Delft University of Technology
Software Engineering Research Group
Technical Report Series

Performance Trade-offs in Client-Side Service Delegation

Khalid Adam Nasr, Hans-Gerhard Gross and Arie van Deursen

Report TUD-SERG-2011-021

TUD-SERG-2011-021

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
`http://www.se.ewi.tudelft.nl/techreports/`

For more information about the Software Engineering Research Group:
`http://www.se.ewi.tudelft.nl/`

Note: Accepted for publication in the Proceedings of the International Symposium on Web Systems Evolution (WSE), 2011, IEEE Computer Society.

Performance Trade-offs in Client-Side Service Delegation

Khalid Adam Nasr
Delft University of Technology
Software Engineering Research Group
Delft, The Netherlands
Email: k.a.nasr@tudelft.nl

Hans-Gerhard Gross
Delft University of Technology
Software Engineering Research Group
Delft, The Netherlands
Email: h.g.gross@tudelft.nl

Arie van Deursen
Delft University of Technology
Software Engineering Research Group
Delft, The Netherlands
Email: Arie.vanDeursen@tudelft.nl

***Abstract*—Service Oriented Architecture, which builds on distributed computing platforms, is increasingly being adopted by organizations in both public and private sectors. Migration from traditional monolithic systems to services, in particular web services, characterizes much of systems evolution today. This paper analyzes some of the performance and modularization problems involved in current service-oriented computing. It investigates under which circumstances the communication between service providers and service consumers can be made more efficient by eliminating certain steps from traditional Remote Procedure Call (RPC) methods. After discussing traditional service invocation and its drawbacks, this paper proposes an alternative approach called Distributed Service Delegates (DSD). DSD is based on emphasizing client-side or local computations. An experiment is designed and implemented to measure the trade offs between traditional methods, in this case Web services, and the proposed DSD. The results of this experiment are discussed and its implications for future research are indicated.**

I. Introduction

Service Oriented Architecture (SOA) is widely being adopted by organizations for realization of web information systems, because it enables them to respond rapidly to business changes, to integrate their systems with business partners and to achieve flexible application assembly. At the same time, SOA is meant to reduce software maintenance and evolution costs. The most prominent feature of current SOA implementations is their focus on achieving the ultimate 'software as a service' goal. This is done by making applications internally loosely coupled, which enables the quick introduction of new solutions for clients. Secondly, their focus is on integrating previously autonomous systems by using standards. Recent studies indicate that SOA is the best available option for system integration [1]. At the same time, as we have argued in [2], system evolution and migration to service solutions are often adopted without a solidly proven positive return on investment (ROI) for the organization concerned. As part of their migration to services, organizations should therefore think carefully about the most suitable architecture and consider performance trade-offs. This paper extends this argument by focusing on performance and efficiency issues of intra-organizational system communication. In other words, we focus on communication between systems within one organization, not on their communication with external devices.

A. Service-oriented computing

SOA represents a new generation distributed computing platform, which builds on the service-oriented computing paradigm. The essence of this paradigm is the provision of business functions as services, whereby the services are realized through software components [3]. The services are published by service providers and used by, usually more than one, consumer or client interested in the specific service offer. Technically speaking this means that an application running on a client's platform (consumer) causes a procedure to be executed on another application running on a provider's platform. This concept is known as Remote Procedure Call (RPC) and it is an inherent element of distributed and network-based architectures. Accessing functionality using RPC, or one of its derivatives, such as Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Web services, is the most widely spread communication method used in such architectures. The leading standards-based technology for implementing SOA systems today is Web services [4]. However, industry practice as well as several studies have indicated the poor performance of web services, in part due to encoding issues [5], [6], [7], [8].

