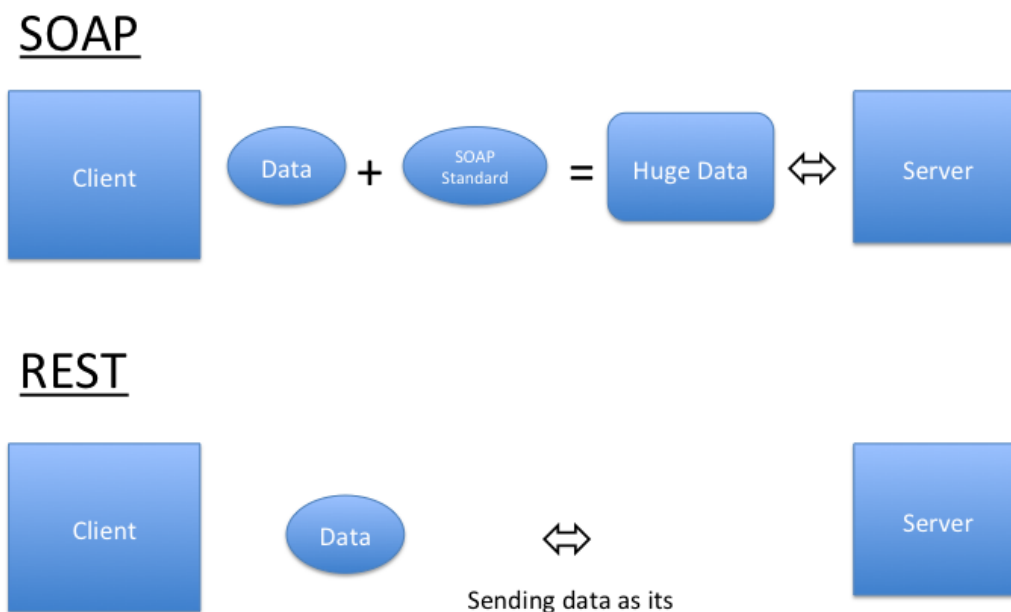
SOAP

REST

Figure 1.1: SOAP and REST Approach.

message should be validated by schema because if it doesn't obey the schema rules, the provider can not understand the message. When message arrives to provider it validates message regarding the same schema and parses that XML message to own objects to understand it then provide response regarding the query, which is done by your phone application. There are some problems with using XML schema because your phone application or weathercast provider can not alter this schema without informing each other. That creates coupling between mobile application and weathercast provider. After message creation by weathercast provider, this response again needs to be converted to XML message using stub generator and send that response message to your phone. Now, your phone uses XML schema and XML parsers to understand XML message and uses these data to display information to you. As seen in that example it is not very easy to communicate systems that use totally different technologies. In case of REST, you can use different message format instead of XML, for example, JSON which is easier to parse and less heavier than XML, but in REST both languages need to know message structure, then they can parse and understand.

In case of REST, there are some differences. If your weathercast application uses REST to communicate with weathercast provider then you don't need to create XML message to send weathercast provider. You can use other formats to send it using HTTP to server. Assume that you 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) because in REST, different HTTP methods can be used with using the same URI. When a message arrived to weathercast provider this message

3

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. REST may seem more flexible, in the sense that, if the server changes the links it sends in the responses, the client will follow this change automatically by using the new links. However, the problem is that this is not as general as it may seem, since the client must be able to understand the structure of the responses. To achieve this, REST imposes the constraint that returns representations using standardized or pre-agreed media types. Your mobile phone application or weathercast provider can not alter this structure of JSON without informing 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.

Current integration technologies, based on either SOA or REST, is that they use a data description language as the representation format and schema sharing as the interoperability mechanism. Other factors, using a connectionless protocol (HTTP) and the lack of native support for binary data and contextual information limit for many applications, although they are not impeditive.

The main problems with using traditional integration technologies can be summerized as follows:

- Data interoperability problems based on XML and JSON (Stub generation,DOM parsing)

- Service interoperabiliy problems based on SOA and REST(Coupling with schema sharing)

- The underlying protocol, usually HTTP, without binary support

These are the main problems with using traditional integration technologies. In section 1.4, an alternative solution will be proposed to solve described problems.

1.4 The solution

As a solution to the problems mentioned above like creating coupling between the provider and the consumer, because both customer and provider are forced to implement full interoperability (sharing schema), now think a new solution way with providing the maximum decoupling possible while ensuring the minimum interoperability requirements and allowing the client or the sender to changebility, as long as the actual used parts do not change. The current solutions that are described in section 1.3 use schema sharing to understand message structure with created unnecessary coupling, and what about using compliance (consumer must satisfy the requirements established by the provider to accept requests sent to it)[4] and conformance (the provider must fulfill the expectations of the consumer regarding the effects of a request)[5] instead of sharing the same schema. 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 unbeknownst to each other. Another solution for performance instead of using XML or JSON, based on text which is heavy and hard to parse, uses binary directly for message transportation which reduce complexity improve performance. Again for performance, regarding message transportation, uses Web Sockets and HTTP/2 protocol (a binary protocol

with small message frames) instead of using classical HTTP. Following chapters give more details and examples about the topics that are mentioned in current section.

1.5 Contributions

The 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.

The contributions are made in dissertation as follows:

- Trying a new method for interoperability problem with using compliance and comformance which is not known and used very well in the market.

- The solution aimed to combining best features of SOA and REST.

- The solution will be demonstrated through experiments. Showing results that it is a better solution for application interoperability and also it is easier to implement. Also assessing performance to show advantage of using binary, with comparison current solutions.

1.6 Organization

The organization of dissertation is prepared 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 will be used, 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 will be used 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[6].
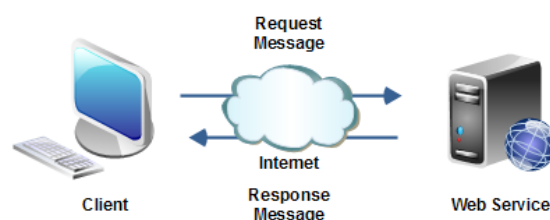

Figure 2.1: Simple Web Service Architecture.

7

For example let's say you have written web service in Java and let's say you have another web service that is 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 a 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 allows you that you can actually pick and choose technologies that you want to use. For example, 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 is 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 are 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 is 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 the 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 popularly. Next section focalize on SOAP and REST web services.

8

2.3   SOAP Web Services

Simple Object Access Protocol, or SOAP as it was the first attempt to standardize a web service inter-face. 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[7]. The specification does not dictate the transport over which the message should be sent, but most implementations send the XML message over HTTP.

Now it will be explained shortly SOAP web service by an example. Let's start an example with Java application. Let's say you have implementation class and you want to share this implementation class with other developer projects. How you 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 you have web service implementation and you want to share details of this web service to consumers. Is it works with an interface? Probably it will not work because as 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 you give this .Net application a Java interface then most probably it will not work because they are different technologies, so the technology that you 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 you do is in case of web service, you actually share that contract as an XML document. This XML document is actually called as WSDL(Web Services Description Language)[8].

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[9] 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 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
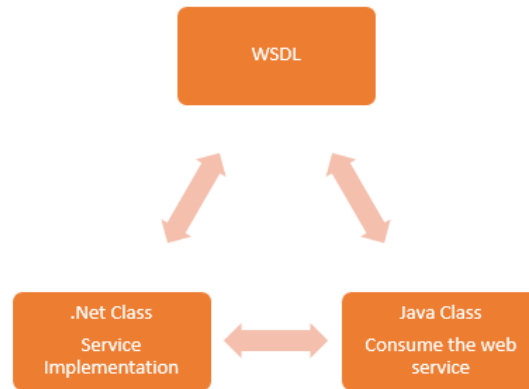
Figure 2.2: WSDL (Web Services Description Language).

.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 send 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 use and this XML is called SOAP(Simple Object Access Protocol). It is a way in which these different technologies can access object 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.

Figure 2.3: SOAP (Simple Object Access Protocol).

Now you know what is the mechanism, you know what need to be send and you know how to send

which uses 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 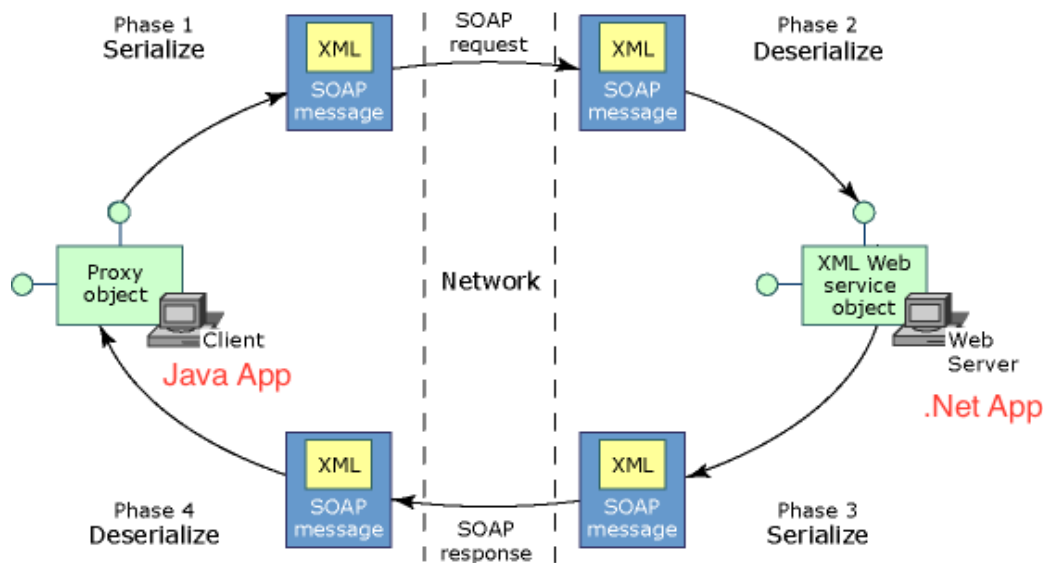done by SEI(Service Endpoint Interface)[10]. The SEI access interface to your web service endpoint so you have an interface at your client app to the service endpoint which translates all web service call to a SOAP message and then it makes sure that the other things is able to understand this message. So you don't have to write this class and all the conversion because you can have it automatically generated for you. When you make a web service call you don't worry about where the web service is. When you need to call, all you need to do is having 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 calls the same web service, so you will have SEI for .NET that knows to convert .Net objects to SOAP message.

