University of Tartu

Institute of Computer Science

Information Technology Curriculum

Rait Ool

SOA without SOAP

Overview of service-oriented architecture technologies, putting emphasis on SOAP

Bachelor's Thesis (6 ECTS)

Supervisor: Maria Gaiduk

Co-supervisor: Kristo Kuusküll

Tartu 2016

SOA ilma SOAP'ita

Ülevaade teenustele orienteeritud arhitektuuri tehnoloogiatest, asetades rõhu SOAP'ile.

Lühikokkuvõte: Käesoleva bakalaureusetöö põhiline eesmärk on süsteemselt üles loetleda teenustele orienteeritud arhitektuuri (SOA) vanade ja uute implementatsioonide tugevused ja nõrkused, asetades rõhu SOAP'ile. Selle täitmiseks on võrdlusesse valitud neli tehnoloogiat, mis mängisid olulist rolli SOA tehnoloogiate kasutuselevõtus.

Võtmesõnad: SOA, teenustele orienteeritud arhitektuur, tehnoloogiate võrdlus, CORBA, SOAP, OSGi, Feign

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

SOA without SOAP

Overview of service-oriented architecture technologies, with emphasis on SOAP.

Abstract: The main goal of this thesis is to systematically identify the advantages and weaknesses in older and newer implementations of service oriented architecture (SOA), putting emphasis on the ones related to SOAP. In order to fulfill this goal, four technologies were chosen, which played a significant role in adopting SOA.

Keywords: SOA, service-oriented architecture, technologies comparison, CORBA, SOAP, OSGi, Feign

CERCS: P170 - Computer science, numerical analysis, systems, control

Table of contents

1. INTRODUCTION

Service Oriented Architecture (SOA) has been circling around the programming world for decades in the form of Common Object Request Broker Architecture (CORBA). Some experts say that SOA really took off with Simple Object Access Protocol (SOAP), which indeed fixed a lot of issues compared to earlier implementations.[1] Others think it still was not enough - for example Tim Bray, Director of Web technology at Sun Microsystems Inc., stated: "No matter how hard I try, I still think the WS-* stack is bloated, opaque, and insanely complex. I think it's going to be hard to understand, hard to implement, hard to interoperate and hard to secure."[2] Only in the past few years, with the rise of microservices, has SOA become the flagship for developing web applications.

There are two ways of designing applications:

- Monolithic - writing a single application which handles all necessary requirements without (or with as few as possible) dependencies on other systems
- Modular - dividing a system into smaller parts

Both of them have their own advantages, but in terms of solving performance issues, modular versions are usually easier. SOA in its essence is a modular design and it inherits those advantages,

Many experts agree that SOAP was a major turning point in adapting SOA. In light of that the main goal of this thesis is to systematically identify the advantages and weaknesses in SOA's older and newer implementations, putting emphasis to comparison SOAP. Because many of the technologies are quite similar to each other, the list of which ones to focus on, was scaled down, based on the collective work experience of me and my supervisors. Each technology resides in separate chapter consisting of a short history, followed by its' key features and ending with an example.

[1] Kristopher Sandoval "Microservices Showdown - REST vs SOAP vs Apache Thrift (And Why It Matters) http://nordicapis.com/microservice-showdown-rest-vs-soap-vs-apache-thrift-and-why-it-matters/
[2] Tim Bray, "Loyal opposition to Web Services" (2004)

2. SERVICE ORIENTED ARCHITECTURE (SOA)

This paragraph gives a short introduction to SOA and lists key concepts when describing any SOA implementation,

The Organization for the Advancement of Structured Information Standards (OASIS) group defines SOA as a programming paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.[3]

The central concept of service oriented architecture is the service. The noun "service" is defined in dictionaries as "The performance of work (a function) by one for another[4]". It is natural to think of one's person's needs being met by capabilities offered by someone else; or, in the world of distributed computing, one computer agent's requirements being met by a computer agent belonging to a different owner. The perceived value of SOA is that it provides a framework to match these needs and capabilities.[3]

