Comprehensive Review and History of Web Services APIs

Pruthviraj M Urankar[1], Vishalakshi Prabhu H[2]

[1]Student, [2]Associate Professor
[1-2]Department of Computer Science and Engineering
[1-2]R V College of Engineering, Bangalore India

---***---

Abstract - *API's or Application Programming Interfaces are software's that enables different applications to communicate over the Internet. Be it to exchange data/information or to provide an interface for a functionality or to gain a particular hardware feature, an API can do it all. An application programming interface, is basically a means for apps to borrow from each other features and data. In this era of internet where every company likes to take its business online and make it available globally on the click of a touch, it becomes necessary to supply mechanisms for different applications with different technologies to interact with each other. Well-developed APIs are useful tools for aspiring developers because they can quickly integrate modern technologies into their new applications (with a limited amount of code), without the need to rewrite everything from scratch. In the earlier days Remote Procedure Calls (RPC) would be mostly used for exchange but it had a downside where it would expose the code and would be prone to attacks but with API's replacing RPC's it has become easier to handle requests by also keeping scalability in mind. This paper performs and in-depth comparative study of these Web Service API's, SOAP, REST, RESTful API's. Such a study will be helpful in identifying the suitable architecture required for an application based on the requirements which are widely used to handle large amounts of data.*

Key Words: **API**, **Webservices**, **REST**, **RESTful**, **SOAP**, **RPC**

1. INTRODUCTION

Web architecture have been predominantly client-server models with them extending to 3-tier based most recently [1]. Such client-server models help us differentiate client and servers in order to examine their architectures and help us identify the interfaces that help both of them interact. Such models give us the ability to write independent code on both client and server sides. Here we shall focus on such interfaces specifically.

1.1 Modern Era of Internet

In this era of internet where every business wants to reach a larger market through web applications, mobile applications and businesses increasingly deploying their services on the internet. It becomes imminent and necessary to exchange data and information between different services. Applications may use different technologies under the hood for example, a web application may be built simple HTML (Hyper Text Markup Language) and CSS (Cascading Style Sheets). There may be many other frameworks that such applications can be built with LAMP (Linux, Apache, MySql, Php), MEAN (MongoDB, ExpressJS, AngularJS, NodeJS) stack technologies being one of the few. With new technologies and framework coming up every other day it becomes increasingly complicated to either standardize or develop RPC's that can be accessed by many applications. Therefore, a standardized technology that has the ability to exchange information without changes to any existing code has been the need of the hour.

1.2 Application Programming Interfaces

An API (Application Programming Interface) provides us such an ability by standardizing how data is exchanged across different platforms. An API provides functionalities that are independent of their respective implementation or definitions and hence allowing us develop programs without the need to compromise on their requirements. How are API's used in the real world? For example, consider you are booking a flight. If you are searching for a flight through a website, you may need to access their database to get available flights, but if you were to access the same through an app from your phone, it becomes difficult for the app and the database to directly interact. But with API's, all the app has to do is to call the API giving it access to the airlines data. So, API is an interface that like your waiter, runs and delivers the data from the database to you and vice versa. Moreover, it facilitates the interaction between the application and the system. According to Programmable Web, there are 15,000 publicly accessible APIs, and several thousands of more proprietary APIs that companies use to extend their internal and external capabilities.

In this article, we discuss client-server architectures throughout a historical sense, explaining how web-based systems and implementations adopt and assimilate innovations and approaches (which are often complementary, divergent or contradictory). The evolution of distributed systems began with message passing functions that were deduced from operating systems need for inter-process communication, therefore we start by looking at how data was exchanged between multiple processes. We then discuss the RPC's that hid the network communication behind interfaces and but had problems with practical implementation due to complicated architectures. We then look at how web services enabled the loosely coupled architectures agglomerated with farinaceous components matching the current requirements of the industry knows as the web services. We also take a look at how HTTP along with JavaScript has enabled us to write independent programs that can communicate over the network [2] without the need for dependency through Web API's. Finally, we will look at different architectures available for the API's currently like REST, SOAP and make a comparative review these web services APIs.

2. Messaging Systems

In applications based on Distributed Systems, the components of the system often need to communicate with other components whenever any event takes place. In the Client/Server messaging system, clients send a request to the server side, and then wait a reply [3]. Messaging systems always involve sending message from one process to another. Messaging systems involve two main higher-level paradigms to achieve this: queuing and publisher/subscriber model. Both these models seek to decouple the recipient from the server with both following a completely different architecture in doing so.