2.4 REST Web Services

REST (Representational State Transfer) was created in 2000 by Roy Fielding[11]. 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 to be exposed 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[12]. 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 your 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 account on Facebook, you will get a profile page that obivously dynamically generated pages, so whenever there is a new profile, it is basically the same page which does the same processing but renders different content depending on profile that you are watching. In REST web services you need to think of resources and create unique URIs 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.

In REST web services, with same unique URI that can be done different actions, for example if you are an administrator of weathercast application and you want to get data of Lisbon city or if you want to

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.

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. This can be implemented with only a single concrete URI, a single URI template, and four HTTP verbs. In fact, it's so compact that can be provided 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.
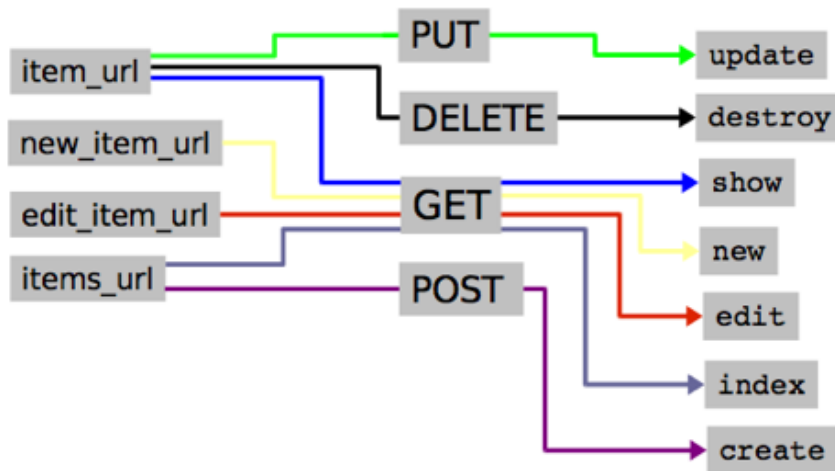
Figure 2.4: REST URIs and HTTP methods.

You've seen the requests so far, and understood what resource URIs and HTTP methods. You can make request to resource URIs with using HTTP methods. Let's switch to responses now. After making request, 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 gives 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 with the format (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[13], such as EXI (Efficient XML Interchange), BSON and others [14]. 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 is maintained the text markup style of HTML, is made computer based clients easier and it became the cornerstone of Web Services technologies together with HTTP. This evolutionary transition is perfectly understandable in market and standardization terms, but it 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 Listing 2.2 of in Listing 2.1 Person Object. XML has generalized HTML with 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 offer compression at less cost than using Gzip, and save processing, otherwise it is needed to decompress compressed XML[15].

```
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[16], 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 a 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 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 [17]. As 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 are needed 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 [13][14], but this does not reduce the parsing time, since text needs 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.[16]

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 reduce 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[18].

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[19].

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 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[20] 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, that 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 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:[21]

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 HTTP/2 compared against WebSocket[19], A lot of similarities can be seen 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 is 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. While using data formats like XML and JSON, The systems need to work with this document which is sent by server or client side to understand message. Another problem is using XML or JSON to solve data interoperability is inefficient in computer terms due to parsing data to build a DOM tree and validating schemas to validate structure of message. This chapter gives detailed information about these interoperability issues.

3.2   Service Interoperability

Currently, enterprise integration is based on SOA with Web Services and REST with HTTP; these are 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. As seen in Figure 3.1, 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, therefore it imposes coupling between the resources for all its possible values, even if only a

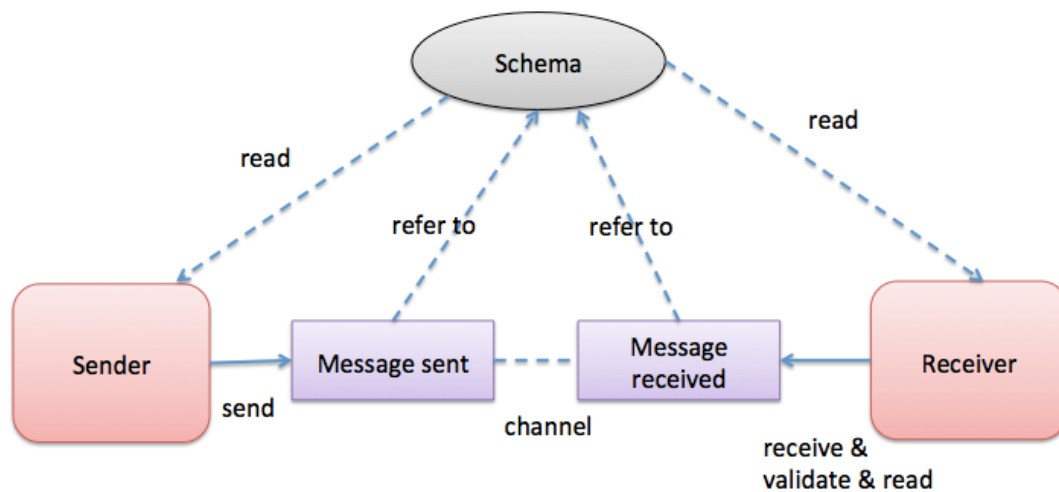few are actually used. In either case, data types need to be fully known by both interacting resources.

Figure 3.1: Interoperability in SOA and REST.

3.3 Data Interoperability

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 is serialized, sent and reconstructed upon reception of the corresponding message. Most used data types for SOA and REST are XML and JSON. Section 2.5 gives detail information about advantages and disadvantages of XML and JSON. The sender and receiver need to 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[22], 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. When using XML, there is a need to work with this document to understand message, which is sent by server or client side. The solutions are exist for that are Databinding, DOM (Document Object Model) or SAX(Simple API for XML)[23]. For example, XML data binding refers to a means of representing information in an XML document as a business object in computer memory. An XML data binder accomplishes this by automatically creating a mapping between elements of the XML schema of the document that needed to bind and members of a class to be represented in memory. Another solutions for create data or access the data in the XML are DOM or SAX. They do same work in different way. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially. In general, DOM is easier to use but has an overhead of parsing the entire XML before you can start using it.

Let's explain this with an example, referring to the example in Chapter 1 that demonstrate interaction

18

between Android mobile phone and .Net weathercast provider. In case of REST approach, An XML or JSON message needed to be created to send server. Lets see how can be created a XML message with Android phone application to send it weathercast provider. The code in Listing 3.2 explains how to create the XML message in Listing 3.1 with using Java.

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

Listing 3.1: Example XML message to send weathercast provider

```java
1       DocumentBuilderFactory docFactory =
            DocumentBuilderFactory.newInstance();
2       DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
3       // root elements
4       Document doc = docBuilder.newDocument();
5       Element rootElement = doc.createElement("weather");
6       doc.appendChild(rootElement);
7       // country elements
8       Element country = doc.createElement("country");
9       rootElement.appendChild(country);
10      // city elements
11      Element city = doc.createElement("city");
12      city.appendChild(doc.createTextNode(cityInfo));
13      rootElement.appendChild(city);
14      // date elements
15      Element date = doc.createElement("date");
16      date.appendChild(doc.createTextNode(new Date().toString()));
17      rootElement.appendChild(date);
18      // weatherResult elements
19      Element nickname = doc.createElement("weatherResult");
20      nickname.appendChild(doc.createTextNode(""));
21      rootElement.appendChild(nickname);
22      TransformerFactory transformerFactory =
            TransformerFactory.newInstance();
23      Transformer transformer = transformerFactory.newTransformer();
24      DOMSource source = new DOMSource(doc); // XML Message
```

Listing 3.2: XML message creation with Java Code

After creation of XML message with Java, here the message is created using DOM which also could be done by using SAX, this message will be sent with suitable message level protocol (e.g., HTTP) and will be received by the provider. Weathercast provider will use XML databinding, DOM or SAX techniques to deserialize the message understand query message. The code in Listing 3.3 explains how to parse an XML message using DOM in .Net.

```
1    string Country;
2    string City;
3    string date;
4    // Create an XmlReader
5    using (XmlReader reader = XmlReader.Create(new StringReader(xmlString)))
6    {
7      reader.ReadToFollowing("weather");
8      reader.ReadToFollowing("country");
9      Country = reader.ReadElementContentAsString();
10     reader.ReadToFollowing("city");
11     City = reader.ReadElementContentAsString();
12     reader.ReadToFollowing("date");
13     date = reader.ReadElementContentAsString();
14   }
```

Listing 3.3: XML message parsing with .Net Code

As seen in Listing 3.3, the receiver knows the format of XML message. Otherwise it cannot deserialize correctly without knowing structure. XML parsing code reads an XML document and uses DOM or SAX APIs to provide programmatic access to its content and structure. After parsing XML message, .Net weathercast provider builds response data as XML message and then sends to your mobile phone. Your mobile phone needs to parse that XML response message using DOM or SAX API again to display the result on your phone screen. Here in this example it is assumed that both your mobile and weathercast provider know the same schema. Example schema for your mobile phone application and weathercast provider can be as in Listing 3.4 and they both first validate message with a validator for example SAX API validator[24] as seen in Listing 3.5, otherwise the componenet can not parse XML correctly and can not communicate each other.

