Distributed Programming in Cloud Computing Platforms

Enes Uysal 72560

Departamento de Engenharia Informatica
Instituto Superior Tecnico

Abstract. In cloud environments, distributed programming solutions that cover interoperability between heterogeneous systems are essentially based on the SOA or REST architectural styles, which in turn are based on HTTP/1.1, XML and JSON. These technologies are not developed for distributed programming in cloud computing environment to develop distributed applications, they were developed in the Web context for the integration of existing systems.

SOA and Web services are heavy and complex technologies. REST is more simple than SOA, but both exhibit a higher degree of coupling than applications require, since both interacting parties share the same data schema and therefore need to cater for all the features of that schema, even if some of them are never used at all. There is also an inefficiency problem, since these technologies are based on HTTP/1.1 and text-based data description languages (XML and JSON). The binary version of XML is no more than a compression mechanism, for transmission purposes only (compressed and decompressed at the endpoints).

This dissertation proposes a new solution to support interoperability of distributed programming systems in cloud computing environments. Contrasting with the schema sharing of SOA and REST, this thesis proposes asymmetric interoperability, in which the sender and receiver may have different schemas, as long as the schema of the sender complies with (satisfies) the part of the schema of the receiver that it actually uses. 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.

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

1 Introduction

Distributed applications are required in most of the central application sectors, including e-commerce, e-banking, e-learning, e-health, telecommunication and transportation. This implies that the Internet plays an important role in business, administration and our everyday activities. Distributed applications that share information do not need to be built with the same technologies. The fundamental problem is the programming of distributed applications with all the basic interoperability problems. They involve distributed platforms and heterogeneous components. Cloud computing platforms create new challenges for distributed systems. They also create distributed platforms, so they need to be able to support distributed applications as easily as possible. That shows the importance of interoperability in cloud environments, because different cloud providers can have different properties, and they need to communicate with each other.

The traditional integration technologies are based on either SOA or REST and they use the document concept as the foundation, with a data description language as the representation format. They use schema sharing as the interoperability mechanism (both sides use the same schema, such as XML Schema) or using previously known data types. The solutions for SOA or REST were also based on text (XML or JSON) and contextual information, imply limitations for many applications.

The main problem of the traditional integration technologies is coupling between the provider and the consumer, because both customer and provider are forced to implement full interoperability (for example, sharing an XML schema). This leads to a greater coupling than needed. Another problem is using XML or JSON is that XML and JSON are inefficient in computer terms due to parsing data to build a DOM tree and checking schemas to validate the structure of the message. The main problems with using traditional integration technologies can be summarized as follows:

- Data interoperability problems based on XML and JSON (Stub generation, DOM parsing)
- Service interoperability problems based on SOA and REST(Coupling with schema sharing)
- The underlying protocol, usually HTTP/1.1, without binary support.

The new solution provides the maximum decoupling possible, while ensuring the minimum interoperability requirements and allowing the client or the sender to change their schema, as long as the actual used parts do not change. The new solution introduces compliance (the consumer must satisfy the requirements established by the provider to accept requests sent to it) and conformance (the provider must fulfill the expectations of the consumer regarding the effects of a request), instead of sharing the same schema. The new solution uses binary directly for message transportation, instead of using XML or JSON. XML and JSON are based on text, which is heavy to parse. Using binary reduces complexity and improves performance. For the performance of message transportation, the new solution uses Web Sockets and HTTP/2 protocol (a binary protocol with small message frames) instead of using the classical HTTP/1.1.

2 State-of-the-art

Web services allow two different machines or two different pieces of code to communicate to each other. Two different applications can communicate 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 it actually is a standard technology, because it is not really specific to Java programming language or any other programming language. You can develop web services using all other programming language technologies. There are primarily two different types of web services. One of these types is called Simple Object Access Protocol (SOAP) web service, and another type called as a REpresentational State Transfer(REST) web service.

Simple Object Access Protocol, or SOAP as it was the first attempt to standardize a web service interface. It is based on sending an XML message to a service in a specific XML format, and receiving an XML response in another specific format. This XML document is actually called WSDL(Web Services Description Language) [4]. As an example, when you send any information across the network from a client to a web service, and the return type back to the client, the data has to be in XML format. You are not really sending a Java string. There is a specification about how you need to send all these different input and output arguments. Basically, any type needs to be send in a specific XML format, which is a protocol that both sender and receiver use, called SOAP(Simple Object Access Protocol). Using this protocol, different technologies can access objects and data. The message can be sent across different transports, including HTTP, FTP (File transfer Protocol), SMTP (Simple Mail Transfer Protocol) and more [6]. The specification does not dictate the transport over which the message should be sent, but most implementations send the XML message over HTTP.