2.1 Queue based Messaging Systems

A queue system is an asynchronous messaging in which a messaging queue is used for communication. By asynchronous we mean, a process need not wait until the message is delivered instead continue with its execution, meaning the need for blocking the sender to wait for a response can be avoided. In a queue system a special messaging queue is established. A message queue is a linked list of messages stored in a kernel. Thus, using a common system messaging queue facilitates the exchange of information between multiple processes because all of them have access to the messaging queue. Figure 1 depicts this scenario. The transmitting process places a message on a queue (through some (OS) message-passing module) that can be read through another process. Each message is given a unique identification to help processes pick the correct messages.

Queues have the following advantages: firstly, they decouple both the producer and consumer using the help of unique named queues. Additionally, queues make use of ordered delivery, which ensures messages are managed in the same order that they were sent. Finally, a queue also provides Quality of Service (QoS), such as timely delivery, non-repeated delivery. Use of message queues however does not scale well for a large number of recipients. Hence, the problem that arises with conventional message queues is that when the number of consumers is large, it makes the entire system effectively slower.

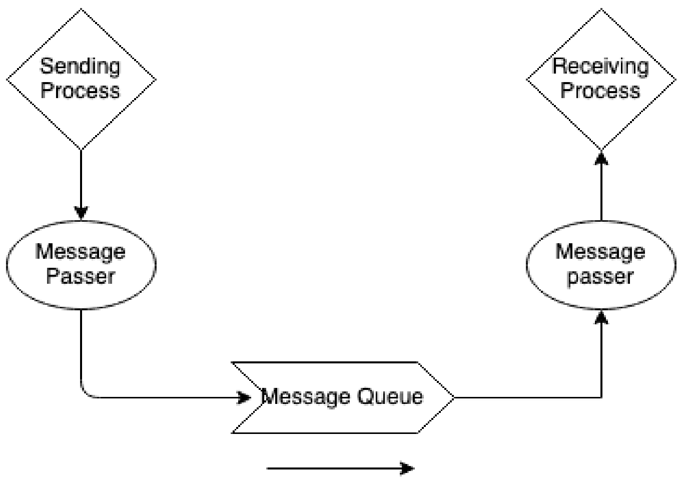

Fig -1: Senders communicating receiver using message queue

Queues have become increasingly popular with cloud applications because they support elastically scalable servers: clients can add messages to a queue, and servers can take messages from the queue and operate upon them. So any server can be filled to full, pulling work when it can support it. If the queue is long, then more servers can be started to do the job.

2.2 Publisher Subscriber Model

The solution to the above scenario is a Publisher subscriber messaging model. Pub-Sub model is one more method of decoupling sender and receivers. The Pub-Sub model differs from a queue model in the following manner a) it defines topic names that are used by receivers to receive all messages of the topic b) enables/allows topic hierarchies, so that the subscription to a root gives aggregates of all its children. Publisher refers to a sender, who sends the message. Subscriber refers to a recipient, receiving that message. So, a message to be sent is published in a topic by the sender [4]. A topic can be any category of messages. All receivers that are subscribed to a topic receive all the messages published in the topic. The publisher therefore does not need to keep a track of the subscribers that need to subscribe to the message. The publisher only needs to send the message to the topic. Subscriber needs to subscribe to the topic [5]. Figure 2 represents a publisher subscriber model using a topic.

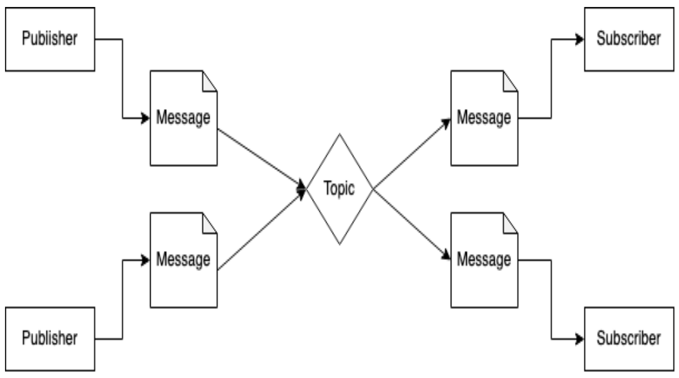

Fig -2: All Sender communicating to receivers through a topic

Publisher/Subscriber Messaging Model offers various benefits such as improving the scalability and reliability. Messaging systems generally focus their attention on transmitting messages, obscuring the functionality and behaviors intended for the client and server. As Nelson commented in [6], from a linguistic design point of view, the message-passing approach can have some drawbacks. Messages implement a basic regulation that is somewhat different from procedure-oriented mechanisms

3. RPC SYSTEMS

A remote procedure call is an interprocess communication technique, also known as the subroutine call or a function call that is mainly used in client-server-based applications. RPC (Remote Procedure Call) implementations encapsulates all the communications that is hides all the complexity behind and only shows what is necessary making it easier to understand the communicating programs. This hiding of complexities is popular, supporting the original intent of Birrell and Nelson whose "primary purpose of our RPC project was to make distributed computing easy"[7]. Remote Procedure Call (RPC) is an effective technique for creating client-server based, distributed applications. The requesting program is considered to be the client and the one that provides the service is called the server.