```
1  <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
2    <xs:element name="weather">
3      <xs:complexType>
4        <xs:sequence>
5          <xs:element type="xs:string" name="country"/>
6          <xs:element type="xs:string" name="city"/>
7          <xs:element type="xs:string" name="date"/>
8          <xs:element type="xs:string" name="weatherResult"/>
9        </xs:sequence>
```

```
10        </xs:complexType>
11      </xs:element>
12    </xs:schema>
```

Listing 3.4: Example schema for Weather XML file

```
1
2          SAXParserFactory factory = SAXParserFactory.newInstance();
3          factory.setValidating(false);
4          factory.setNamespaceAware(true);
5          SchemaFactory schemaFactory
6                  = SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
7          factory.setSchema(schemaFactory.newSchema(
8                  new Source[]{new StreamSource("weather.xsd")}));
9          SAXParser parser = factory.newSAXParser();
10
11         XMLReader reader = parser.getXMLReader();
12       // reader.setErrorHandler(new SimpleErrorHandler());
13         reader.parse(new InputSource("weather.xml"));
```

Listing 3.5: Example XML message to send weathercast provider

The same example can be implemented 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.

Webservice is developed in .NET for weather cast provider such as in Listing 3.6 and also following WSDL document will be created automatically for that web service by .NET as seen in Listing 3.7.

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

```
11          }
12     }
```

Listing 3.6: SOAP Web Service with .Net

So if your mobile application knows that WSDL document in Listing 3.7, it can easily call a method from weathercast provider by RPC (Remote Procedure Call) style to get current weathercast info for requested city as shown in Listing 3.8.

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <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/">
3      <wsdl:types>
4       <s:schema elementFormDefault="qualified"
            targetNamespace="http://tempuri.org/">
5         <s:element name="GetWeatherCast">
6           <s:complexType>
7             <s:sequence>
8               <s:element minOccurs="0" maxOccurs="1" name="weather"
                     type="tns:Weather" />
9             </s:sequence>
10          </s:complexType>
11        </s:element>
12        <s:complexType name="Weather">
13          <s:sequence>
14            <s:element minOccurs="0" maxOccurs="1" name="country"
                   type="s:string" />
15            <s:element minOccurs="0" maxOccurs="1" name="city" type="s:string"
                   />
16            <s:element minOccurs="1" maxOccurs="1" name="date"
                   type="s:dateTime" />
17            <s:element minOccurs="0" maxOccurs="1" name="weatherResult"
                   type="s:string" />
18          </s:sequence>
19        </s:complexType>
20        <s:element name="GetWeatherCastResponse">
```

```
21        <s:complexType>
22          <s:sequence>
23            <s:element minOccurs="0" maxOccurs="1"
                   name="GetWeatherCastResult" type="tns:Weather" />
24          </s:sequence>
25        </s:complexType>
26      </s:element>
27    </s:schema>
28  </wsdl:types>
29  <wsdl:message name="GetWeatherCastSoapIn">
30    <wsdl:part name="parameters" element="tns:GetWeatherCast" />
31  </wsdl:message>
32  <wsdl:message name="GetWeatherCastSoapOut">
33    <wsdl:part name="parameters" element="tns:GetWeatherCastResponse" />
34  </wsdl:message>
35  <wsdl:portType name="WebServiceSoap">
36    <wsdl:operation name="GetWeatherCast">
37      <wsdl:input message="tns:GetWeatherCastSoapIn" />
38      <wsdl:output message="tns:GetWeatherCastSoapOut" />
39    </wsdl:operation>
40  </wsdl:portType>
41  <wsdl:binding name="WebServiceSoap" type="tns:WebServiceSoap">
42    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
43    <wsdl:operation name="GetWeatherCast">
44      <soap:operation soapAction="http://tempuri.org/GetWeatherCast"
                   style="document" />
45      <wsdl:input>
46        <soap:body use="literal" />
47      </wsdl:input>
48      <wsdl:output>
49        <soap:body use="literal" />
50      </wsdl:output>
51    </wsdl:operation>
52  </wsdl:binding>
53  <wsdl:binding name="WebServiceSoap12" type="tns:WebServiceSoap">
54    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
55    <wsdl:operation name="GetWeatherCast">
56      <soap12:operation soapAction="http://tempuri.org/GetWeatherCast"
                   style="document" />
57      <wsdl:input>
58        <soap12:body use="literal" />
59      </wsdl:input>
```

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

Listing 3.7: WSDL Document for Weathercast Provider

For calling a Web service from any programming language you need the stubs (interfaces) of the web service. From client side the business objects communicates with stub objects and stub takes the responsibility form the message and invoke the web service. Hence, here also the first step is to obtain the stubs for web service. Developing client side code for given WSDL is usally automatically generated for you by using programming language. This is also called schema-first definitions that require schema to be created prior to instances. When developping code in client side, The following steps need to be completed to generate stubs code for given WSDL file in Java.

- Click to "Add new file" as seen in Figure 3.2.

- Choose file type as "Web Service Client".

- Choose WSDL URL and Copy and paste your WSDL URl adress as seen in Figure 3.3.

- Click Finish.

After these steps you can start to write client side code in Java as shown in Listing 3.8.

```
1    import CSharpWebService.Weather;
2    import CSharpWebService.WebService;
3    import CSharpWebService.WebServiceSoap;
4    public class Run {
5
6        public Weather GetWeatherCast() {
7
8            Weather  w = new Weather();
9            w.setCountry("Portugal");
10           w.setCity("Lisbon");
11           w.setDate(null);
```

```
12            WebService service  = new WebService();
13            WebServiceSoap port = service .getWebServiceSoap12();
14            return (port.getWeatherCast(w));
15        }
16    }
```

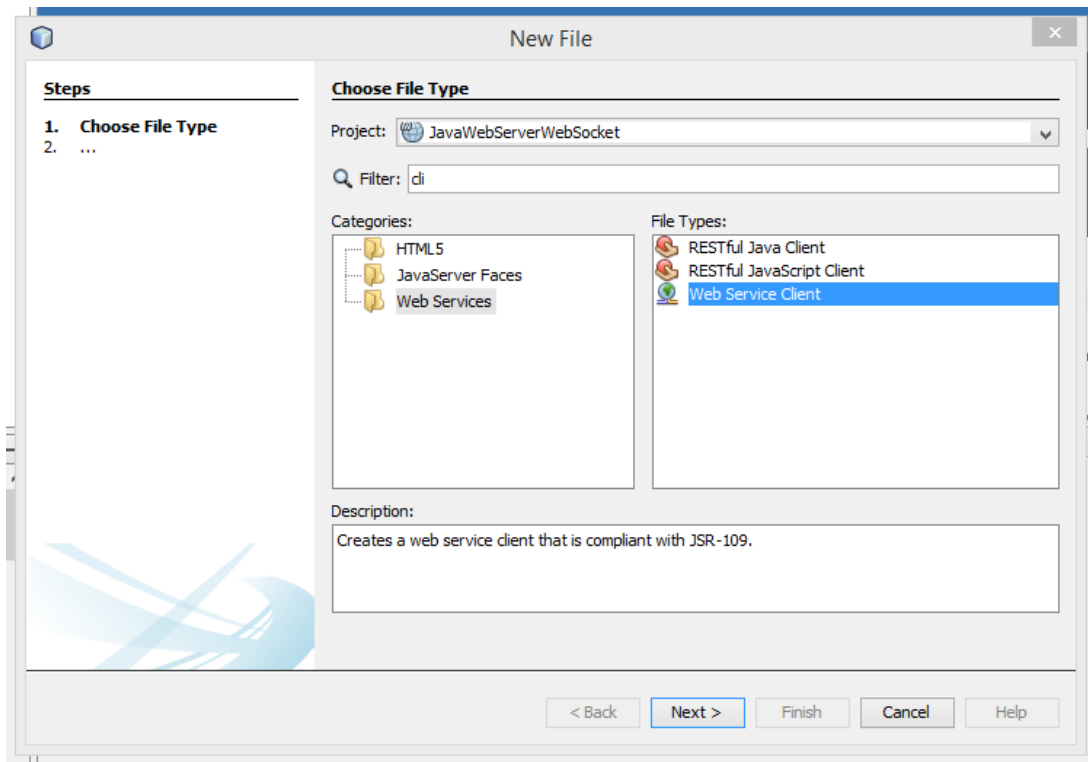Listing 3.8: SOAP Web Service Client with Java

Figure 3.2: Step 1 to generate client code from WSDL file.

As seen in Listing 3.7 for very basic example there is a big WSDL document in case using SOA architecture. Once the invoking is done by the client side, at service provider side as seen in listing 3.6, skeleton is the parallel object for stub and it receives the request message and understands it and passes on the information to service side business objects.

As seen with examples, a typical and pragmatic solution is to resort to Web Services and XML data, sharing schemas and namespaces, or to RESTful APIs, which are simpler to use and require that schemas (media types) be standardized or pre-agreed. In these technologies, both customer and provider are forced to implement full interoperability (for example, sharing a XML schema), even if only a fraction of the possible interactions is used. This leads to a greater coupling than needed. In Chapter 4, the proposed solution reduce coupling with using partial interoperability and creating the maximum decoupling possible while ensuring the minimum interoperability requirements.