REST (Representational State Transfer) was created in 2000 by Roy Fielding [7]. 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 by using the URL approach. REST can use four different HTTP verbs (GET, POST, PUT, and DELETE) to perform tasks.

Everything in REST web services has a URL, unique and standard. For example, Facebook, when you open an account on Facebook, you will get a profile page that obviously is dynamically generated, so that whenever there is a new profile, it is basically the same page which does the same processing but renders different content depending on the profile that you are watching. In REST web services, you need to think of resources and to create unique URLs for them. For example, you are creating a weathercast application, and you want to get weather information for different cities of Portugal, so your URL needs to be unique for each city as seen in Table 1.

The contents of messages in distributed systems need to be serialized to be sent over the channel with a 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 on the Web (e.g., XML, JSON) are text-based and

Table 1. URL Example.

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

thus verbose and costly in communications. Technologies have been developed to compress text-based documents, such as EXI (Efficient XML Interchange) [8], BSON [9] and others. However, these are compression technologies, which need text parsing after decompression.

In spite of the differences, both suffer from drawbacks and limitations; They are text-based, which means inefficient parsing and data traversal where all characters of a component need to be parsed to reach the next one, high memory consumption, relevant message transmission times and poor support for binary data.

HTTP/2 [10] 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, to minimize protocol overhead via efficient compression of HTTP header fields, and to add support for request prioritization and server push [12].

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 the binary format. This is the foundation that enables all other features and performance optimizations provided by the HTTP/2 protocol.

WebSockets[13] are a relatively new technology that promises to make websites more reactive by allowing lower latency interaction between users and the server. Web Sockets circumvent some of the limitations of HTTP, namely by adding binary support, and increases performance.

Web Sockets entail a protocol that supports 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 communication) and its header is much smaller than the HTTP header, allowing for more efficient communication even with small packets of data.

3 Interoperability

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 is 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.

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 a symmetric arrangement in which a sender produces a message according to some schema, and the receiver uses the same schema to validate and to get the contents of the message. The message is sent over a channel between sender and receiver. In SOA with web services, the schema is usually expressed in XML Schema and WSDL. In the REST world, schemas are known as media types but perform the same role. In any case, the schema or media type must be the same at both sender and receiver. This imposes coupling between the resources for all possible values that satisfy the schema, even if only a few are actually used. In either case, data types need to be fully known by both interacting resources. Resources send requests to each other to invoke a given operation with an 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 of the data types used in SOA and REST are expressed in XML and JSON. 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 some work with this document to understand the message, which is sent by the server or client sides. The solutions are Data binding, DOM (Document Object Model) or SAX(Simple API for XML) [14]. In those technologies, both customer and provider are forced to implement full interoperability (for example, by sharing an XML schema), even if only a fraction of the possible interactions is used. This leads to a greater coupling than needed. The proposed solution reduces coupling by using partial interoperability, creating the maximum decoupling possible while ensuring the minimum interoperability requirements.

4 Architecture of the solution

In the new implemented solution, it will be shown that 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.

In this solution, a different approach will be shown, based on compliance. Messages do not obey some external schema. Each message has one specific value. Messages 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 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 receivers own schema. If it complies, which means it satisfies all the requirements of the receivers schema, the message can be accepted and processed. The advantage of this is that a resource can send a message to all the resources with schemas that the message complies with and, conversely, a resource can receive messages from any resource that sends messages compliant with its receiving schema. In other words, coupling occurs only in the characteristics actually used by messages and not in all the characteristics of the schemas used to generate the message or to describe the service of the receiving resource. Since the schemas of the message and the schemas of the receiver are not agreed upon 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 the 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 be different in the message and in the receiving schema, there is a 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. The sender and receiver no longer need to be designed for each other. 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 a former supports all the characteristics supported by the latter. When a resource can interact with another, although not entirely interoperable with it, this means that there is partial interoperability.