The following key concepts are used to describe SOA paradigm:[3]

1. Visibility – capability for these with needs and those with capabilities to see each other.
2. Interaction – activity of using a capability
3. Effect – the purpose of using a capability, the expected outcome of the performance of work
4. Service descriptions – contains necessary information to use a capability, combines concepts 1 to 3
5. Service providers – entities (people and organizations) that offer capabilities
6. Service consumers – entities (people and organizations) with needs who make use of services
7. Service participants – combines concepts 5 and 6

[3] OASIS, "Reference Model for Service Oriented Architecture 1.0" https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.doc
[4] Merriam-Webster, "Service | Definition of Service", http://www.merriam-webster.com/dictionary/service

Using a capability is most frequently accomplished by sending and receiving messages. Although there are other means of invoking a service (for example, modifying the state of a shared resource), exchanging messages is often referred as the primary mode of interaction. [3]

A service is self-contained logical representation of a business activity that has a specified outcome. For consumers of the service, it is a “black box” - no internal logic is visible (including whether or not the service is composed of other services).

This design is beneficial for multiple reasons:[5]

- Service can be internally changed without the need to update clients
- Clear input-output allows automated tests
- Load balancing and failover services can be configured
- Forces developer to write loosely-coupled, high cohesion services

[5] Service Oriented Architecture (SOA), https://msdn.microsoft.com/en-us/library/bb833022.aspx

3. COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

This paragraph provides a short introduction to CORBA. The first section gives a high-level overview of CORBA itself.

The first release of CORBA came of in October of 1991. Having only a bare minimum of components resulted in a bad start. Although there were some minor releases in the following years, it was not until 1997, that a new major version came out with a standardized protocol and a C++ language mapping. The next minor release (in 1998) provided a Java language mapping, as well. This gave the developers a powerful framework to build their heterogeneous distributed applications. [6]

One of the main features of CORBA was to allow mutual communication between application without the restrictions of platforms or languages. All this flexibility was achieved through the following ways: [7]

- CORBA did not just follow the lead of a single large corporation and it is very deliberately vendor neutral
- It specifies an interface description language (IDL) that allow the specification from interface to objects.
    - Data that the object makes public
    - Operations (with complete signature) that the object can respond to
- A network protocol (Internet InterOrb Protocol, OORP) is specified to allow requests be transmitted over TCP/IP

The most obvious technical problem is CORBA's complexity—specifically, the complexity of its API (application programming interface). Many of CORBA's API's are far larger than necessary. For example, CORBA's object adapter requires more than 200 lines of interface

[6] Michi Henning, "The Rise and Fall of CORBA", http://queue.acm.org/detail.cfm?id=1142044
[7] Oracle Corporation, "Oracle8i Enterprise JavaBeans and CORBA developer's guide, release 8.1.6" https://docs.oracle.com/cd/A81042_01/DOC/java.816/a81356/corba.htm

definitions, even though the same functionality can be provided in about 30 lines—the other 170 lines contribute nothing to functionality, but severely complicate program interactions with the CORBA runtime.[4]

Although CORBA provided quite rich functionality, it failed to provide two core features:[4]

- Security - all traffic was unencrypted, subject to eavesdropping and man-in-the middle attacks
- Versioning - trying to version a CORBA application breaks the on-the-wire contract between client and server. This forces all parts of deployed application to be replaced at once.

3.1. Interface Definition Language (IDL)

The first step to creating a CORBA application is to specify all objects and their interfaces in Object Management Group's (OMG) Interface Definition Language. This file is used to generate both client and server implementations.[7]

3.2. Example

Figures 3.1, 3.2 and 3.3 define all the necessary components needed for a CORBA service. A typical "Hello World" type of program was chosen to emphasis the bare minimum configuration needed.