Figure 3.3: Step 2 to generate client code from WSDL file.

Chapter 4

Architecture of the solution

4.1 Overview

As discussed before, the idea is to compare current technologies (Web Services, REST, XML, JSON) with new solution. This solution is a new programming technology alternative current solution. 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 being implemented the interoperability, primitive data formats and solution of binary level compliance and conformance.

4.2 Asymmetric interoperability

In the new implemented solution, it is showed that how interaction is still 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 bad, but without it no interaction is possible. The goal in that thesis 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 solution, a different approach will be showed, 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 4.1 exposes a schema that defines the values it is willing to accept. When a 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
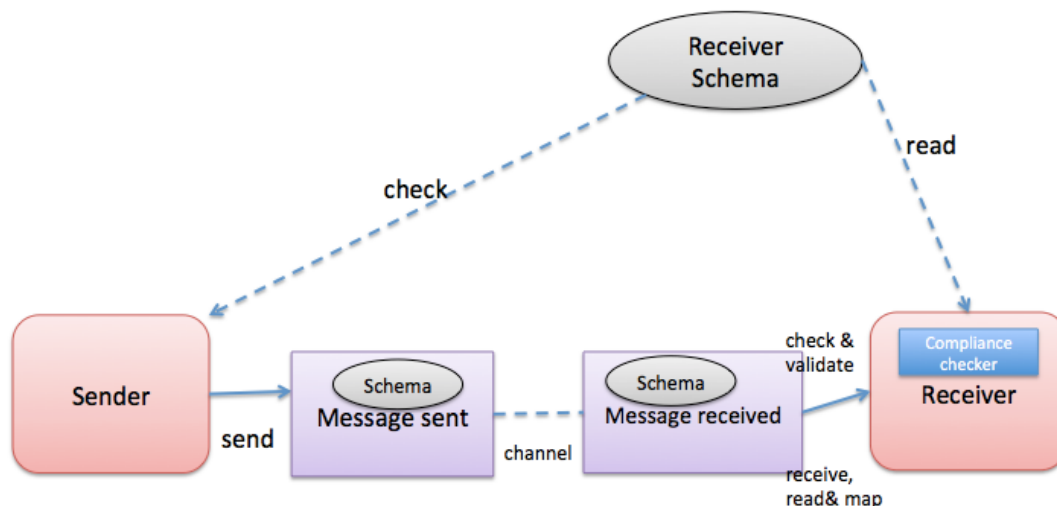
27

Figure 4.1: Asymmetric interoperability.

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, it means that there is partial interoperability.

Figure 4.2 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[4] 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[5] 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
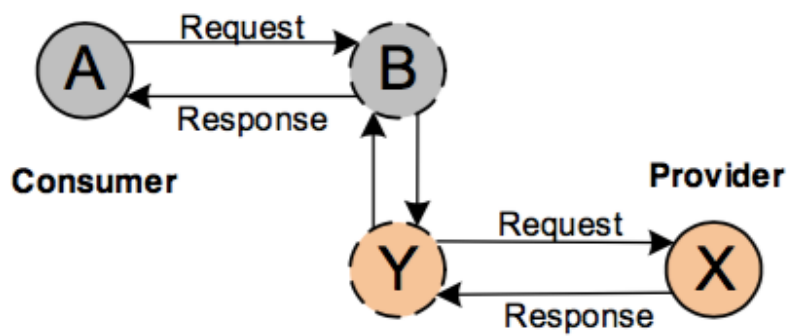
28

Figure 4.2: Partial interoperability, based on compliance and conformance.

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 4.2 , 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's assume that the provider of A is needed to change 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 you 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.

Referring again the example in Chapter 1 that demonstrate interaction between Android mobile phone and .Net weathercast provider. Now let's demonstrate how it works with a new implemented solution. Again your mobile phone application creates a query and then sends it to weathercast provider. After message is created by your phone application then the array of bytes of message are transferred

from sender to receiver requires a binary channel (Web Sockets or HTTP2). HTTP still can be used, but BASE64 must be used to overcome the limitation of HTTP of not supporting binary data. Everything is serialized 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. Here again your phone application is developed with Java and weathercast provider is developed with .Net technology. In implemented new solution the mobile application will create a binary message and send it to weathercast provider using Web Sockets or HTTP2 technology. The example will be explained with 4 different scenario:

Scenario 1:

Client side which is your mobile phone application has very simple weather object class such as in Listing 4.1. This object class will be serialized to binary and it will be sent to server side, which is weathercast provider. Server side has `Receiver class` and it has different methods with `AvaliableMethod` notation and the formal argument of the receiver method will be used for mapping. One of the receiver method with `AvaliableMethod` annotation has `Weather1` object as a input parameter as shown in Listing 4.3. Let's say class has primitive values as seen in Listing 5.11. Every field in that class has Mandatory notation so while mapping is happening, type of primitive, name of primitive and value of primitive must be the matched.

```
1              public class Weather
2                {
3                   public String country = "Portugal";
4                   public String city = "Lisbon";
5                }
```

Listing 4.1: Simple weather object class

```
1          [AvaliableMethod]
2          public void MakeObjectC(Weather1 test)
3          {
4          }
```

Listing 4.2: AvaliableMethod notation in Receiver Class

```
1   public class Weather1
2     {
3       [Mandatory]
4       public String country = "Portugal";
5       [Mandatory]
6       public String city = "Lisbon";
7     }
```

Listing 4.3: Weather1 object in Receiver

When a message with `Weather` object is received to server it will mapped `Weather1` object as seen in Figure 4.3 since compliance is occurred and then it will run the operation and the result will be sent to your mobile application.
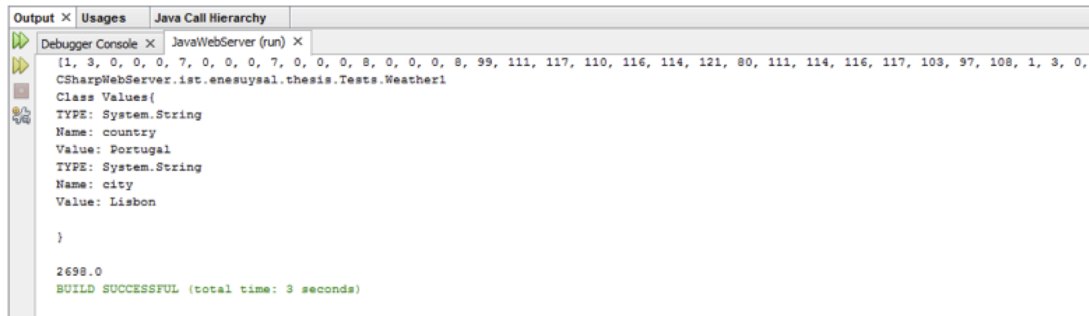
Figure 4.3: The result of Scenario 3 and 4 execution.

Scenario 2:

In that scenario, everything is very similar to previous scenario except in `Weather` class has a different value of city which is Braga instead of Lisbon as seen in Listing 4.4. In that case the result will be not be same as previous. Since city field is a mandatory field then Weather object will not mapped with `Weather1` class as in as seen in Figure 4.4 and server will not invoke any operation and response will not be returned to user phone application. But in the future if server adds one method its `Receiver` class such as `Weather3` class as seen Listing 4.6 with city name is Braga as a default then without changing anything in the code of your mobile application, mapping will be occur between your mobile application and weathercast provider and result will be returned.

```
1        public class Weather
2          {
3            public String country = "Portugal";
4            public String city = "Braga";
5          }
```

Listing 4.4: Simple weather object class

Figure 4.4: The result of Scenario 2 execution.

Scenario 3:

Now having another class in `Receiver` class which is called `Weather2` as seen Listing 4.5. `Weather2` has one mandatory field and another one non-mandatory which is optional. Lets send again the same message from mobile phone application as seen in Listing 4.7. In that case there is a value for country but it is not mandatory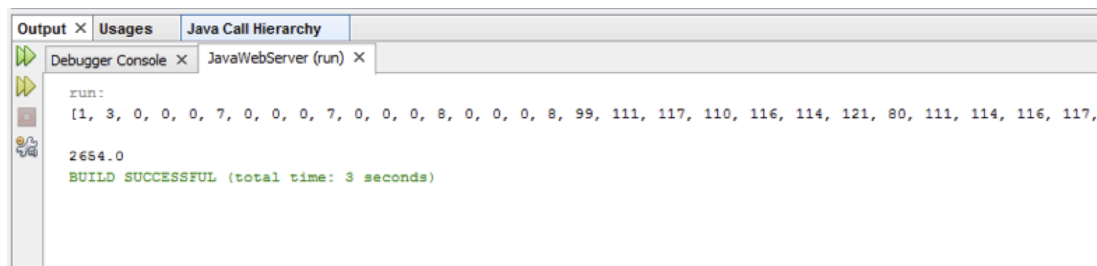. As expected `Weather` object will be mapped to `Weather2` object and server will invoke a operation as seen in Figure 4.3 and response will be returned to user phone application.

```
1   public class Weather2
2     {
3       public String country = "Portugal";
4       [Mandatory]
5       public String city = "Lisbon";
6     }
```

Listing 4.5: Weather2 object in Receiver

```
1   public class Weather3
2     {
3       public String country = "Portugal";
4       [Mandatory]
5       public String city = "Braga";
6     }
```

Listing 4.6: Weather3 object in Receiver

```
1       public class Weather
2         {
3           public String country = "Portugal";
4           public String city = "Lisbon";
5         }
```

Listing 4.7: Simple weather object class

Scenario 4:

In that scenario, there are some small changes regarding in Scenario 3. There is no country name as seen in Listing 4.8 , again using `Weather2` class in the `Receiver` where country is non-mandatory. In that case when the message is sent, it will be matched with `Weather2` because country field is not a mandatory field and server will use default value instead of the one in the message. The result will be provided as seen in Figure 4.3 and server will invoke a operation and response will be returned to user phone application.

```
1       public class Weather
2         {
3           public String country = "";
4           public String city = "Lisbon";
5         }
```

As seen in the examples, there is a possibility to have infinity number class in `Receiver` and if the message matches one of them, it runs the operation, returning eventually a result to client. This eliminates data binding parts as stub generation and DOM inspection, because the receiver needs only to be concerned with the interface, which is defined at compile time. Mapping between the message and the formal argument of the receiver method is done in binary, automatically and based on compliance.