B. Motivation

From our experience with large-scale web systems migration projects in industry [2], we learned that the communication between providers and consumers in distributed systems is often inefficient and moreover, that this issue is often insufficiently addressed during the early phases of a project. With the current methods, in order to execute a business function or service computation needs to happen on both the consumer's platform (by means of proxies/stubs) and the provider's platform (the implementation of proxies/stubs). It is our hypothesis that, under certain circumstances, performance can be improved by eliminating one of these computation steps. The computation that happens on the provider's platform, invoked by the consumer, is in many cases either a validation or calculation related to a specific business rule, or a retrieve, update or delete on data. Instead of invoking the provider's platform for all these computations, we investigate whether, how, and when these could be executed locally on the consumer's platform. The advantage of local instead of remote invocation would be less request-reply communication between consumer and provider and therefore more efficient execution of business functions and less network traffic. We believe that, because today the consumer and provider are often equal in terms of processing power and binary representation, the local computation of services is increasingly feasible. The notion of client/server is much less relevant, or indeed, as Kumaran has said, 'With processors and memory becoming cheaper, a redefinition of the "computing device" is on the horizon' [9].

Our research questions are:

- **RQ1**: Can the performance of service applications be improved through replacing traditional remote invocation by an alternative method of local invocation?
- **RQ2**: How can the difference in performance between the traditional and the alternative method be explained?
- **RQ3**: Under which circumstances is it sensible to apply the alternative method and what are the performance trade-offs?

In section 2 we discuss related work. In section 3 we analyze how traditional remote invocation works and what its drawbacks are. In section 4 we propose an alternative method of service-oriented programming, based on local invocation, which in this paper we will refer to as Distributed Service Delegates (DSD). We illustrate this method in section 5 with an experiment comparing the performance of Web services with DSD. In section 6, the discussion, we reflect on what the experiment tells us about our research questions. We conclude with section 7.

II. Related Work

Our work in this paper touches on the area of distributed architecture and on the issue of communication performance between systems through RPC and its derivatives, in particular web services. Researchers have studied and compared the performance of different distributed application paradigms. Rozman et al. [10] conducted a comparison and performance study for Java distributed architectures. Their qualitative and quantitative analysis included three groups: Java native distributed architecture RMI and secure RMI-SSL; HTTP-to-port and HTTP-to-CGI/servlet; and RMI tunneling techniques and Web services and WS-Security. Juric et al. [11] studied the different distributed object models, such as Common Object Request Broker Architecture (CORBA) and Remote Method Invocation (RMI) for Java. The authors evaluated the performance of these distributed object models in order to guide developers in making the right choice about which object model to use.

Another set of studies concerns web services in their different styles. Christensen et al. [12] and Papazoglou et al. [13] discuss Remote Invocation, traditional Remote Procedure Calls and document-oriented services; Fielding and Taylor [14] discuss Representational State Transfer (REST). All build on knowledge based on distributed architecture technologies (CORBA and RMI). Web services have the advantage of a standardized application-to-application communication and interoperability. However, their performance, which is of concern to our study, has been the subject of several studies, including [15], [5], [16], [6], [7], [17], [8]. These authors discuss some of the performance bottlenecks, such as encoding problems and network latency, and suggest solutions such as message bundling, caching, or alternative technologies.

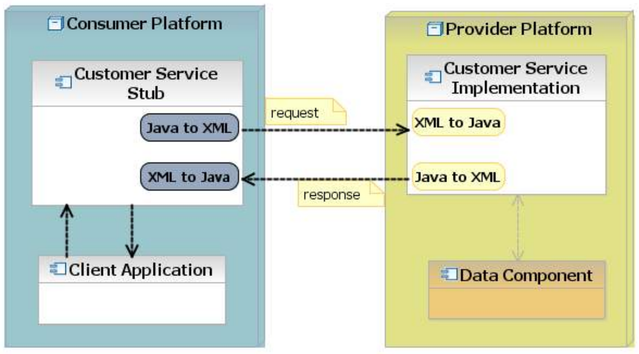


Figure 1. Traditional service invocation

The Jini networking technology [18] is of particular interest to this study. Jini is an open architecture networking technology that enables different services within the same network to interact in an easy manner. The difference between Jini and the solution we propose in section IV of this paper is that Jini was developed specifically for Java platforms. Moreover, Jini is meant to simplify the sharing of resources, tasks and a network of devices, while our solution is rather focused on executing business functions. Our solution could benefit from Jini by using their code download mechanism.

