2016 International Conference on Open Source Software Computing (OSSCOM)

Toward Extending Apache Thrift Open Source to Alleviate SOAP Service Consumption

Behrouz Sefid-Dashti
Department of Computer Engineering,
Kashan University,
Kashan, Iran
b.sefiddashti@grad.kashanu.ac.ir

Seyed Morteza Babamir
Department of Computer Engineering,
Kashan University,
Kashan, Iran
babamir@kashanu.ac.ir

***Abstract*— The improved computational capabilities and low-cost facilities of smart phones and tablets makes these devices candidates for enterprise application platforms. Business alignment of organizational and inter-organizational systems is a matter for service-oriented solutions. The most common implementation of SOA service is SOAP web service, which support accepted standards of composition, security, and reliability, but is verbose in its use of XML tags that require text serialization/deserialization. This poses problems for integration of mobile devices having constraints on their resources and experience connection intermittence. Thrift service is an alternative mode of implementation for SOA services. In contrast to SOAP services, thrift services offer binary serialization/deserialization formats that outperform SOAP. Thrift open source services lack sufficient standards and tools for service composition that are major advantages of SOA. To take the advantage of SOAP and Thrift together, the present study proposes an architecture to automatically produce Thrift services that facilitates consumption of SOAP services such that the overhead of SOAP messages is removed while their availability and orchestration is preserved. Initial implementation demonstrates the proposed architecture.**

Keywords—Thrift; SOAP; mobile service consumption; SOA

I. INTRODUCTION

Business alignment and the loosely coupled interactions achieved by service orientation have fostered an interest in developing new systems using service-oriented architecture (SOA) and migration of legacy systems to service-oriented counterparts. These IT systems include enterprise resource planning (ERP) and customer relationship management (CRM). They provide access by employees at their workstations or away from their desks to information needed to make decisions or accomplish their tasks. Mobile devices such as smartphones and tablets can provide business users access to such enterprise systems so that the services can be used by mobile devices. Mobile devices suffer from scarcity of bandwidth, memory, processing power and energy (battery). A number of challenges exist to improve the mobile user experience and resource utilization [1, 2]. These challenges and requirements are discussed in section III.

Many SOAP services which exist on the internet, W3C standardization, comprehensive technical picture (WS-Security, WS-ReliableMessaging and etc.) and a variety of standards and tools that support composition (WS-BPEL, WS-CDL, BPML, ebXML, OWL-S and WSMF [3]) facilitate development of enterprise systems. However, SOAP platform and language independence and human readability are possible through the detailed description provided in XML/JSON that effects the memory and bandwidth required to store and transmit messages. It requires parsing XML and text serialization/deserialization, which in return effects processing power, memory and battery usage.

In addition to SOAP, service-based systems can be designed using REST. There is conversion of SOAP web services to REST web services [4] and vice versa [5]. REST web services have less computational overhead than SOAP web services [6, 7] and provide better mobile web services [8, 9]. Nonetheless, REST suffers from text serialization/deserialization overhead, as SOAP does. Furthermore, REST and SOA are different architectural styles. SOA services are identified by the organizational goals needs additional tasks to be mapped to REST (data) services while each SOAP service implements a SOA service without mapping.

To alleviate the text serialization/deserialization of web services, binary serialization/deserialization formats such as Thrift [10, 11] and protocol buffers (ProtoBuf) [12] have been evaluated [13, 14] (section IV). Thrift and ProtoBuf outperform XML and JSON, but Thrift supports more programming languages than ProtoBuf. Thrift services are another implementation of SOA services and were developed by Facebook for mobile and other applications [11]. Thrift is now supported by Apache as an open source project. Instead of text-based computations, Thrift provides language independence through a code generation tool and a software stack. Binary representation by Thrift requires less memory, bandwidth and processing than text-based representations (XML and JSON); hence, it outperforms text-based representations in mobile service consumption [13, 14].

The present study proposes an architecture which offers the advantages of Thrift in SOAP service consumption. Mobile devices can consume accessible and compostable SOAP services through generation of Thrift services that act as a façade. Initial implementation (section VI) shows that the proposed work is promising.