The 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 seen in the example, WSDL file is not necessary or generating a interface based on schema in proposed 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. With using this solution, no longer need to produce a client stub, and that either the client/producer or the server/reader can be changed without breaking interoperability, as longer as compliance (on the client side) and conformance (on the server side) are respected.

To make compliance possible, the formal argument of each operation should also be serialized as seen in the example. 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 to define tags for primitive data type. Messages sent use only mandatory values. Serialized formal arguments can use both mandatory and optional.

4.3 Binary Format

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 serialization or deserialization performance is compared with binary, XML and JSON, it is easily seen that binary format gives faster speed than the others especially with large data[25].

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 representa-

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).

tion provides native support for binary data, has a smaller length and is faster to parse.

Defining a binary format which messages are serialized on send and recovered on reception. For binary format using TLV (Tag, Length and Value) binary markup[26], array of bytes are used 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 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 allows maintaining all the information necessary to communicate in a standard and platform independent way.

As long as controlling the serialization format, the serialization can be performed 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 as seen in the example of 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.

The procedure starts by serializing query message to the binary format in Java. In this case, application in your phone will create binary message instead of XML or JSON to send weathercast provider as in Listing 4.9

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

```
9    }
```

Creating a binary message will be similar to TLV (Tag, Length and Value) binary markup idea. Reagarding the code in Listing 4.9, the provided binary will be as in Listing 4.10.

```
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, 100, 97, 116, 101]
```

Listing 4.10: Creating serialized binary Message

Weather object in Listing 4.9 is serialized by its primitive elements. So in Figure 4.5, it can be seen that how first primitive element of weather object, which is, country element is been serialized to binary.
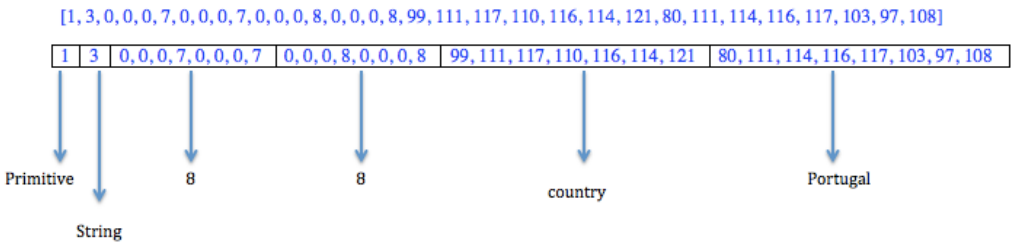
Figure 4.5: Binary representation of primitive resource.

Equation 4.1a tells that serialization starts with checking if the object is primitive or structure. After that step, equation 4.1b informs you with 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 size of field name and value is known, They can be converted to binary. Lastly, equation 4.1e and equation 4.1f tell 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 only 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.

- 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.

Chapter 5

Implementation

5.1 Overview

This chapter describes the implementation details of the Binary Level Serialization, Receiver and Compliance, Message Transportation and Deployment. This chapter also gives experimental examples about new implementation to show that solution works in practice and that is better than XML or JSON regarding data binding and decoupling.

5.2 Binary Level Serialization

In this section, the implementation of the Binary level serialization will be presented. As in previous chapters, explained advantages of binary comparing to text-based solutions, it will be showed with some experimentation.

Computer languages have their own object types and special serialization algorithms for their object types, so when you are working with same language you can easily serialize an object and deserialize it back easily with other application which is written by same language.

When starting to work with different computer languages and their object types, most probably you have problem with serialization. You cannot use standard Java or .Net(C#) object serialization, because different program languages use different algorithms for serialization, that's why you'll run into the problem with serialization. Let's explain that with a simple example. For example simple Person object such as in Listing 5.1 has a binary representation which is not the same in .Net(Table 5.1) and Java(Table 5.2).

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

Listing 5.1: Person Object

37

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

Because both Java and .Net has different algorithms to serialize and deserialize their object classes and they are unaware of each other. The unique solution for that problem is avoid using standard serialization libraries for Java and .Net.

After that experiment, creating a common algorithm for different languages is decided to be use instead of using standard 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. Each data has a tag, which describe data type and 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.

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

Instead of every class has their own serializer class, it is implemented with 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, 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 is an instance

Format	Avarage (bytes)
XML	62008
EXI	3030
JSON	32036
Binary	35984

Table 5.3: Sizes (in bytes) of several resource representations.

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.

To examine the performance in serializing structured data in binary and text-based data(XML,JSON), an experiment was designed using following hardware and software:

- Hardware: IMac(by Apple Inc.) with Intel Core i7 1,7 GHz and 8GB memory.

- Operation System: Mac OS X version 10.11.4.

- Java: version 1.8.

Current version of object serialization libraries were selected shown in the following:

- JAXB Serializer for XML serialization.

- Jackson Serializer for JSON serialization.

- OpenEXI for XML compression

The experiment was designed as follows:

- Ten kinds of query were prepared for weathercast provider. They were queries with ten different size of weather: 0, 100, 200, 300, 400, 500, 600, 700, 800, 900 weathercast query.

- The serialized file was measured and the execution time was measured using System.currentTimeMillis() shown in Listing 5.2.

```
1       long start = System.currentTimeMillis();
2       Query q = new Query();
3       for (int i = 0; i < 900; i++) {
4           q.weathers.add(new Weather());
5       }
6       json = serialize(query);
7       long end = System.currentTimeMillis();
8       double time  =  (double) (end-start);
```

Listing 5.2: Serialization program for testing

The avarage size of ten kinds of serialized files given in Table 5.3.

From the point of the average size, the largest serialized file is obtained using XML, but EXI (Efficient XML Interchange) that use compression technology has best serialized file size when compared with
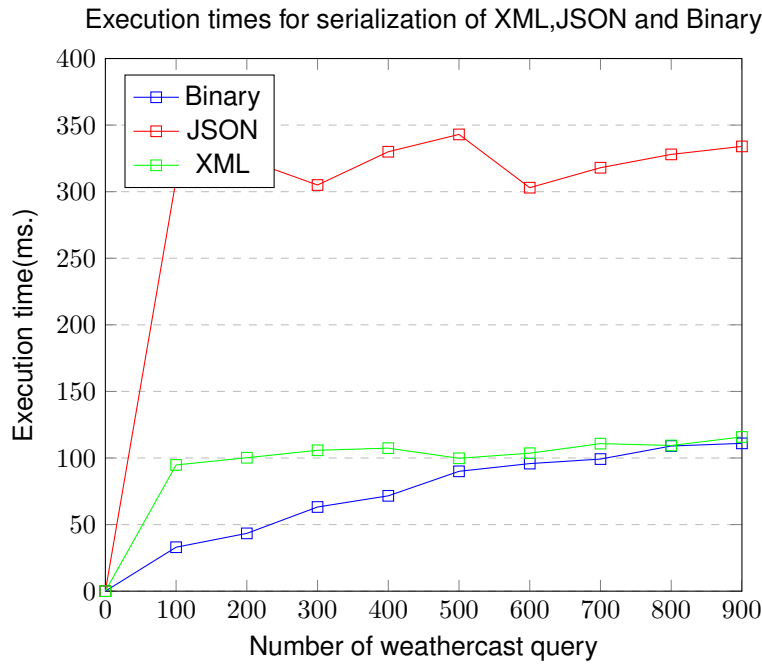
Figure 5.1: Avarage Execution times for serialization of XML,JSON and Binary

others. But it does not reduce the parsing time, since text need to be recovered. On the other hand, Binary and JSON have very similar serialized file size. From the point of execution time binary serialization spends less time when compared with the others as seen in Figure 5.1. From quantative aspects, the size of binary-based serialized data is better than XML-based and JSON-based serialization since there is no schema required and also in term of data binding, the binary-based serialization gives us big advantage with removing stub generation and DOM inspection.

5.3   Receiver and Compliance

Compliance is done at the binary level, with primitive components. It is done both formal argument of the receiver method and received message. 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.

When a suitable operation is found, the server will complete operation and create a response for client. The system also supports optional components, which use the formal argument component if optional fields don't have value or different value. Therefore, the serialization methods in the static serialization class should include the name of the component, whether it is mandatory (with annotation), the type (encoded in the tag) and the value. Each primitive data type can have mandatory annotation. Messages sent use only with mandatory values. Serialized formal arguments can use both mandatory and non-mandatory. The receiver has always a default value for the message, so for the data that don't have mandatory annotation it will always use default value.