An ongoing discussion concerns the desirability of distributing software objects. Fowler formulated his controversial 'Don't distribute your objects!' law [19]. While distributed systems are not the silver bullet for every problem, sometimes distribution cannot be avoided. In large organizations this is often the case and not least due to organizational governance issues and the fact that different departments want to retain control over their services [2]. This paper argues that if organizations need to distribute, they need to think carefully about what is the most appropriate set-up for their systems.

III. Traditional service invocation in service-oriented programming

In this section, we analyze the merits and drawbacks of traditional web service invocation by means of an example taken from a real-life industry project.

The traditional approach of realizing web service invocation is illustrated in Figure 1. Service development always involves two types of parties with their associated execution platforms. The first party, service providers, design their applications in a loosely coupled manner and deploy their business logic or individual functions as services on the provider platform. The second party, the service consumers, connect their client application running on the consumer platform. Web services are described as a set of endpoints operating on messages using an XML format known as Web Services Description Language (WSDL) [20]. Request and reply are transformed into a text format, typically XML, in order to realize independence of the clients' and servers' binary representations.

An advantage of web services is that the developers of the client application do not require any knowledge of the underlying implementations of the services. They generate stubs from the WSDL that use the Simple Object Access Protocol (SOAP) [21] for their invocations and in order to access the service. When the service consumer requests a function (whether a simple data request or a composite business function) from the service provider, the latter usually does some validation and internal processing before responding to the requester. What sounds like a straightforward process on paper, can, in practice, involve unnecessary detours.

A. Scenario

We illustrate this by means of the example of a Customer Service, which was included in a real-life SOA project. Two of the business functions that are part of this Customer Service are 'open account for new customer' and 'retrieve a list of existing customer(s)'. The business analyst has defined the so-called business rules for the Customer Service and for each of its business functions. For 'opening account for new customer' the business rules are: the applicant must be 18 years or older, s/he must be a resident and living in the country, s/he must not have outstanding loans greater than 10,000 Euros, s/he must not be on the black list for clients and s/he must not be an existing customer. For the the business function 'retrieve a list of existing customer(s)' the rules define that retrieval can take place based on either ID number or name, or as 'retrieve all'.

In order to make it technically possible for remote clients to invoke these two business functions on the provider's platform, the developers of both the client (consumer) and server (provider) applications need to perform a considerable number of steps.

On the server side, the developer first has to write code to implement the business logic (service implementation).

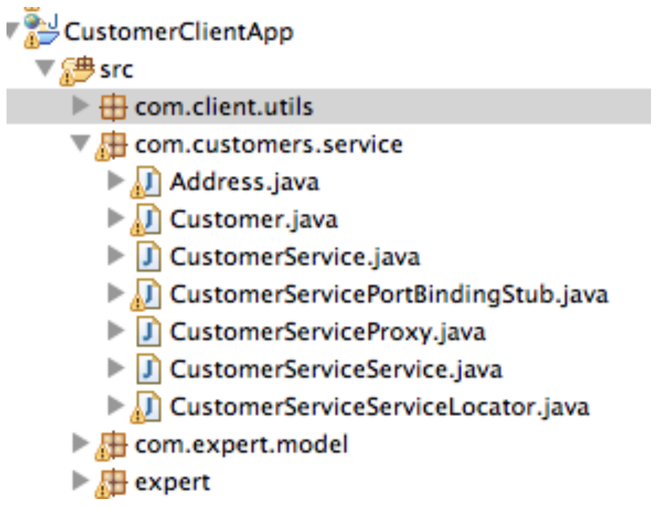

```
CustomerClientApp
  src
    com.client.utils
    com.customers.service
      Address.java
      Customer.java
      CustomerService.java
      CustomerServicePortBindingStub.java
      CustomerServiceProxy.java
      CustomerServiceService.java
      CustomerServiceServiceLocator.java
    com.expert.model
    expert
```