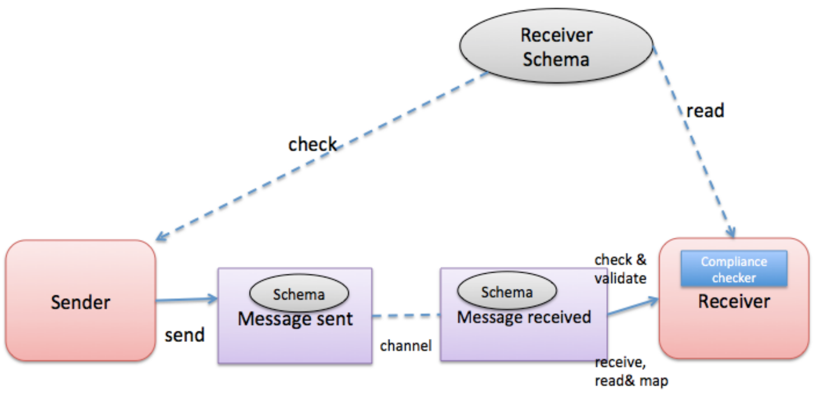

Fig. 1. Asymmetric interoperability.

One of the ideas in this new technology is using the 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[15]. Defining a binary format which messages are serialized on send and recovered on reception. For binary format, we use TLV (Tag, Length and Value) binary markup[16]. An array of bytes is used for 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).

5 Implementation

In the implemented solution, compliance is done at the binary level, with primitive components. It is checked between the received message and the formal argument of the receiver's method. 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 the 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 the client. The system also supports optional components. Therefore, the serialization methods in the static serialization class should include the name of the component, whether it is mandatory or not (with annotation), the type (encoded in the tag) and the value. Each primitive data type can have a mandatory annotation. Messages sent use only the mandatory values. Serialized formal arguments can be either mandatory and non-mandatory. The receiver has always a default value for the message, so for the data that don't have mandatory annotations, it will always use the default values.

Computer programming languages have their own object types and special serialization algorithms for their object types, so when you are working with the same language you can easily serialize an object and deserialize it back. When starting to work with different computer languages and their object types, we cannot use standard Java or .Net(C#) object serialization, because different programming languages use different algorithms for serialization, therefore raising the problem of language-dependent serialization. Creating a common algorithm for different languages was used, instead of a standard serializer for Java or .Net. Then both languages can interoperate by serializing to and deserializing from a common format. The binary format is always the result of serializing data in each language. Each data has a tag, which describes the data type, the number of bytes that follow and the byte-serialized content. Recovering the serialized data is simply done by testing the tag to find the data type and then using the number of bytes and the serialized content.

Transferring the array of bytes from the sender to the receiver requires a binary channel such as Web Sockets or HTTP/2, or a more classical way by encoding and decoding the binary array with BASE64 and then using 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. The message transportation in this new solution is done essentially with JavaScript and Web Sockets, to circumvent some of the limitations of HTTP/1.1. Nevertheless, the solution also supports the classical way by encoding and decoding the binary array with BASE64 and then using typical HTTP-based solutions.

The new solution is developed in two different programming languages, .Net and Java. The solution is deployed to Microsoft Azure Cloud, chosen instead of other providers because Microsoft provides free access to their application servers of Microsoft Azure Cloud with a student subscription account. Another reason, .Net and Java technologies can be deployed using the same platform. Microsoft Azure support Java and .Net, so two different providers will be deployed on the same platform. The Azure application servers support Web Sockets, which is also a benefit when testing the new solution using Web Sockets in a cloud environment. The .Net client can send a message to a Java provider over the cloud by using Web Sockets technology, and similarly a Java client can send a message to a .Net provider. Using Microsoft Azure Cloud technologies allowed us to test the project in the cloud environment.

6 Comparison with existing technologies

Current integration technologies for distributed systems in cloud environments are generally based on textual data description languages (e.g., XML and JSON) and the HTTP protocol. These technologies were especially designed for human-level interaction, but create integration problems at the application level. SOA is usually implemented by Web Services with WSDL. WSDL 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 find SOAP cumbersome and hard to use. For example, working with SOAP in JavaScript means writing tons 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, be standardized or previously agreed upon, when they are application specific. REST doesnt have to use XML to provide the response. You can find REST-based Web services that output data in Comma-separated values(CSV) [17], JavaScript Object Notation (JSON) and Really Simple Syndication (RSS). The point is that you can obtain the output in a form that you need. That is why it is easy to parse with the language you need for your application. While this may seem to add complexity to REST due to the need of handling multiple formats, JSON usually is a better fit for data and parses much faster. REST allows better support for browser clients due to their support for JSON. SOA and REST use textual representation. The text has been touted as human readable and therefore it seems advantageous over binary, but this is true only for very simple documents, especially when using XML. By the way, textual representation brings parsing overheads and poor support for binary data. When using SOA and REST solutions you need to produce a client stub, Schema validation and also DOM parsing. All these overheads 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.