Device	Role	Development technology
Mobile 1 **(Windows Mobile)**	Consumer	C#
Mobile 2 **(Android Mobile)**	Consumer	Java

Table 5.4: General properties of the consumers for experiment

	Cloud Server 1	**Cloud Server 2**
Role	Provider	Provider
Development technology	.Net(C#)	Java
Server technology	IIS 7.0 Server	Tomcat 8 Server
Url	http://csharpserverthesis.azurewebsites.net/	http://javatomcatthesis.azurewebsites.net/

Table 5.5: General properties of the providers for experiment

Let's explain that with an experiment. For that experiments there will be demonstration of 2 consumers (one developed with C# and another with Java) and 2 providers (one developed with C# and another with Java). General properties of 2 the consumers can be seen in Table 5.4, then the properties of the providers also can be seen in Table 5.5

Windows Mobile application (in Listing 5.3) sends 2 different query messages (Query1 as seen in Listing 5.7 and Query2 as seen in Listing 5.8) to Cloud1 and Cloud2 providers. Android Mobile application (in Listing 5.4) sends a query message(Query3 as seen in Listing 5.9) to Cloud 1 and Cloud 2 providers. If the receiver conforms to what the sender expects then there is compliance, the message received is partially assigned to that argument and the operation invoked.

```csharp
class Program
    {
        static void Main(string[] args)
        {
            Query1 q = new Query1();
            Message msg = new Message(q);
            string msgToSend = msg.Seriliaze();
            JavaWebServiceClient service = new JavaWebServiceClient();
            // Show result from server
            Console.WriteLine(" Result From Server: \n" +
                service.GetResult(msgToSend));
            Console.ReadLine();
        }

    }
```

Listing 5.3: Windows Mobile application

```java
public static void main(String[] args) throws Exception {
    final ChatClientEndpoint clientEndPoint = new ChatClientEndpoint(new
```

```
        URI("ws://javatomcatthesis.azurewebsites.net/JavaWebServerWebSocket-1.0/javawsendp
3       clientEndPoint.addMessageHandler(new ChatClientEndpoint.MessageHandler()
           {
4               public void handleMessage(String message) {
5                   System.out.println(message);
6               }
7           });
8       Query3 q = new Query3();;
9       Message msg = new Message(q);
10      byte[] msgToSend =  msg.SeriliazeBinary();
11        clientEndPoint.sendMessage(msgToSend);
12        Thread.sleep(30000);
13
14   }
```

Listing 5.4: Android Mobile application

```
1    namespace CSharpWebServer.ist.enesuysal.thesis
2    {
3      public class Receiver
4      {
5          [AvaliableMethod]
6          public void AvaliableMethod(Weather1 weather1){
7          }
8          [AvaliableMethod]
9          public void AvaliableMethod(Weather2 weather2){
10         }
11         [AvaliableMethod]
12         public void AvaliableMethod(Weather3 weather3){
13         }
14     }
15   }
```

Listing 5.5: .Net provider [AvaliableMethod

```
1    package ist.enesuysal.thesis;
2    public class Receiver
3    {
4      @AvaliableMethod
5      public void AvaliableMethod(Weather4 weather4) {
6      }
7      @AvaliableMethod
8      public void AvaliableMethod(Weather5 weather5) {
9      }
```

Cloud Server 1	Cloud Server 2
Weather1 (in Listing 5.10)	Weather4 (in Listing 5.13)
Weather2 (in Listing 5.11)	Weather5 (in Listing 5.14)
Weather3 (in Listing 5.12)	Weather6 (in Listing 5.15)

Table 5.6: Information about input parameter of operations in the provider for experiment.

```
10    @AvaliableMethod
11    public void AvaliableMethod(Weather6 weather6) {
12    }
13  }
```

Listing 5.6: Java provider @AvaliableMethod notation in Receiver Class

```
1   public class WeatherQuery1
2     {
3       public String country = "Portugal";
4       public String city = "Lisbon";
5     }
```

Listing 5.7: Details of Query1 Object

```
1   public class WeatherQuery2
2   {
3     public int cityCode = 35121;
4   }
```

Listing 5.8: Details of Query2 Object

```
1   public class WeatherQuery3
2   {
3     public double latitude = 38.736946;
4     public double longitude  = -9.142685;
5   }
```

Listing 5.9: Details of Query3 Object

Providers have operations with an input parameter of each of its operations. Operations have at most one argument, which can be structured. Matching will be done with message from consumer and input parameter of operations. Input parameters of operations in each server can be seen in Table 5.6

```
1   public class Weather1
2     {
3       [Mandatory]
4       public String country = "Portugal";
5       [Mandatory]
6       public String city = "Braga";
```

```
7      }
```

Listing 5.10: Details of Weather Object

```
1    public class Weather2
2      {
3        [Mandatory]
4        public int cityCode = 35121;
5        public String country = "Portugal";
6        public String city = "Lisbon";
7      }
```

Listing 5.11: Details of Weather2 Object

```
1    public class Weather3
2      {
3        [Mandatory]
4        public String country = "Portugal";
5        [Mandatory]
6        public String city = "Braga";
7        public int cityCode = 351253;
8        public double latitude = 41.530918;
9        public double longitude  = -8.780565;
10     }
```

Listing 5.12: Details of Weather3 Object

```
1    public class Weather4
2    {
3      @Mandatory
4      public String country = "Portugal";
5      @Mandatory
6      public String city = "Lisbon";
7      public double latitude = 38.736946;
8      public double longitude  = -9.142685;
9    }
```

Listing 5.13: Details of Weather4 Object

```
1    public class Weather5
2    {
3      @Mandatory
4      public double latitude = 38.736946;
5      @Mandatory
6      public double longitude  = -9.142685;
```

```
7    }
```

Listing 5.14: Details of Weather5 Object

```
1    public class Weather5
2    {
3        @Mandatory
4        public int cityCode = 35121;
5    }
```

Listing 5.15: Details of Weather6 Object

Lets start with Windows mobile application (Mobile 1); to explain compliance and conformance concepts, by providing a message with `Query1` (in Listing 5.7) and sending it to .Net cloud provider (Cloud Server 1). `Weather1` object in Cloud 1 provider will not be mapped to `Query1`, because `Weather1` in Cloud 1 provider has 2 mandatory fields as seen in Listing 5.10 `country` and `city` fields do not have same value. `Weather2` in Cloud 1 provider has 3 fields one mandatory other two are optional as seen in Listing 5.11 , since mandatory field is not in the `Query1`, mapping will not be occur. `Weather3` object in Cloud 1 provider also will not be mapped to `Query1`, because `Weather3` in Cloud 1 has 2 mandatory fields and 3 optional fields as seen in Listing 5.12, but mandatory `city` field does not have the same value as in `city` field in `Query1`. No suitable operation is found, the server will not complete operation and create a response for consumer in Cloud Server 1. Next step, let's try sending `Query1` to Cloud Server 2 to check if there is any compliable operation. `Weather4` object in Cloud 2 provider will be mapped to `Query1` and compliance hold, because `Weather4` in Cloud 2 has 2 mandatory fields and 2 optional fields as seen in Listing 5.13 and `city` and `country` fields have same value with the `Query1`. The optional fields in `Weather4` does not exist in `Query1`, but that is not important since they are optional and they will be used with their default values. Now suitable operation is found, the server will complete operation and create a response for client in Cloud Server 2. Now, what about when sending `Query2` (in Listing) with Windows mobile application to Cloud 1 and Cloud 2 providers. `Weather1` object in Cloud 1 provider will not be mapped to `Query2` because `Weather1` has 2 mandatory fiels and these fields are not exist in `Query2`. `Weather2` object in Cloud 1 provider will be mapped to `Query2` because, `Weather2` in Cloud 1 provider has 1 mandatory fields and 2 optional fields and `cityCode` field has same value with the `Query2`. The optional fields in `Weather2` does not exist in `Query2`, but that is not important since they are optional and they will be used with their default values. `Query2` message will be complied with `Weather2`, the server will complete operation and create a response for client in Cloud Server 1.

As explained above, sending `Query1` message in Windows phone application will be mapped in Java weather provider and return result message after compliance. `Query2` message in Windows phone application will be mapped in .Net weather provider and return result message after compliance.

The consumer does not need to be written in C#. it can be written in Java or other language. Lets show implementation of Android phone application (Mobile 2). This application has `Query3` as seen in Listing 5.9 and it will send that query message to Cloud 1 and Cloud 2 providers. Because of similar reasons explained before, any operation in Cloud 1 provider will not be mapped to `Query3` because

	Weather1	Weather2	Weather3	Weather4	Weather5	Weather6
Query 1	-	-	-	+	-	-
Query 2	-	+	-	+	-	+
Query 3	-	-	-	-	-	-

Table 5.7: Compliance and Conformance between Query message and Weather objects.

		Cloud Server 1	Cloud Server 2
Mobile 1	Query 1	No	Yes
	Query 2	Yes	Yes
Mobile 2	Query 3	No	No

Table 5.8: Compliance and Conformance between Cloud Servers and Mobile phones.

mandatory fields of `Weather1`, `Weather2` or `Weather3` are not exist in `Query3`. Any operation in Cloud 2 provider also will not be mapped with `Query3`, because again mandatory fields of `Weather4`, `Weather5` or `Weather6` are not exist in Query3.

The results that are explained above about partial assignment of Mobile 1 and Mobile 2 applications can be seen in Table 5.7 and in Table 5.8.

The demonstration in this section, show that provided solution is an alternative solution to XML, JSON, WSDL and REST. The example that presented here, involves compliance or conformance, that it works and that it allows changes to the client and to the server without informing both sides.

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 non-optimal compared to the other ones, but it is easier to implement with existent tools.