Figure 2. The generated web service client

Then, to provide the business functions to remote clients, the developer specifies the web service operations by defining methods in an interface and s/he codes one or more classes that implement these methods. The web service description is then published to clients by generating a WSDL file and XSD schema. Lastly, the web service is deployed to an application server.

On the client side, the developer generates client-side stubs from the WSDL in order to access the service (see Figure 2). These stubs hide the implementation details of the service, but expose the operations of the service (name, input and output types, and parameters). In order to invoke these operations on the provider's application, the client developer codes a proxy (a local object representing the service), creates a web service port object for the web service, and, finally, invokes the methods on this object. This is illustrated in Listing 1.

The same steps need to be taken for each additional business function that the Customer Service provides.

B. Interaction between client and provider

When, in our scenario, a remote client wants to open a new customer account, the client application first executes a local call to the client stub, which transforms this request into a SOAP request. This is called marshalling, and it is realized by converting a Java content tree to XML data. The SOAP request is then sent over the network using HTTP protocol. The service implementation on the server unmarshalls the request back into a tree of Java content objects. The requested computation is then executed and the result is returned in a SOAP response to the client stub using HTTP protocol again. Once more, the client stub needs to unmarshall the received SOAP response before the client application can use the received result.

In our scenario, network communication for the first

```
try {

    // Make a service
    CustomerServiceService service
        = new CustomerServiceServiceLocator();

    // Now use the service to get a stub which
    // implements the Service Definition Interface
    CustomerService port
        = service.getCustomerServicePort();

    // Make the actual call
    port.getAll();

}

catch (ServiceException e1) {
    e1.printStackTrace();
}

catch (RemoteException er) {
    er.printStackTrace();
}
```

Listing 1. Customer service call

business function 'open account for a new customer' is considerable. However, the long route of remote invocation and computation may well be unnecessary for merely performing the simple validations of age, residency, outstanding loans, etc. In case of the second business function 'retrieve list of existing customer(s)' the same is true, and even compounded by the fact that for each retrieved customer object the process of marshalling and unmarshalling needs to take place.

C. Limitations

The performance problems of web services have been discussed in several studies, as mentioned in Section II. The above scenario indicates that Remote Procedure Call (RPC) technology, even though it is one of the most widely used communication methods for distributed systems, indeed has performance drawbacks. The disadvantage of expensive remote method calls have also been discussed by [7], [22], [23].

From a software development perspective, the above discussed traditional approach has other limitations too. This is due to the fact that service providers and consumers strongly depend on each other for the testing and implementation of their services. Several studies, including [24], [25], [26], point at the challenges of testing and integration testing in a service-oriented context. A case study by the authors [2] shows how this mutual dependency in a software project that involves the development and testing of web services and runtime execution of applications using web service technology, creates the risk of not meeting the initially planned functionality or to deliver on time. In the worst case, projects fail as a result.

IV. Distributed Service Delegates (DSD) Architecture

We propose an alternative approach to service-oriented programming in order to address some of the above-mentioned limitations. This proposal, which we refer to as Distributed Service Delegates, is primarily meant to take away the performance or inefficiency disadvantage of remote service invocation. It is, by no means, meant to replace all remote method invocation in a distributed system. Instead, we argue, that its application should be carefully weighed against the potential benefits that it might provide in a distributed application. In particular, we aim at intra-organizational distributed method invocations.