The implemented approach of the 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 (schema 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 upon. 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 unaware of each other. 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. The compliance-based assignment of the message to the receivers 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 names. This is how coupling is reduced, as long as compliance holds. This is the main advantage of asymmetric interoperability. If a consumer tries to communicate with a producer and there is no match between the message and the service operations formal argument then no operation will be invoked, which means that the consumer and producer will not exchange the information.

Current solutions for 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 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. The advantage of using 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 uses WebSocket or HTTP/2 will increase usability and performance regarding message transportation.

7 Conclusions

In this thesis, the proposed solution for a new programming technology is an alternative to current solutions, which solve the interoperability problems by sharing data schemas. This entails a coupling problem between the provider and the consumer, because both are forced to implement full interoperability.

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

Current solutions use XML or JSON as a data type for sending or receiving a 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 they still need data to be compressed and decompressed. Using XML and JSON forces both sides to check and validate their schemas to guarantee that each arrived message is in a correct format.

Typical current solutions use a connectionless protocol (HTTP/1.1), which lacks native support for binary data. The solution has been designed to use new protocols such as Web Sockets and HTTP/2, which support binary data and increase performance. However, the classical HTTP protocol can still be used in the solution, by resorting to BASE64 encoding.

References

1. A. L. Zielinski, Kurt Geihs. New Developments in Distributed Applications and Interoperable Systems, page 327. Springer, 2006.
2. F. A. Natallia Kokash. Formal behavioral modeling and compliance analysis for service-oriented systems. Springer Berlin Heidelberg, 3(9-10):2141, Oct. 2009. doi:10.1007 978-3-642-04167-9 2.
3. 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, Ko- rea, 2007. DOI: 10.1145 1244002.1244305.
4. N. M. Josuttis. SOA in Practice: The Art of Distributed System Design, page 221. OReilly Media, Inc, 2007.
5. M. D. Hansen. SOA Using Java Web Services, page 34. Pearson Education, 2007.
6. L. G. Nikos Antonopoulos. Cloud Computing: Principles, Systems and Applications, page 15. Springer Science, Business Media, 2010.
7. S. P. Jim Webber and I. Robinson. REST in Practice: Hypermedia and Systems Architecture, page 12. OReilly Media, 1st edition, 2010. ISBN-13: 978-0596805821.
8. S. Sakr. Xml compression techniques: A survey and comparison. J Comput Syst Sci, pages 303322, 2008.
9. 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.
10. A. Denis. Will WebSocket survive HTTP/2? http://www.infoq.com/articles/websocket-and-http2-coexist, 2016. [Online; accessed 30/04/2016].
11. J. Delgado. Service Interoperability in the Internet of Things, pages pp 5187. Springer Berlin Heidelberg, 2013.

12. I. Grigorik. High Performance Browser Networking. http://chimera.labs.oreilly.com/books/ 1230000000545/ch12.html, 2013. [Online; accessed 24/04/2016].
13. S. P. Euzenat J. Ontology matching. Berlin. Springer, 2nd edition, 2007. ISBN: 978-3-642-38720-3.
14. J. R. D. Brian Benz. XML Programming Bible, page pp 88. John Wiley and Sons, 2004.
15. Serialization Performance comparison(XML,Binary,JSON,P...). http://maxondev.com/serialization-performance-comparison-c-net-formats-frameworks-xmldatacontractserializer-xmlserializer-binaryformatter-json-newtonsoft-servicestack-text/ 2016. [Online; accessed 24/04/2016].
16. Olivier.Dubuisson. Communication Between Heterogeneous Systems. OSS Nokalva, 2nd edition, 2000. ISBN 978-3-642-38721-0.
17. E. Ron Cody. Learning SAS by Example: A Programmers Guide, page 33. SAS Institute, 1st edition, 2007. ISBN: 159994426X, 9781599944265.