Message Transportation in this solution is done with essentially JavaScript and Web Sockets, to circumvent some of the limitations of HTTP. But also the solution supports classical way by encoding and decoding the binary array with BASE64 and then using typical HTTP-based solutions.

Web Sockets are chosen technology for implementation because they 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. Now they are part of the HTML5 world, servers and firewalls are increasingly supporting them and remove this restriction, adds binary support and increases performance.

To examine the performance in HTTP-based solutions and Web Sockets, an experiment was designed using following hardware and software:

- Hardware: IMac(by Apple Inc.) with Intel Core i7 1,7 GHz and 8GB memory.

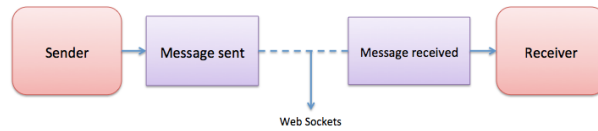- Operation System: Mac OS X version 10.11.4.
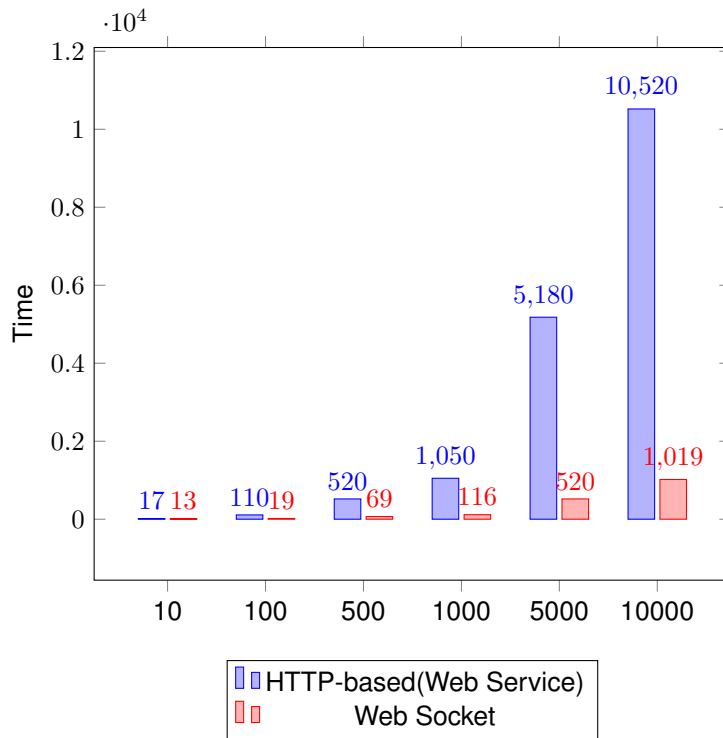
Figure 5.2: Message Transportation.

Figure 5.3: Avarage Times(in ms) of HTTP-based solutions and WebSocket.

- Java: version 1.8.

The experiment was designed as follows:

- Ten kinds of query were prepared for weathercast provider. They were queries with 6 different size of weather: 10, 100, 500, 1000, 5000, 10000 weathercast query.

As seen in Table 5.9 and in Figure 5.3, there is a big difference between the HTTP-based solutions and Web Sockets. It is clear to see using Web sockets technology adds a lot difference in performance aspect. Web Sockets also removes the limitations of HTTP which adds binary support and increases
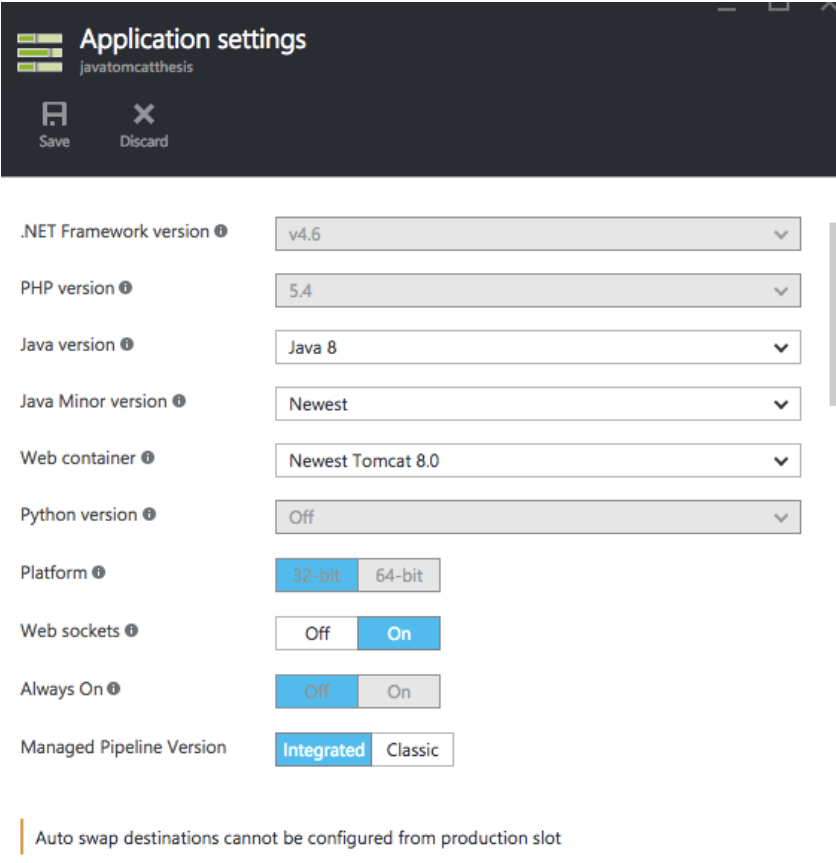
Nb of Message	Web Service(HTTP) (in ms)	WebSocket(in ms)
10	17	13
100	110	19
500	520	69
1000	1050	116
5000	5180	520
10000	10520	1019

Table 5.9: Times (in ms) of HTTP-based solutions and WebSocket.

performance.

5.5   Deployment

The solution is developed and in two different languages that are .NET and Java. The solution is deployed to Microsoft Azure Cloud. The reason choosing Microsoft Azure Cloud instead of other providers is because Microsoft provide free access to their App Servers of Azure with student subscription account. Another reason, .Net and Java technologies can be deployed using the same platform. Microsoft Azure support Java and .Net and that's way 2 different provider will be in the same platform. The Azure application servers support Web Sockets, which is also benefit to test new solution using Web Sockets in cloud environment. .Net client can send a message to Java provider over the cloud by using Web Sockets technology and also Java client can send message to .Net provider. Using Microsoft Azure Cloud technologies allowed us to test the project in cloud environment. The characteristics of Java application server and .Net application server can be seen in Figure 5.4 and in Figure 5.5

Figure 5.4: The characteristics of Java application server.

Figure 5.5: The characteristics of .Net application server.

Chapter 6

Comparison with existing technologies

6.1 Overview

This chapter compares the current technologies and new implemented approach. This comparation is done qualitatively and quantatively. The aim of this chapter is to summarize the advantages and disadvantages of new approach, compared to the current technologies.

6.2 Qualitatively and Quantatively 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.

As described before, 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

51

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. Text has been touted as human readable and therefore advantageous over binary, but this is true only for very simple documents, especially when using XML. By the way textual representation brings parsing expenses and poor support for binary data. When using these solutions you need to produce a client stub, Schema validation and also DOM parsing. All these expenses are a big deal for performance and create complexity in terms of usability. Instead of using textual representation, the binary representation provides native support for binary data, has a smaller length and is faster to parse. Section 5.2 with Figure 5.1 proves that using binary data has more advantage than using XML or JSON.

The implemented approach in new solution 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 [26]. 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

Current solutions of distributed applications use message protocol to communicate each other. SOAP can use almost any transport to send the request, using everything from the before mentioned to SMTP (Simple Mail Transfer Protocol) and even JMS (Java Messaging Service), but REST has restrictions because REST requires use of only HTTP/HTTPS.

The approach in implemented in thesis 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.

HTTP still can be used, but BASE64 must be used to overcome the limitation of HTTP of not supporting binary data. Everything is serialized to the binary, then encode in BASE64 and decode at the receiver to recover the binary data. Advantage of using a new protocols, such as WebSockets, reduce some of the problems. Because WebSockets, now part of the HTML5 world, removes this restriction, adds binary support and increases performance. Using a platform which use Websocket or HTTP/2 will increase usability and performance regarding message transportation.

As seen in section 5.4, there is big difference in terms of performance using WebSockets instead of using HTTP-based solutions. Regarding that the implemented solution in this thesis gives big advantage by supporting new protocols such as WebSockets or HTTP/2.

6.2.3 The interoperability problem

As seen in previous chapters that 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). This is provided with full interoperability and there

is usually no support for partial interoperability and polymorphism in distributed systems with SOA and REST. Asymmetric interoperability allows to show that interaction is still possible with only a partial knowledge of types, as long as the characteristics actually used which are included with supporting partial interoperability. The trick is to allow partial interoperability, by considering only the intersection between what the consumer needs and what the provider offers. It allows for increased interoperability, adaptability and changeability, without the need to have resource types necessarily shared or previously agreed. Next sections (section 6.2.4 and section 6.2.5) will provide information and comparation between full interoperability and partial interoperability.

6.2.4 Full interoperability in SOA and REST

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. These solutions use full interoperability, which the consumer can use all the characteristics that the provider exposes. 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. Additionally you cannot change schema without informing both client and server. So you cannot change either the client or the server without breaking interoperability. For example, in case of SOA style, if producer changes some operation without informing consumer then the producer will have different WSDL schema than consumer currently has, because WSDL schema shows which operations are available and how data should be structured to send to those operations. Since the WSDL schema is different between producer and consumer, consumer will not communicate with producer. Another example, thinking that there are a producer and two consumers that call operations of this producer, but they do not use same operations, because they do different tasks with using the same producer. Still, they need to have the same WSDL schema even they don't use all the characteristics of provider and in case if producer changes some operations for one consumer then both consumers need to be informed since they use the same schema.