DSD essentially boils down to service providers no longer having to expose their business functions as serviced systems themselves, and, as a consequence, service consumers no longer having to invoke these services remotely. Instead, service providers only need to clearly identify and define their business logic and rules, and expose these as delegate packages to their potential consumers. It is then up to these consumers to download and execute these functions, and perform all necessary validation locally, after which they can execute the final (ideally one-time) remote request to send data to be saved on the provider's server. Our hypothesis is that this approach will result in enhanced performance of the service invocation mechanism. Namely, using service delegates will reduce the number of transactions required to perform an operation using remote method invocation, and is thus expected to be more time efficient. Less network traffic is an additional expected advantage.

The DSD architecture, illustrated in Figure 3, consists of what we call client delegate and server delegate components.

A. Client delegates

Client delegates reside on the client's (service consumer) platform and run within the same process as the client application. Client delegates have an id and version number. They consist of the following four components. The first is the interface component, which provides interfaces for the client and server applications. The interfaces to the client's application provide well-defined entry points to the client delegate's implementation. They resemble procedural call type connectors and can be invoked by client applications (consumers of data) as local methods during the interaction between client applications and client delegates. When invoked, these interfaces participate in data access, data collection and the modification of data. After the data is accessed, it is transferred between components using parameters and return values. The interfaces to the server application are the entry points to deliver the client delegate's computation results. The result is delivered to the server application (owner of data) in a single delivery.

The second component is the security component, which maintains the integrity of the information on the server by

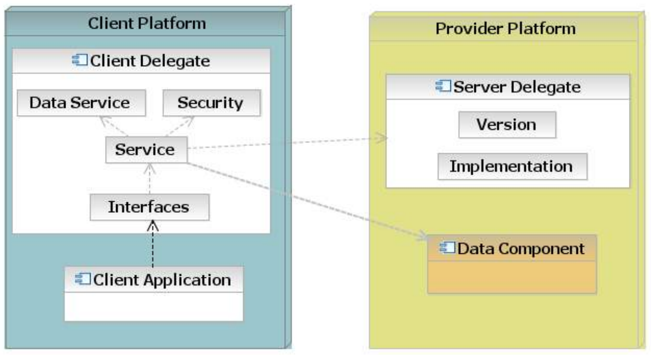


Figure 3. Distributed Service Delegates (DSD) architecture

only allowing requests from authorized client delegates. To determine the legitimacy of a client delegate, its identity is verified using an authentication procedure, which includes a combination of the client delegate's id and password. Authentication can be applied at the beginning of the communication session or for each data packet. The latter is the most secure option, however, the trade-off is slower performance. Since security issues are of no concern in this paper, the performance experiments with the DSD presented in Section V use the simple one-time authentication method at the beginning of the communication session.

The third component is the data service component. This component is responsible for providing data to the client delegate, which uses these data for the validation of business rules. The data can be requested by the client application for generating reports and lists. The data can be either stored locally in an embedded database packaged with the client delegate, or requested from the remote data owner (server provider) in case of sensitive data.

The fourth component is the service component, which can be seen as a local component available to be consumed by the client application. The service encapsulates the business functions of the client delegate. Through this component, business rules such as validate, create, delete, retrieve and calculate operations are executed. These functions represent the service implementation. They are exposed to the client application through the interfaces using parameters and return values. The service component is also responsible for starting a transaction with the server delegates (service provider) and establishing a secure connection in order to deliver results or retrieve data.

B. Server delegates

Server delegates are remote components, which reside on the server's (service provider's) platform. Server delegates are responsible for handling certain requests from client delegates and for version control. They consist of two components.

Firstly, the version component validates whether the current version of the client delegate complies with the latest version of the associated server delegate. In case of mismatches the client delegate request is rerouted to the old version implementation and the client is notified about the new version. The client delegate can then decide to request the updated version from the server delegate. In order to perform this action, the download mechanism developed by Jini may be a suitable method.