It is based on expanding the traditional local calling protocol so that there is no need to allow the so-called method within the same address space as the calling protocol. The two processes can be in the same system, or on different systems. This request could be a call to a remote server, or a task. Upon receiving the message, the server sends back the appropriate response to the client RPC is a synchronous operation (Figure 3 depicts this scenario) requiring the client to be suspended until the server responds meaning, the client is interrupted when the call is being handled by the server and operation is only restored when the session is completed. Using a RPC one process can request for a service from another process that may be present in a) different computer b) different virtual address space c) or over the network.

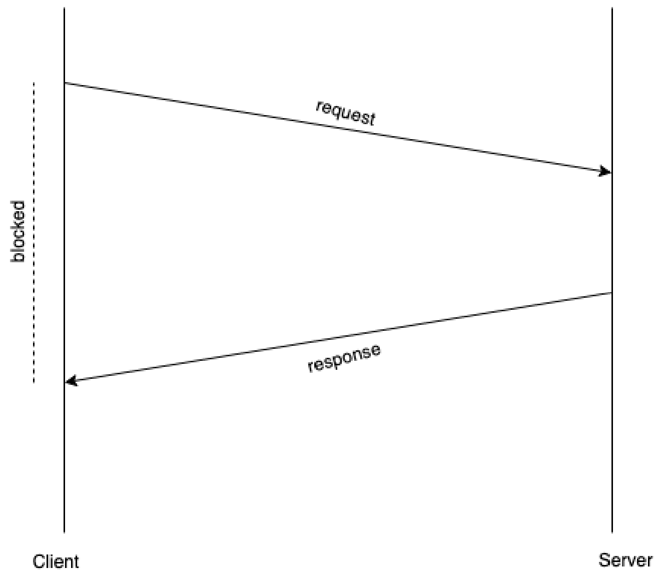

Fig -3: An RPC call representing the blocked state of client.

A few advantages of RPC include the internal mechanism and message passing are hidden from the user, effort required to rewrite the code is minimum in RPC, RPC's have support for both threads oriented and process-oriented models and can either be used in distributed systems as well as in local systems. There are downsides to the RPC's as stated in [6] like

1. Increase in costs due to RPC's.
2. RPC's haven't been standardized different companies may implement in different ways causing them to be incompatible or cause interoperability issues.
3. RPC's systems are tightly coupled. This may lead to issues in another if one of the codebases is changed.
4. Since RPC's hide network communications and are tightly coupled. Developers may ignore or forget the issues that may arise due to network failures.
5. RPC obscures the intrinsic vulnerability in networks that can be detected, exploited or hindered by several parties.

These problems can be avoided, through diligent design and planning, and using RPC extensions (e.g. using asynchronous calls), but RPCs tend to hide these issues rather than reveal them and promote good practice. These problems have not been solved in successors of RPC, such as CORBA either.

4. WEB SERVICE API's

The web was built with a client-server system for exchange of human-oriented hypertext documents [2]. When the web was first launched it was mostly constrained to following links and reading documents [10]. Later when web started to evolve, from providing HTML the ability to both request and send data through HTML forms to today where exchange of data happens without the need to submit forms all asynchronously. Subsequently W3C developed XML and started to gain attention for the usefulness of using it for distributed systems.

Two new innovations were launched in 1999: XML-RPC [8] and SOAP 0.9 (which was eventually standardized as SOAP 1.2 in [9]), leading to a push movement to standardize under the name of Web Services. The movement was led mostly by a large players like IBM, Microsoft, Sun and Oracle. They all wanted to develop an interoperable standard that would be able to connect large systems. A key feature of the Web Services was the description. Each service provided a description through a Web Service Description Language (WSDL). WSDL made mapping of services with their existing objects on server and WSDL's with object on client very simple. But due to the increasing popularity of Web Services, each company decided to create supremacy that would lead to political disputes among sellers. In the end, this has led to the development of many alternate norms. This resulted in low interoperability and increased complexity. This section further describes 3 such web services SOAP, REST and RESTful API's that are widely being used nowadays in the industry.