Another issue, when you expose a method as a Web service, the data contained in the programming language object must be converted to a language-independent format - namely XML. In the other direction, the XML must be converted into the expected object in the producer (as seen in Figure 6.1). Databinding is all about converting from XML to programming-language specific data structures. Web service tools help by generating code that performs this mapping. The code generation process takes a WSDL as input and generates client code in a particular programming language, referred to as a "stub", which can access the Web service described by that WSDL. Also, it can generate server side code, referred to as a "service skeleton". Databinding frameworks use SAX or DOM to read and write XML documents. Solving data interoperability with databinding, stub and skeloton creation is inefficient with a waste of time, processor power and memory.
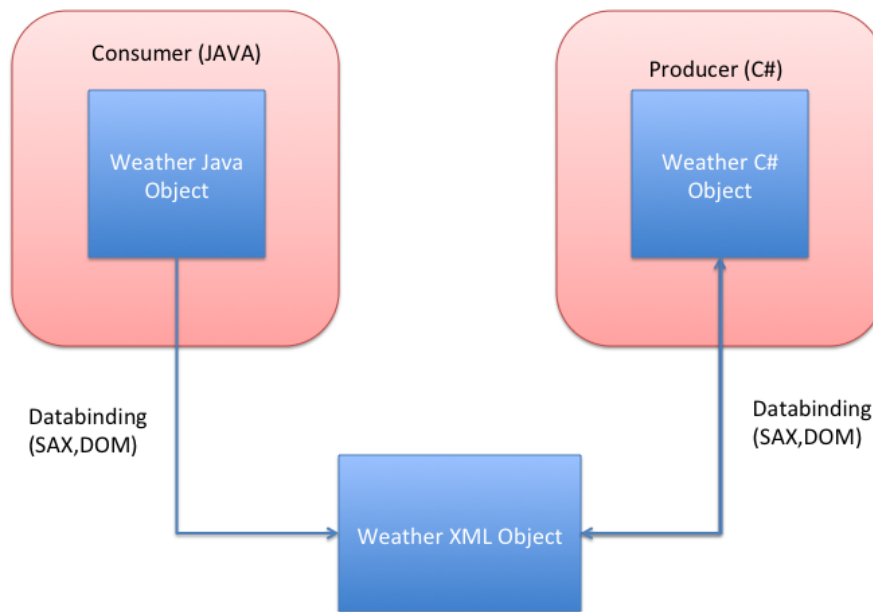
53

Figure 6.1: Data interoperability.

Basically, in these solutions, the receiver needs to be worried with the format and field names of the incoming messages that consumer creates. The target of asymmetric interoperability is removing this coupling problem between consumer and producer.

6.2.5 Asymmetric interoperability

The implemented approach in new solution proposes 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 allows 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. In that solution, consumer will communicate with provider as in current classical solutions, but with a different perspective which avoiding schema sharing. Since there is no shared schema, both consumer and producer can change their structure without informing each other. So consumer and producer do not need to have same structure, Matching will be done field by field between the message and the service operation's formal argument for necessary fields then in case there is a matching, compliance and conformance will be provided. With asymmetric interoperability the receiver deals with its own format and field names, there is no need to generate a stub to deal with the message (Figure 6.2).

The compliance-based assignment of the message to the receiver's formal argument is made at binary level, field by field. The receiver deals only with the message format and field names it already knows, instead of having to deal with the whole structure of the message and its field name. This is how it reduces the coupling as long as compliance holds. This is the main advantage of asymmetric interoperability. The disadvantage of this solution that consumer can try to communicate with producer and if there is nothing mapped between the message and the service operation's formal argument then any operation will be invoked, which means consumer and producer will not exchange the information.
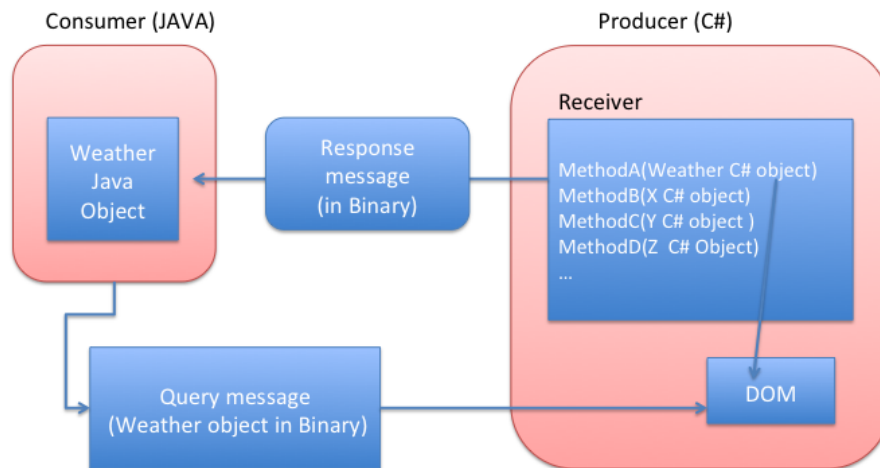

Figure 6.2: Asymmetric interoperability.

As seen in the example of section 4.2, client or the server without breaking interoperability can change their structure of messages. This brings great facility both client and the server by reducing coupling when compared with current traditional solutions. The example (in Section 4.2) also shows that using the asymmetric compliance and structured data, it is equally suitable for both examples of SOA and REST (in Section 3.3), allowing both service interfaces and structured resources.

Chapter 7

Conclusions

7.1   Summary

In this thesis, the proposed solution for a new programming technology is an alternative for current solutions. Current solutions solve interoperability problem with sharing schema, but sharing schema brings the coupling problem between the provider and the consumer because both customer and provider are forced to implement full interoperability.

This solution provides the maximum decoupling possible while ensuring the minimum interoperability requirements with using compliance and conformance instead of sharing the same schema. As long as compliance and conformance is holded, any two resources can interoperate, even if they were designed unbeknownst to each other. This solution allows changing the structure for the client and the server.

Current solutions use XML or JSON as data type for sending or receiving message. They are based on text, heavy, hard to parse and costly in communications. There are technologies for the binary solutions of XML or JSON format but still needed to decompress and compressed. Using XML and JSON forces both side to check and validate their schemas to guarantee that message arrived in correct format. After validation of XML and JSON, it is inefficient in computer terms due to parsing data to build a DOM tree where all characters of a component need to be parsed to reach the next one and it is high memory consumption. This solution uses binary data format natively. It does not require schema and also increases performance. Binary message format is a resulting from compilation of the source program and that uses self-description information when needed. This allows maintaining all the necessary information to communicate in a standard and platform independent way.

Current solutions use a connectionless protocol (HTTP) and it is with the lack of native support for binary data. This solution is implemented to use new protocols such as Web Sockets and HTTP/2 protocol instead using classical HTTP to destroy limitations of HTTP. While it removes this restriction, it adds binary support and increases performance. But still classical HTTP protocol can be used in the solution, but BASE64 must be used to overcome the limitation of HTTP.

7.2   Future Work

Although the provided analyses and methodologies are quite good and show that new solution can be replacament for current ones, there are some improvements that can still be implemented. Current section is related to discuss some possible future developments that could be made to improve implemented work.

Here are these points:

- The solution currently has some limitations such as structures as compound resources without operations (only data fields). So requests and responses must be either primitive resources (e.g., integers) or structures (not complete resources). It would be desirable to add the functionality of operations instead of only data fields.

- The interoperability framework presented in this work needs to be improved and should be completed.

- The compliance and conformance algorithms are implemented in the solution but they need to be optimized in the binary to increase the performance.

- There is a need to create a plan and implement some quality tests for solution to be sure all its funtionalities implemented correctly and possible development bugs are corrected.

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] 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.

[5] 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.

[6] Definition of Interoperability. `http://interoperability-definition.info/en/`, 2016. [Online; accessed 24/04/2016].

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

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

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

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

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

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

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

[14] 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.

[15] Efficient XML Interchange Working Group. `https://www.w3.org/XML/EXI/`, 2016. [Online; accessed 24/04/2016].

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

[17] A. Binstock. After XML, JSON: Then What? `http://www.drdobbs.com/web-development/after-xml-json-then-what/240151851`, 2013. [Online; accessed 24/04/2016].

[18] I. Grigorik. High Performance Browser Networking. `http://chimera.labs.oreilly.com/books/1230000000545/ch12.html`, 2013. [Online; accessed 24/04/2016].

[19] A. Denis. Will WebSocket survive HTTP/2? `http://www.infoq.com/articles/websocket-and-http2-coexist`, 2016. [Online; accessed 30/04/2016].

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

[21] WebSockets – A Quick Introduction and a Sample Application. `https://blog.idrsolutions.com/2013/12/websockets-an-introduction/`, 2013. [Online; accessed 24/04/2016].

[22] 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 24/04/2016].

[23] J. R. D. Brian Benz. *XML Programming Bible*, page pp 88. John Wiley and Sons, 2004.

[24] A. Vohra. *Pro XML Development with Java Technology*, page pp 78. Apress, 2007.

[25] Serialization Performance comparison(XML,Binary,JSON,P...). `http://maxondev.com/serialization-performance-comparison-c-net-formats-frameworks-xmldatacontractserializer-xmlseria` 2016. [Online; accessed 24/04/2016].

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