Secondly, the implementation component accepts the computed result from the client delegate. The implementation component performs the final insertion or deletion tasks on the database. The developers who devised the code for the server delegate is the same group of developers, or at least is from the same organization, that devises its client delegate. So, in principle, server delegates can trust their own client delegates. However, some service providers may not wish to export sensitive data to their client delegates. In this case, the client delegates perform a minimum set of tasks, and then return control to the server delegate, which performs the required business rules. This is similar to the traditional service paradigm, without involving the marshalling and unmarshalling steps.

C. Trade-offs

This paper sets out to investigate under which circumstances the above-described DSD method, which is based on local invocation, can replace traditional remote invocation methods in order to achieve more time efficient service-oriented computing. The DSD method is expected to have trade-offs related to:

- Interprobability. In projects where interprobability is a key requirement, the DSD method may be less suitable because it will require defining different delegates for each client platform. Web services on the other hand have the benefit of being platform independent.
- Type of service. The DSD method can be expected to work well for data services and validation services. It may be more complicated to apply DSD to composite business functions.
- Inter-organizational use. The DSD method requires more attention when used across different organizations with different platforms and with specific security regulations. With intra-organizational use of the DSD method this is not an issue.

V. Empirical Evaluation

As argued in the introduction, it is our hypothesis that, under certain circumstances, the performance of service-oriented applications can be improved by replacing the remote interactions between service provider and consumer with a local set-up. We introduced the DSD architecture in order to help us test this hypothesis. This section describes an experiment that we designed to compare service invocation using our proposed DSD model with the traditional method of, in this case, web services. We discuss the design and implementation of this experiment, its measurements and the interpretation thereof.

A. Experimental design

The objective of our experiment is to measure the overhead performance of the traditional web services method and to compare this to the overhead of our proposed DSD. This comparison will allow us to establish whether local instead of remote invocation of services can indeed make service-oriented computing more efficient, in certain cases. Keeping in mind the trade-offs listed in the previous section (Sect. IV-C), we design our experiment as follows:

- There are no specific requirements for interprobability.
- Our experiment is limited to data and validation services.
- The services are retrieved in a context of the same platform for both client and provider (intra-organizational use).
- Measuring the network overhead, which is not directly relevant to this experiment, is excluded from the comparison.

Our experiment is based on the data service invocation scenario described in Sect. III-A, which simulates a client requesting the 'retrieve list of existing customer(s)' business function from the provider of the Customer Service. We chose this common business scenario because it allows us to focus on testing our proposed solution in a straightforward way. In order to execute this experiment we develop two applications: the service provider and the service consumer application. The service consumer (client) application is designed to be able to retrieve data using both the traditional and the DSD methods. Both implementations (web services and DSD) are developed in Java to ensure a valid comparison with equal variables. We also develop a tool to execute the comparative test (see Figure 4).

B. Implementation

The *service provider* publishes the Customer Service. It is implemented as a document/literal (wrapped) style web service with the Java API for XML Web Services (JAX-WS) runtime platform. This runtime platform is used for building message-oriented and RPC web services and clients. JAX-WS makes the development and deployment of web services and clients less complicated by hiding the XML and SOAP messages complexity from the application developer.

The *@WebService* annotation in Listing 2 indicates to the Java interpreter that the methods of this class are intended to be published as a web service. The Customer Service contains the following operations:

- *getAll* – retrieves a list of all customers.
- *getById* – retrieves one single customer by ID.
- *getByName* – retrieves one single customer by name.
- *save* – saves a new entry.

```
package com.customers.service;

import javax.jws.WebService;

@WebService
public class CustomerService {

   public Customer[] getAll() {

      Object[] object
         = CustomerDAO.Factory.getInstance().
            findAll();
      Customer[] customer
         = new Customer[object.length];

      for (int i = 0; i < object.length; i++) {
         customer[i] = (Customer) object[i];
      }

      return customer;
   }
}
```

Listing 2. Customer service

- *delete* – deletes an existing entry.