Figures respectfully define the following parts of an SOA service:

- Figure 3.1 - common configuration known to both provider and consumer
- Figure 3.2 - implementation of provider
- Figure 3.3 - implementation of consumer

This example was taken from Oracle's materials. All examples are unmodified to adhere to Oracle's Terms of Use. [8]

[8] Oracle Inc.,"Java IDL: The Hello World Example", http://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html

```
module HelloApp
{
  interface Hello
  {
  string sayHello();
  oneway void shutdown();
  };
};
```
Figure 3.1. HelloApp.idl

3.2.1. Provider

```
// HelloServer.java
// Copyright and License
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

import java.util.Properties;

class HelloImpl extends HelloPOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // implement sayHello() method
  public String sayHello() {
    return "\nHello world !!\n";
  }

  // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }
}
public class HelloServer {
  public static void main(String args[]) {
    try{
```

```
    // create and initialize the ORB
    ORB orb = ORB.init(args, null);

    // get reference to rootpoa & activate the POAManager
    POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
    rootpoa.the_POAManager().activate();

    // create servant and register it with the ORB
    HelloImpl helloImpl = new HelloImpl();
    helloImpl.setORB(orb);

    // get object reference from the servant
    org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
    Hello href = HelloHelper.narrow(ref);

    // get the root naming context
    // NameService invokes the name service
    org.omg.CORBA.Object objRef =
        orb.resolve_initial_references("NameService");
    // Use NamingContextExt which is part of the Interoperable
    // Naming Service (INS) specification.
    NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

    // bind the Object Reference in Naming
    String name = "Hello";
    NameComponent path[] = ncRef.to_name( name );
    ncRef.rebind(path, href);

    System.out.println("HelloServer ready and waiting ...");

    // wait for invocations from clients
    orb.run();
   }

    catch (Exception e) {
      System.err.println("ERROR: " + e);
      e.printStackTrace(System.out);
    }

    System.out.println("HelloServer Exiting ...");
  }
}
```

Figure 3.2. HelloServer.java

3.2.2. Consumer

```
// Copyright and License

import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient
{
  static Hello helloImpl;

  public static void main(String args[])
    {
      try{
        // create and initialize the ORB
        ORB orb = ORB.init(args, null);

        // get the root naming context
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");
        // Use NamingContextExt instead of NamingContext. This is
        // part of the Interoperable naming Service.
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // resolve the Object Reference in Naming
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("Obtained a handle on server object: " + helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();

        } catch (Exception e) {
          System.out.println("ERROR : " + e) ;
          e.printStackTrace(System.out);
          }
    }

}
```
Figure 3.3. HelloClient.java

4. SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

The goal of this chapter is to give a brief overview of the history and key concepts of SOAP.

SOAP was started in March 1998. Back then extensible markup language (XML) was still in its early years and because of that most of the focus was spent on defining a type system. There existed serialization formats and remote procedure call (RPC) protocols, which could have suited, but Don Box and the other authors of SOAP wanted to satisfy 80% cases elegantly and still allow the system to be bent for the remaining 20%. Due to Microsoft politics, SOAP specification was not shipped until 4th quarter of 1999. [9]

In that specification SOAP defines a remote procedure call (RPC) mechanism using XML for client-server interaction across a network by using the following mechanisms: HTTP as the base transport and XML documents for encoding of invocation requests and responses. [9]

4.1. XML – extensible markup language

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. [10]

The design goals for XML were:

1. XML must be straightforwardly usable over the Internet.
2. XML must support a wide variety of applications.
3. XML must be compatible with SGML.
4. It must be easy to write programs which process XML documents.