Section II provides an overview of Thrift. Section III discusses the requirements of mobile service interaction. Related researches on enhancing mobile service consumption are discussed in section IV. The proposed architecture is described in section V and section VI demonstrates the initial implementation. The paper is concluded in section VII.

II. THRIFT

Apache Thrift is a software framework open source to develop cross-language and cross-platform low-overhead services. Facebook developed Thrift for internal use and released it to open source in 2007 [11]. Thrift is now backed by Apache [15] and supports over 15 languages, although the level of support differs for each language [10].

Thrift consists of a software stack, an interface definition language (IDL) and a code generation engine. The first provides flexibility for developers to employ various transport media, protocols (serialization/deserialization) and server types. The others provide cross-language support.

The developer defines data, services and the name of target languages in a file with a ".thrift" extension written in IDL. The Thrift compiler uses the .thrift file to generate the corresponding service and client in the designated languages. The developer implements the server business logic and determines the appropriate configuration for the software stack for both server and client.

The Thrift software stack is depicted in Fig. 1. The transport layer provides support for variations of TCP sockets, raw data in the memory, and files on the disk. There are also nested transports such as buffering options that can wrap other transports to provide input and output buffering. This layer makes the application code independent of transportation reads and writes [10, 11]. Moreover, connection can be secured by wrapping it in TLS/SSL encryption [10].

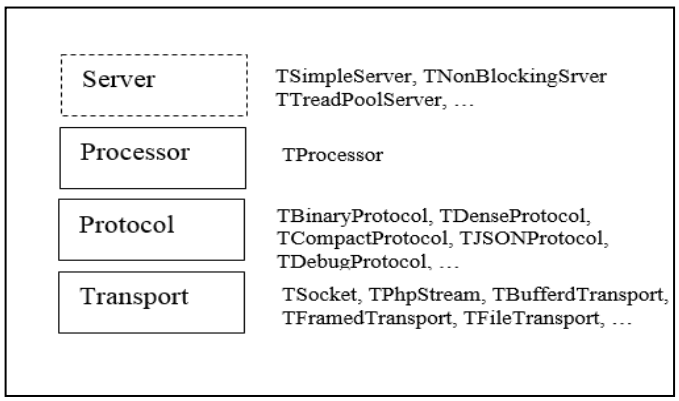

Fig. 1. Thrift software stack

Serialization/deserialization formats such as binary and JSON are the concern of the protocol layer. This layer determines the structure of messages that will be transported. The processor layer is generated in accordance with the .thrift file. It handles sends and receives [10], such as remote procedure calls for network transport [11].

The server layer defines the server that uses elements of the aforementioned layers to provide communication and options for employing blocking/non-blocking and threading servers. In Fig. 1, the server is denoted as a dashed line to emphasize that other layers exist on both the server and client while the server is configured and used only in server code. This framework allows the application developer to concentrate on the application logic and hides differences between various transports, protocols, servers and mapping between datatypes of different languages which can be either statically or dynamically typed.

III. REQUIREMENTS OF MOBILE SERVICE INTERACTION

Integration of mobile devices into service-oriented enterprise systems requires appropriate use of available device resources (e.g., memory, energy, network interfaces) and resources in the environment (e.g., networks and augmented posters) to cope with the constraints of mobile device integration (e.g., constrained resources and connection intermittence). Considerations that affect mobile device functioning as a service consumer and/or service provider have been discussed by Sefid-Dashti and Habibi [1] in terms of architectural drivers and architectural patterns. The present study is a continuance of the research of Sefid-Dashti and Habibi [1, 16]. The following is brief list of constraints affecting mobile device service consumption and/or provision.

- The processing power of mobile devices is lower than that of stationary devices and can vary according to mobile device brand.
- The memory size of mobile devices is lower than that of stationary devices and can vary according to brand.
- The bandwidth is limited by device network interfaces and environment-available networks.
- Connection intermittence can be caused by factors such as roaming.
- Battery capacity affects available energy; both processing and communicating, required by service invocation, consume energy.
- Smaller keypads of mobile devices in comparison with those of stationary computers can affect input methods.
- Smaller screens of mobile devices than those of stationary computers can affect effectiveness of presentation methods.