4.1. SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

SOAP or Simple Object Access Protocol is a web communication interface that has a standard defined, is simpler and hence providing an effective way of web communication. SOAP has more standards like security and how messages are sent. It was designed for Microsoft back in 1998. Today it is mostly used for web services to transfer data from client to server and vice versa over HTTP/HTTPS. Unlike the REST template, SOAP only supports XML data format and strictly adheres to predetermined specifications such as messaging structure, a set of encoding rules, and a convention for providing the requests and responses.

SOAP follows standards, has good security features and has huge extensible capability [6]. SOAP is also language and platform independent due to its built-in functionality that allows creating web-based services. Some of the features and advantages of SOAP include,

1. SOAP works with XML only: SOAP has been designed to work only with XML and can only handle data that is available in XML format. XML is a verbose language that structures data in both human and machine-readable format. This kind of standardization allows developers to be sure of what kind of data they will be processing and hence making it easier to write code without thinking of contradictions.
2. SOAP apparent from the data standard also has another kind of standard. It defines the message structure that must be followed. This message structure makes interoperability easier.
3. Another important feature is that SOAP is extensible with WS standard protocols. Even though SOAP defines the message structure, it doesn't define the content of structure making it customizable as per needs of different users.
4. SOAP is ACID compliant. This reduces anomalies and protects integrity of database by providing how exactly it must interact with the database.

4.2. RESPRESENTATIONAL STATE TRANSFER (REST) AND RESTFUL

REST stands for Representational State Transfer. It is another architectural style/standard like SOAP that defines how two systems can communicate over the network through HTTP/HTTPS. Sharing data between systems has been the fundamental reason why network was developed. REST is data-driven meaning it suggests that all data be converted to an object and then the state of the object be sent as a response. REST does not restrict client-server communication protocol, but it mostly used with HTTP due to its popularity. REST was introduced and defined by Roy Fielding in 2000. It is much simpler compared to other API's like SOAP [11]. REST exploited the existing technology and protocols of web to define and recommend a standard that can be used for communication.

The recommendation made by the REST architecture include

1. The communication must happen between a client and server and hence separating the data from the user-interface.
2. It must be stateless meaning the request sent by client must contain all the information necessary to respond and must not depend on the server for any information.
3. The server must define whether a response is cacheable or not and such cached responses must be reusable for further request if necessary.
4. The requesting client must be aware of who it is communicating with, whether it is the server itself, or a proxy, or an intermediate server.

In a REST architecture, information is referred to as the resource, the response payload provided by the server can be anything that is data/resource [12], be it in the form of HTML, an image file, audio file. Data responses are usually JSON-encoded, but they can use XML, CSV, short strings, or some other format depending on needs.

A few advantages of REST include, REST's design architecture helps exploit the reduced bandwidth usage to make an application more internet-friendly and hence it often referred to as the "language of the internet" and is completely works on resources. Due to this REST has quickly become popular over the internet and has become the most preferred model for building API's

4.2.1. RESTFUL API

A RESTful API is one such API that is based on the REST architectural recommendations. It follows the guidelines suggested in REST. Since RESTful is based on REST, it is more flexible and fast compared to others. A RESTful API works in the following way, it first builds a set of tiny modules by breaking down a transaction. Each element addresses a specific aspect of the operation that underlies it. This modularity allows developers a lot of versatility, but developers can find it difficult to build their REST API from scratch.

An example of using RESTful API is with HTTP. The simple HTTP methodologies specified by the RFC 2616 protocol are used, that is to say GET is used to retrieve a resource, PUT is used to alter the status of an existing resource, POST produces a resource, and DELETE removes the object. The methodologies are used to transfer object and resources from client to the server and get a response. REST along with HTTP has become hugely popular with them being used in cloud applications etc.

5. Comparison of XML-RPC, SOAP and REST/RESTful

Even though RPC hides complexity and supports the Json but has a very narrow scope. Introduced in the early 2000's. RPC uses HTTP as the transport and XML for encoding. XML-RPC is structured to be as basic as possible and simple, but allows for the transfer, encoding and return of complex data structures. An XML-RPC requires that the order be relevant than the parameters. There are downsides to RPC that include, being simpler compared to the other two API's, XML-RPC is less powerful when it comes to capabilities and hence less widely adapted. The architecture of RPC isn't well defined either, this results in less interoperability problems as two different systems may be completely different. RPC's are also tightly coupled hence may cause issues if one of the codebase changes.