[9] G. Kakivaya, A. Layman, S. Thatte, “SOAP:Simple Object Access Protocol”, https://tools.ietf.org/html/draft-box-http-soap-00

[10] Tim Bray, Jean Paoli, C.M.Sperberg-McQueen, Eve Maler, François Yergeau, “Extensible Markup Language (XML) 1.0 (Fifth Edition)”, https://www.w3.org/TR/REC-xml/

5. The number of optional features in XML must be kept to the absolute minimum, ideally zero.
6. XML documents must be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML must be formal and concise.
9. XML documents must be easy to create.
10. Terseness in XML markup is of minimal importance.

4.2. WSDL – Web Services Description Language

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. [11]

4.3. Example

Figures 4.1, 4.2 and 4.3 define the XML's related to an SOAP service. A "Hello World" type of program was used to bring out the versions needed for a minimal setup.

Figures respectfully define XML's on three sides:

- Figure 4.1 - WSDL, knows for both producer and consumer
- Figure 4.2 - example request from consumer to producer
- Figure 4.3 - example response from producer to consumer

This example was taken from TutorialsPoint and complies with its Terms of Use. [12]

[11] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, "Web services Description Language(WSDL) 1.0", http://xml.coverpages.org/wsdl20000929.html
[12] Tutorials Point (I) Pvt. Ltd, "WSDL, Web Service Description Language", http://www.tutorialspoint.com/wsdl/wsdl_tutorial.pdf

```
<definitions name="HelloService"
  targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
  </message>

  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>

  <portType name="Hello_PortType">
    <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
    </operation>
  </portType>

  <binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="encoded"/>
      </input>

      <output>
```

```
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="encoded"/>
      </output>
    </operation>
  </binding>
  <service name="Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding="tns:Hello_Binding" name="Hello_Port">
      <soap:address
        location="http://www.examples.com/SayHello/" />
    </port>
  </service>
</definitions>
```

Figure 4.1. HelloService.wsdl

4.3.1. Consumer

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:SayHelloRequest xmlns:m="http://www.examples.com/wsdl/HelloService.wsdl">
<firstName>Rait</firstName>
</m:SayHelloRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4.2 Example request

4.3.2. Producer

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:SayHelloResponse xmlns:m="http://www.examples.com/wsdl/HelloService.wsdl">
<greeting>Hello, Rait</greeting>
</m:SayHelloResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4.3 Example response

5. OSGI

The goal of this chapter is to provide a brief introduction to OSGi.

OSGi is a framework for Java in which units of resources called bundles can be installed. Bundles can export services or run processes, and have their dependencies managed, such that a bundle can be expected to have its requirements managed by the container. Each bundle can also have its own internal classpath, so that it can serve as an independent unit, should that be desireable. All of this is standardized such that any valid OSGi bundle can theoretically be installed in any valid OSGi container.[13]

5.1. Example

Figures from 5.1 to 5.6 define minimal configuration needed for an OSGi service. They are divided into separate chapters to emphasise the three sides - common, provider, consumer.

The example was taken from Baptiste Wicht blog. The use of this is was in accordance with the "Terms of Use" document under his website.[14]

```
package com.bw.osgi.provider.able;

public interface HelloWorldService {
   void hello();
}
```

Figure 5.1. HelloWorldService.java

5.1.1. Provider

```
package com.bw.osgi.provider.impl;

import com.bw.osgi.provider.able.HelloWorldService;

```

[13] Joseph Ottinger, "OSGi for Beginners", http://www.theserverside.com/news/1363825/OSGi-for-Beginners
[14] Baptiste Wicht, "OSGI Hello World Services", http://baptiste-wicht.com/posts/2010/07/osgi-hello-world-services.html

```
public class HelloWorldServiceImpl implements HelloWorldService {
   @Override
   public void hello(){
      System.out.println("Hello World !");
   }
}
```

Figure 5.2. ProviderActivator.java

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>OSGiDmHelloWorldProvider</groupId>
   <artifactId>OSGiDmHelloWorldProvider</artifactId>
   <version>1.0</version>
   <packaging>bundle</packaging>