The present study targets the first five constraints and employs Thrift low-overhead communications to decrease processing power, memory, bandwidth and energy (in turn) for mobile service consumption. The last two constraints are beyond the scope of this study and are more or less addressed in works such as Sefid-Dashti and Habibi [16]. The proposed

architecture (section V) can be used to generate Thrift services equivalent to SOAP services that offer business-aligned service oriented solutions. Mobile devices can use Thrift services at a lower processing cost and with less data communication than SOAP. In turn, the SOAP services can be orchestrated according to changes of business needs and the equivalent Thrift services can be generated so that mobile users can use the new composite service with fewer resources than that required for the original composite SOAP service.

IV. Related Works

A. Comparison of SOAP, Thrift and Similar Protocols

Serialization and deserialization formats defer the processing power, memory, bandwidth and energy that are scarce resources for mobile devices and affect user operator cost, perceived delay, battery usage and ease of development. A number of comparisons can be made between serialization and deserialization formats (protocols). Some have been done for mobile devices and some have been evaluated independently from mobile platforms.

Sumaray and Makki [13] compared XML, JSON, ProtoBuf, and Thrift (binary) formats on an Android platform. They used one text-intensive object and one number-intensive object for experimentation and concluded that the smaller size and lower serialization/deserialization time of binary formats (ProtoBuf and Thrift) versus the ease of debugging of human readable text-based formats (XML and JSON) point to the use of JSON for existing web services and either Thrift or ProtoBuf for new services. Their evaluation found that Thrift is about 10 times faster than XML for both serialization and deserialization. Thrift has a slightly smaller data size for the text-intensive serialized object than XML while, the output data size is less than half the output of XML for the number-intensive serialized object.

Gligorić et al. [17] demonstrated that ProtoBuf (ProtoBuf binary protocol is more or less comparable to Thrift) is more efficient than XML in terms of time, memory, and battery usage. Maeda [14] compared several serialization libraries and demonstrated that Thrift and some binary formats (ProtoBuf, Avro) offer smaller output size and lower serialization/deserialization time than XML and JSON.

Rakowski [10] broadly discussed the effects of several tools of a distributed system and emphasized the availability of SOAP web services over the internet, ease of debugging through standardization, overhead of XML processing and of encoding binary data into text format. He proposed employing SOAP web services when a services has more external consumers and needs standard architecture with proper documentation. He proposed Thrift for transferring binary data in a high performance manner and applying service definition changes in different platforms and languages.

B. Alleviating SOAP Service Interaction Overheads

Since the introduction of SOAP services, research communities have developed different approaches to managing the computing and communicating overhead of SOAP messages so that more devices can afford the processing and communication requirements needed to consume/provide services. Research indicates three approaches to improving interaction with SOAP services:

1) Mediation

This approach delegates some or all of the tasks of service invocation to a mediator. Mediation pre-processes service requests [18], filters invocation results [18, 19], processes WS-Security on behalf of a resource-constrained service provider [20] or processes tasks of a partitioned SOAP engine [21]. Such mediation depends on the existence of a secure network channel that connects the resource-constrained service consumer/provider to the mediator, but this assumption can be violated for resource-constrained nodes such as sensors and mobile devices from time to time.

2) Conversion

This approach converts SOAP service invocations into another communication scheme that requires less processing and/or communicating resources. Converting SOAP services to equivalent RESTful services [4], streaming SOAP communication [22], lossless-compression [23], business process based lossy-compression [24], structural metadata transmission-based compression [25], use of agent protocols such as ACL and FIPA-SL which have compact messages [26] and uses of JSON service invocation results to reduce client side processing and data exchange [19] are among the existing conversions.

3) Sub-setting

This approach supports a subset of SOAP specifications, such as simple types to reduce required mobile device communicating and/or processing. Use of a subset of SOAP specifications for resource-constrained service providers [27] and service consumers [24] are examples of this category.