Table -1: Comparison of REST and SOAP

Comparison Parameters	**REST**	**SOAP**
What is it?	Representational state transfer	Simple Object Access Protocol
Design	A standardized protocol with pre-defined rules	An architecture with loose recommendations
Message format	Only supports XML	Supports lot of formats like plain text, JSON, XML
Approach	Function driven	Data Driven
Caching	No caching of API calls	API calls can be cached
Statefulness	Stateless/Stateful	Stateless only
Performance	Needs more computing power and bandwidth	Comparatively lesser bandwidth
Advantage	Security, standardized, extensible	Scalable, flexible, friendly
Disadvantage	More complex, less flexible, poor performance	Less secure

SOAP on the other hand is its own protocol and is not an architectural style. It is a bit more complex compared to the other two as it has more standards defined than others like how messages are framed and sent. This may be an added overhead but can be useful to organizations that may need more features like security, transactions and may also need to be ACID compliant. SOAP also has a logic to retry if the message failures to deliver and, hence providing reliability. It is also platform and language independent and works well with distributed enterprise environments. Automation can also be used when necessary in SOAP with certain language products. SOAP is mostly used situations that need high security like bank transactions and highly secure information's like codes etc. are transferred.

REST is more of an architectural style that is stateless and more flexible. It follows a data-driven philosophy that access resource for a data. REST supports various data formats other than just XML like JSON, Plain text which makes it more browser compatible. Even though REST is less secure compared to SOAP, but it can make use of transport level security SSL using HTTPS. REST also consumes less bandwidth in this era where limitations of bandwidth is a well-known problem. REST is also coupled to a lesser degree and hence has fewer errors that may arise due to changes on any side.

6. CONCLUSION

This paper presents a general study and comparison on different web API services used by various software applications. Based on the comparisons done on RPC, SOAP, REST it can be concluded that each software can deliver an effective data exchange interface depending specifically on the use cases. API's have become an integral part of the current and web and are widely being used to exchange data and perform operations and may evolve as different technologies are invented.

REFERENCES

[1] Wayne W Eckerson, "Three tier client/server architectures: Achieving scalability, performance, and efficiency in client/server applications," Open Information Systems, 3(20):46–50, 1995.

[2] Tim J Berners-Lee, "The world-wide web". Computer Networks and ISDN Systems, 25(4):454–459, 1992.

[3] D. Serain, "Client/server: Why? What? How?," International Seminar on Client/Server Computing. Seminar Proceedings (Digest No. 1995/184), La Hulpe, Belgium, 1995, pp. 1/1-111 vol.1.

[4] A. Bhawiyuga, D. P. Kartikasari and E. S. Pramukantoro, "A publish subscribe based middleware for enabling real time web access on constrained device," 2017 9th International Conference on Information Technology and Electrical Engineering (ICITEE), Phuket, 2017, pp. 1-5

[5] Xiwei Feng, Chuanying Jia and Jiaxuan Yang, "Semantic web-based publish-subscribe system," 2008 IEEE International Conference on Service Operations and Logistics, and Informatics, Beijing, 2008, pp. 1093-1096.

[6] Bruce Jay Nelson, "Remote Procedure Call", PhD thesis, Carnegie-Mellon University, May 1981. Published as CMU Technical Report CMU-CS-81-119, XEROX PARC Technical Report CSL-81-9.

[7] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. ACM Transactions on Computer Systems, 2(1):39–59, February 1984.

[8] Dave Winer, "XML-RPC Specification. Technical report", June 1999. Available at http://www.xmlrpc.com/spec

[9] SOAP Version 1.2 Part 1: "Messaging Framework. Recommendation", W3C, June 2003.Available at http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[10] Maria Maleshkova, Carlos Pedrinaci, and John Domingue, "Investigating Web APIs on the World Wide Web", In Proceedings of the 8th IEEE European Conference on Web Services (ECOWS 2010), 2010. Available at http://oro.open.ac.uk/24320/

[11] Fatna Belqasmi, Jagdeep Singh, Suhib Bani Melhem, Roch H. Glitho, "SOAP-Based Web Services vs. RESTful Web Services for Multimedia Conferencing Applications: A Case Study", 2012 IEEE Internet Computing 16(4):54-63 · July 2012.

[12] F. Haupt, F. Leymann, A . Scherer, a n d K. Vukojevic Haupt, "A Framework for the Structural Analysis of REST APIs," in Soft-ware Architecture (ICSA), 2017 IEEE International Conference on, 2017, pp. 55–58.