   <dependencies>
      <dependency>
         <groupId>org.apache.felix</groupId>
         <artifactId>org.osgi.core</artifactId>
         <version>1.4.0</version>
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.0.2</version>
            <configuration>
               <source>1.6</source>
               <target>1.6</target>
            </configuration>
         </plugin>

         <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <extensions>true</extensions>
            <configuration>
               <instructions>
                  <Bundle-SymbolicName>OSGiDmHelloWorldProvider</Bundle-SymbolicName>
                  <Export-Package>com.bw.osgi.provider.able</Export-Package>
                  <Bundle-Activator>com.bw.osgi.provider.ProviderActivator</Bundle-Activator>
                  <Bundle-Vendor>Baptiste Wicht</Bundle-Vendor>
               </instructions>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

Figure 5.3. Provider pom.xml

5.1.2 Consumer

```
package com.bw.osgi.consumer;

import javax.swing.Timer;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import com.bw.osgi.provider.able.HelloWorldService;

public class HelloWorldConsumer implements ActionListener {
   private final HelloWorldService service;
   private final Timer timer;
   public HelloWorldConsumer(HelloWorldService service) {
      super();
      this.service = service;
      timer = new Timer(1000, this);
   }
   public void startTimer(){
      timer.start();
   }
   public void stopTimer() {
      timer.stop();
   }
   @Override
   public void actionPerformed(ActionEvent e) {
      service.hello();
   }
}
```

Figure 5.4. HelloWorldConsumer.java

```
package com.bw.osgi.consumer;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
import com.bw.osgi.provider.able.HelloWorldService;

public class HelloWorldActivator implements BundleActivator {
   private HelloWorldConsumer consumer;

   @Override
   public void start(BundleContext bundleContext) throws Exception {
      ServiceReference reference =
bundleContext.getServiceReference(HelloWorldService.class.getName());

      consumer = new HelloWorldConsumer((HelloWorldService)
bundleContext.getService(reference));
      consumer.startTimer();
   }

   @Override
   public void stop(BundleContext bundleContext) throws Exception {
      consumer.stopTimer();
   }
}
```

Figure 5.5. HelloWorldActivator.java

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>OSGiDmHelloWorldConsumer</groupId>
   <artifactId>OSGiDmHelloWorldConsumer</artifactId>
   <version>1.0</version>
   <packaging>bundle</packaging>

   <dependencies>
      <dependency>
         <groupId>org.apache.felix</groupId>
         <artifactId>org.osgi.core</artifactId>
         <version>1.0.0</version>
        </dependency>
        <dependency>
            <groupId>OSGiDmHelloWorldProvider</groupId>
            <artifactId>OSGiDmHelloWorldProvider</artifactId>
            <version>1.0</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.0.2</version>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.felix</groupId>
                <artifactId>maven-bundle-plugin</artifactId>
                <extensions>true</extensions>
                <configuration>
                    <instructions>
                        <Bundle-SymbolicName>OSGiDmHelloWorldConsumer</Bundle-SymbolicName>
                        <Bundle-Activator>com.bw.osgi.consumer.HelloWorldActivator</Bundle-Activator>
                        <Bundle-Vendor>Baptiste Wicht</Bundle-Vendor>
                    </instructions>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```
Figure 5.6. Consumer pom.xml

6. FEIGN

The goal of this chapter is to give a brief introduction to Feign.

Feign is a Java to HTTP client binder inspired by Retrofit, JAXRS-2.0, and WebSocket. Feign's first goal was reducing the complexity of binding Denominator uniformly to HTTP APIs regardless of restfulness.[15]

The main goal of Feign is to make writing web service clients easier - all that is needed is to create an interface and annotate it. Feign only provides a framework for writing client side (consumer in terms of SOA key concepts) - server side must be implemented

6.1. Example

Figure 6.1 emphasises the small amount of code needed for a client implementation. Although a “Hello World” type of program was used for other examples, this “GitHub contributor list query” consumer presents how easy it is to define a slightly more complex implementation.

```
/*
 * Copyright 2013 Netflix, Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package feign.example.github;

import feign.Feign;
```