The proposed approach fits the second category (conversion). Thrift was designed to provide high performance cross-language communication [10, 11]. To the best of our knowledge, no SOAP-to-Thrift conversion architecture currently exists. This study proposes a novel Thrift-based architecture that provides access by constrained-resource devices to business-aligned SOAP-based SOA enterprise systems.

V. Proposed Architecture

This section describes the proposed architecture to automatically wrap SOAP web services into Thrift services so that mobile devices can use the service with less computation and communication overhead. The proposed architecture is shown in Fig. 2 which uses UML notation. The Thrift code generation engine is denoted as the Thrift compiler. The other generators (the SOAP2Thrift service definition generator and the façade and configuration generator) are specific to the proposed architecture. They generate Thrift IDL files and the three codes required to automatically façade SOAP service through a generated Thrift service. These components are described below, but the associated flow is presented first. The flow of SOAP2Thrift service conversion is as follows:

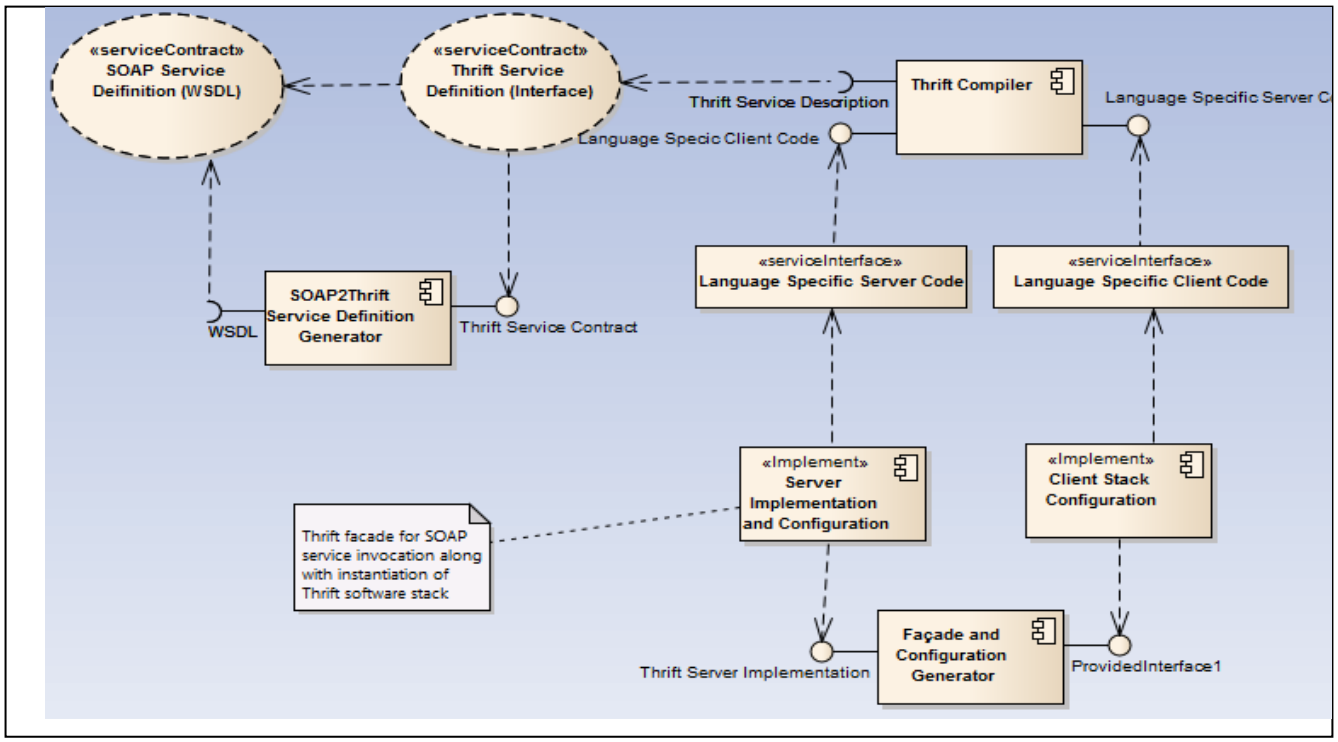

Fig. 2. Proposed Architecture

1. Based on the WSDL file containing the service description of a SOAP web service, the Thrift IDL file for service description is generated. The IDL files are programming language agnostic.
2. The generated IDL file is compiled by the Thrift compiler to generate the language-specific communication code of the client and the server. These codes are generated in languages as determined by the IDL file.
3. The Thrift server file that invokes SOAP service as a façade is generated. This file implements an interface generated by the Thrift complier such that the class that implements the interface wraps the SOAP service invocation.
4. Instantiation of the Thrift software stack is generated for server initiation.
5. Compatible instantiation is generated for client initiation.
6. The generated server and client codes are compiled by the desired language compiler.
7. The complied server and client are deployed.
8. The server is run.
9. The client is run and uses the Thrift service as equivalent to the (composite) SOAP service.

Except for steps 2 and 6 (the functionality of the Thrift compiler and the language specific compiler, respectively), the other steps are specific to the proposed architecture. Each component of the architecture is described below.

A. SOAP2Thrift Service Definition Generator

SOAP web services use WSDL files to describe service interfaces that are platform and programming language independent. Composite services can be defined by tools such as BPEL, a language for orchestration and composition of services. BPEL exposes a WSDL interface so that both the atomic and composite services are described by WSDL files. The Thrift IDL files provide independence in a manner similar to WSDL. The SOAP2Thrift Service Definition Generator generates a Thrift IDL file according to operations defined in the WSDL file of SOAP service.

B. Thrift Compiler

The Thrift compiler aligned with the Thrift software stack provides code generation engine capability in the supported languages. The code generation engine creates a language-specific client and server codes according to language-agnostic IDL files. This generated code is a validated code that removes the need for advanced runtime type-checking [11].

C. Façade and Configuration Generator

This component creates language-specific server and client files. The server file imports the necessary modules, defines a façade that invokes the corresponding SOAP service, and creates a configuration for the Thrift software stack. The configuration defines endpoint information and uses TSocket transport and TBinaryProtocol protocol to communicate between network endpoints with low transmission and processing. It can also envelope TSocket transport in a TBufferedTransport to equip it with send and receive buffering.

The client file defines the list of necessary modules to be imported and configures the Thrift software stack for the client language so that the client configuration matches the defined server configuration. Both the client and server files are compiled, deployed[1] and run.

VI. Initial Implementation

To demonstrate the feasibility of the proposed architecture, it was initially implemented in C# language, which supports basic types. The generators support solely C#, but implementation is extendable for as many languages as Thrift supports.

For brevity, a SOAP web service that provides simple arithmetic operations was used as input for implementation of the SOAP2Thrift Service Definition Generator. This generator iterates through the web methods of the SOAP web service to generate the corresponding Thrift IDL file. The generated IDL file is shown in Fig. 3. It introduces C# as the target language for the client and server and defines a service that includes three arithmetic methods.

The Thrift compiler is used to generate the client and server corresponding to the IDL file. It defines a Thrift service inherited from the base Thrift classes. In this step, the business logic of the Thrift service should be added and the configuration of the Thrift software stack should be determined.

The implementation of the Façade and Configuration Generator generates three snippets of code as depicted in Figs. 4, 5, and 6. Fig. 4 shows invocation of the SOAP web service as business logic of the Thrift service in which each Thrift service generated acts as a façade for the corresponding SOAP service. Fig. 5 depicts Thrift software stack configuration for the server. The predefined configuration (section V) is employed that uses binary serialization/deserialization over socket transport, which is not equipped with send and receive buffering. Fig. 6 depicts Thrift software stack configuration for the client. It should match the server configuration; hence, it uses binary protocol and socket transfer as did the server.

The illustrated implementation demonstrates the feasibility of the proposed architecture for the exemplar mathematics service. All required codes were generated and no human manipulation was included.