[15] Adrian Cole, “Feign README.txt, https://github.com/OpenFeign/feign

```
import feign.Param;
import feign.RequestLine;
import feign.gson.GsonDecoder;
import java.util.List;

/**
 * Inspired by {@code com.example.retrofit.GitHubClient}
 */
public class GitHubExample {
  interface GitHub {
    @RequestLine("GET /repos/{owner}/{repo}/contributors")
    List<Contributor> contributors(@Param("owner") String owner, @Param("repo") String repo);
  }

  static class Contributor {
    String login;
    int contributions;
  }

  public static void main(String... args) {
    GitHub github = Feign.builder()
                     .decoder(new GsonDecoder())
                     .target(GitHub.class, "https://api.github.com");

    // Fetch and print a list of the contributors to this library.
    List<Contributor> contributors = github.contributors("netflix", "feign");
    for (Contributor contributor : contributors) {
      System.out.println(contributor.login + " (" + contributor.contributions + ")");
    }
  }
}
```
Figure 6.1 Feign Github client example.

7. COMPARISON

Considering all technologies have their advantages and disadvantages, it is fairly hard to present a meaningful use case for all of them. With fairly large requests it is known that XML-serialization will not perform as good as JSON or binary formats do. Furthermore to get SOAP to perform best, one needs to know which data types are thread-safe and configure serialization according to that. The goal of this comparison is not to bring out situations where one technology Will definitely perform better, but rather finding an equally suitable test case.

Taking that into account, there is one test case from which all programmers (should) start - the infamous “Hello, World!”. As this example is also used for designing any new SOA implementations, then it should not give any technology an unfair advantage.

Additionally to having a common service provider capability for all technologies, a few other conditions must be met to provide a meaningful and comparable set of data. The following list summarizes the conditions that must be met for all SOA implementation technologies:

1. All applications must be written in the same programming language
    - As OSGi and Feign are only available in Java, then it is the only valid option
2. Each provider and consumer must contain minimal logic necessary for the test
3. Sample size of data sets per technology must be at least 5
4. Sample size of requests per experiment must be at least 100
5. The computers running the applications for different technologies must be as similar as possible

All test cases are executed from a main class as shown on figure 6.1. All necessary configuration for each technology is in each implementation class under setup() method.

```
import java.util.concurrent.TimeUnit;

public class TestSOA {
   public static void main(String[] args) throws InterruptedException {
      for (int i = 0; i < 1; i++) {
         testSoa(new CorbaServiceImpl());
      }
   }
   private static void testSoa(SoaService service) throws InterruptedException {
      long start = System.nanoTime();
      service.setup();
      long end = System.nanoTime();
      System.out.println(String.format("Server setup %d nanoseconds", end - start));

      long sum = 0L;
      for (int i = 0; i < 100; i++) {
         start = System.nanoTime();
         String response = service.execute();
         end = System.nanoTime();
         System.out.println(String.format("Execution %d finished. Response retrieved in %d
nanoseconds", i + 1,  end - start));
         sum+= end-start;
         TimeUnit.SECONDS.sleep(10);
      }
      System.out.println("Average = " + (sum /100));
      System.out.println("---------------------------");
   }
}
```

Figure 6.1. Main class running the test cases.