VII. Conclusion

The present study illustrates the advantages of two known implementations of SOA services: SOAP and Thrift. While SOAP benefits from standardization and wide tool-support, it suffers from verbose XML processing and text serialization/deserialization overhead. Thrift employs binary serialization/deserialization to achieve low-overhead communications that is more suitable for resource-constrained mobile devices (section III). Combining the benefits of the SOAP and Thrift services provides the business alignment and

[1] Deploying client components of the proposed architecture in Microsoft Windows Phone platform requires employing ThriftClientPortable NuGet that is not discussed in this paper.

```
namespace csharp ArithmeticService

service ArithmeticService{
i32 Add(i32,i32),
i32 Subtract(i32,i32),
i32 Multiply(i32,i32)

}
```

Fig. 3. Generated IDL file

```
namespace ArithmeticService
  {
    class ArithmeticService:
              ArithmeticService.ArithmeticService.Iface
    {
        public ArithmeticService() {}
        ArithmeticServiceSoapClient service =
            new ArithmeticService.ArithmeticServiceSoapClient();

                public int Add(int x, int y)
                {

                     return service.Add(x, y)

                 }
                 public int Subtract(int x, int y)
                {

                     return service.Subtract(x, y)

                 }
                 public int Multiply(int x, int y)
                {

                     return service.Multiply(x, y)

                 }
    }
  }
```

Fig. 4. Generated server logic to façade invocations

```
try
{
    ArithmeticService handler =
                new ArithmeticService();
    ArithmeticService.Processor processor =
       new ArithmeticService.Processor(handler);
    TServerTransport serverTransport =
                    new TServerSocket(9090);

    TServer server = new
            TSimpleServer(processor, serverTransport);
    Console.WriteLine("Starting the server...");
    //Console.WriteLine(handler.add(2, 3).ToString());

    server.Serve();
}
catch (Exception x)
{
    Console.WriteLine(x.StackTrace);
}
Console.WriteLine("done.");
```

Fig. 5. Generated Thrift stack configuration for the server

```
TTransport transport =
    new TSocket("localhost", 9090);
transport.Open();
TProtocol protocol =
   new  TBinaryProtocol(transport);
ArithmeticService.Client client =
   new ArithmeticService.Client(protocol);
```

wide accessibility to the many existing SOAP services and benefits from low-overhead Thrift communication. The proposed architecture generates a Thrift IDL file and required snippets of codes to provide Thrift services equivalent to each

Fig. 6. Generated Thrift stack configuration for the client

SOAP service. Each Thrift service acts as a façade for the equivalent SOAP service such that mobile consumers interact with Thrift service using much fewer resources than the equivalent SOAP service.

Initial implementation was provided for an exemplar mathematics SOAP service. Although it was limited in terms of supported typed and language, it showed the feasibility of the architecture and that all required codes were generated and no human manipulation was included. This fully-automated conversion can accelerate provision of mobile services as fast as the composite services are defined and can foster the adoption of the proposed architecture for mobile users.

REFERENCES

[1] B. Sefid-Dashti, and J. Habibi. "A Reference Architecture for Mobile SOA", *Systems Engineering, vol.* 17, no. 4, 2014, pp. 407-425.

[2] B. Gao, L. He, and C. Chen. "Modelling the Bandwidth Allocation Problem in Mobile Service-Oriented Networks", In *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2015, pp. 307-311.

[3] Q.Z. Sheng, , X. Qiao, A.V. Vasilakos, C. Szabo, S. Bourne, and X. Xu. "Web services composition: A decade's overview", Information Sciences, vol.280, 2014, pp. 218-238.

[4] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau. "Migration of SOAP-based services to RESTful services." In *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*, 2011, pp. 105-114.

[5] Y.Y. Peng, S.P. Ma, and J. Lee. "REST2SOAP: A framework to integrate SOAP services and RESTful services", In *2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, 2009, pp. 1-4.

[6] F. Belqasmi, J. Singh, S.Y.B Melhem, and R.H. Glitho. "SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing", IEEE Internet Computing, vol. 16 , issue: 4 , July- Aug. 2012 , pp. 54-63.