Technology	Request-response format	Security	Versioning of services	Average execution time of 5*100 requests
CORBA	binary	none	not recommended	4,56 ms
SOAP (JAX-WS)	XML	HTTPS if need be	easily possible	21,30 ms
OSGi	XML	HTTPS if need be	easily possible	16,80 ms

Feign	JSON	HTTPS if need be	easily possible	10,00 ms

Figure 6.2. General comparison between technologies

From figure 6.2 the following statements can be made:

- CORBA has the fastest execution time
- SOAP (JAX-WS) and OSGi are the slowest
- Feign average from the four
- If OSGi provider and consumer were in the same application then execution time was averaging around 0,50 ms. This points to slow XML serialization and SOAP (JAX-WS) results confirm it.
- CORBA lacks security features and is not viable for situations handling sensitive data
- CORBA lacks versioning features and is not viable for situations needing highly available producers

Technology	1st set	2nd set	3rd set	4th set	5. request
CORBA	5702 ns	5051 ns	3969 ns	4126 ns	4043 ns
SOAP (JAX-WS)	20145 ns	24512 ns	19542 ns	22124 ns	20198 ns
OSGi	15839 ns	16483 ns	15543 ns	18606 ns	17543 ns
Feign	10610 ns	8860 ns	9792 ns	9928 ns	10822 ns

Figure 6.3. Execution average per 100 non-parallel requests

SUMMARY

The aim of this thesis was to compare different implementations of SOA.

In the first part of the a general overview of service-oriented architecture was given including key concepts used to describe any SOA implementation. Second part of the thesis gave an overview of every technology used for comparison. Third part compared the four technologies with each other in terms of

In terms of execution time of a “Hello World” type of program, CORBA has a clear advantage over Feign, JAX-WS (using SOAP) and OSGi (using SOAP). That being said, CORBA is not suitable for any situation, which handles sensitive data or which needs producer to be highly available. In those situations Feign, AX-ws or OSGI should be used.

In terms of complexity only CORBA stands out - the rest

REFERENCES

1. Kristopher Sandoval “Microservices Showdown - REST vs SOAP vs Apache Thrift (And Why It Matters) http://nordicapis.com/microservice-showdown-rest-vs-soap-vs-apache-thrift-and-why-it-matters/
2. Tim Bray, “Loyal opposition to Web Services” (2004)
3. OASIS, “Reference Model for Service Oriented Architecture 1.0” https://docs.oasis-open.org/soa-rm/v1.0/soa-rm.doc
4. Merriam-Webster, “Service | Definition of Service by Merriam-Webster”, http://www.merriam-webster.com/dictionary/service
5. Service Oriented Architecture (SOA), https://msdn.microsoft.com/en-us/library/bb833022.aspx
6. Michi Henning, “The Rise and Fall of CORBA”, http://queue.acm.org/detail.cfm?id=1142044
7. Oracle Corporation, “Oracle8i Enterprise JavaBeans and CORBA developer's guide, release 8.1.6” https://docs.oracle.com/cd/A81042_01/DOC/java.816/a81356/corba.htm
8. Oracle Inc.,”Java IDL: The Hello World Example”, http://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html
9. Don Box, A brief History of SOAP, http://www.xml.com/pub/a/ws/2001/04/04/soap.html
10. G. Kakivaya, A. Layman, S. Thatte, “SOAP:Simple Object Access Protocol”, https://tools.ietf.org/html/draft-box-http-soap-00
11. Tim Bray, Jean Paoli, C.M.Sperberg-McQueen, Eve Maler, François Yergeau, “Extensible Markup Language (XML) 1.0 (Fifth Edition)”, https://www.w3.org/TR/REC-xml/
12. Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, “Web services Description Language(WSDL) 1.0”, http://xml.coverpages.org/wsdl20000929.html

13. Tutorials Point (I) Pvt. Ltd, "WSDL, Web Service Description Language", http://www.tutorialspoint.com/wsdl/wsdl_tutorial.pdf
14. Joseph Ottinger, "OSGi for Beginners", http://www.theserverside.com/news/1363825/OSGi-for-Beginners
15. Baptiste Wicht, "OSGI Hello World Services", http://baptiste-wicht.com/posts/2010/07/osgi-hello-world-services.html
16. Adrian Cole, "Feign README.txt, https://github.com/OpenFeign/feign

All links had been checked on 10th of Aug 2016.

Appendix

Licence