[7] G. Mulligan, and D. Gracanin. "A comparison of SOAP and REST implementations of a service based interaction independence middleware framework." In Proceedings of the 2009 Winter Simulation Conference (WSC), 2009, pp. 1423-1432.

[8] K. Wagh, and R. Thool. "A comparative study of soap vs rest web services provisioning techniques for mobile host", Journal of Information Engineering and Applications, vol. 2, no. 5, 2012, pp. 12-16.

[9] F. AlShahwan, and K. Moessner. "Providing soap web services and restful web services from mobile hosts", In 2010 Fifth International Conference on Internet and Web Applications and Services (ICIW), 2010, pp. 174-179.

[10] K. Rakowski, Learning Apache Thrift. Packt Publishing Ltd; 2015 Dec 30

[11] M. Slee, A. Agarwal, and M. Kwiatkowski. "Thrift: Scalable cross-language services implementation", *Facebook White Paper* 5, no. 8, 2007.

[12] Google Protocol Buffers: Google's Data Interchange Format, "Documentation and open source release", http://code.google.com/p/protobuf/

[13] A. Sumaray, and S. K. Makki. "A comparison of data serialization formats for optimal efficiency on a mobile platform", In Proceedings of the 6th international conference on ubiquitous information management and communication, no. 48, 2012.

[14] K. Maeda. "Performance evaluation of object serialization libraries in XML, JSON and binary formats", In 2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP), 2012, pp. 177-182.

[15] Apache Foundation, "Apache Thrift", https://thrift.apache.org, last accessed July. 19, 2016.

[16] B. Sefid-Dashti, and J. Habibi. "A conceptual usability framework for mobile service consumers", International Journal of Computer Technology & Application (IJCTA), vol. 2, no. 4, 2011, pp. 894-903.

[17] N. Gligorić, I. Dejanović, and S. Krčo. "Performance evaluation of compact binary XML representation for constrained devices." In 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011, pp. 1-5.

[18] D. Schreiber, A. Göb, E. Aitenbichler, and M. Mühlhäuser. "Reducing user perceived latency with a proactive prefetching middleware for mobile SOA access", International Journal of Web Services Research (IJWSR), vol. 8, no. 1, 2011, pp. 68-85.

[19] L. Hamdi, H. Wu, S. Dagtas, and A. Benharref. "Ajax for mobility: mobileweaver ajax framework", In *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 1077-1078.

[20] S.N. Srirama, M. Jarke, and W. Prinz. "Mobile web services mediation framework", In Proceedings of the 2nd workshop on Middleware for service oriented computing: held at the ACM/IFIP/USENIX International Middleware Conference, 2007, pp. 6-11.

[21] M. Asif, S. Majumdar, and R. Dragnea. "Partitioning the WS execution environment for hosting mobile web services", In Services Computing, 2008. SCC'08. IEEE International Conference on, vol. 2, 2008, pp. 315-322.

[22] S. Oh, and G.C. Fox. "Optimizing Web Service messaging performance in mobile computing." Future Generation Computer Systems, vol. 23, no. 4, 2007, pp. 623-632.

[23] A. Papageorgiou, J. Blendin, A. Miede, J. Eckert, and R. Steinmetz. "Study and comparison of adaptation mechanisms for performance enhancements of mobile web service consumption", In 2010 6th World Congress on Services, 2010, pp. 667-670.

[24] Y. Natchetoi, V. Kaufman, and A. Shapiro. "Service-oriented architecture for mobile applications", In Proceedings of the 1st international workshop on Software architectures and mobility, 2008, pp. 27-32.

[25] H. Wu, and Y. Natchetoi. "Mobile shopping assistant: integration of mobile applications and web services", In Proceedings of the 16th international conference on World Wide Web, 2007, pp. 1259-1260.

[26] A. Shajjar, N. Khalid, H.F. Ahmad, and H. Suguri. "Service interoperability between agents and semantic web services for nomadic environment", In Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. 02, 2008, pp. 583-586.

P. Pawar, S. Srirama, B.J.V. Beijnum, and A.V. Halteren. "A comparative study of nomadic mobile service provisioning approaches", In The 2007 International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST 2007), 2007, pp. 277-286.