Listing 2 illustrates how the getAll operation of the Customer Service returns a list of customers back when invoked by the client. The customer list is an array of customer objects. Each list item contains the following customer attributes:

- *id*
- *name*
- *address* including (street, postal code, city, country)
- *telephone*

The Java Persistence API (JPA) framework is used to manage the relational data in the application retrieved from the database. A *MySql* database is used to hold the customer data tables. The service is deployed on the Glassfish application server v3 as a web application.

The *service consumer* is implemented as a Java application running on the same platform. In this client application, we generate a web service client for the Customer Service from the service's Web Services Definition Language (WSDL) using the Axis WSDL2Java tool. Figure 2 shows the generated client Java artifacts.

The first method we add in the client application is designed to invoke the Customer Service using the traditional web service invocation through the generated service client. The second method we add is designed to invoke the Customer Service using the DSD implementation. The important difference between these two methods lies in the location of where the actual computation happens. The code for the computation itself is the same in both cases (see listing 3). In the first and traditional case, the computation of the actual business logic, which is called service implementation, happens on the provider's side after a request/invocation by the generated client. In the second

```
public Object[] findAll(String query) {

   logger.info("Entering findAll...");

   EntityManager e = getEntityManager();
   Object[] object = null;

   try {
      object = getEntityManager().createQuery(
          query).getResultList().toArray();

   } catch (Exception ex) {
      logger.info(ex.getMessage());
   }
   finally {
      e.close();
   }

   logger.info("Finished performing findAll...");

   return object;
}
```

Listing 3. Computation code

case, the computation happens locally at the client's side by the client delegate.

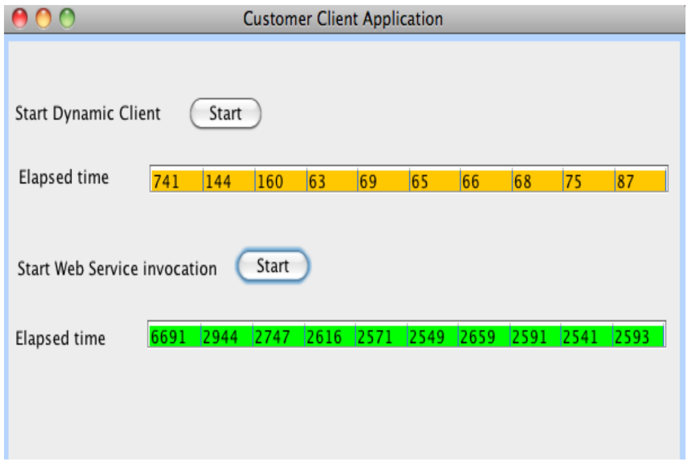
C. *Performance Measurements*

The comparative time measures in this experiment are conducted using the Java timer method, namely System.currentTime(). This method returns the current value of the most precise available system timer, in milliseconds (ms). In this experiment we compare the traditional and DSD performance by means of two comparative measurements using the getAll operation:

- (1a) The elapsed time the web service client takes to retrieve the result from the service implementation from the provider application.
- (1b) The elapsed time the client delegate takes to retrieve the result from the service implementation through DSD.
- (2a) The same measurement as in (1a) for 100 web service clients retrieving results simultaneously
- (2b) The same measurement as in (1b) for 100 service delegates retrieving results simultaneously

Figure 4 and table 1 show the results on these measurements. Note that the results reported in table 1 are the average of the 10 repetitions as shown in figure 4. To achieve a sufficient level of accuracy, all four invocations were repeated 100 times.

Table1		
Method	Measurement 1	Measurement 2
web services	1a: 3050 ms	2a: 281880 ms
DSD	1b: 154 ms	2b: 6553 ms



```
Elapsed time  741 | 144 | 160 | 63 | 69 | 65 | 66 | 68 | 75 | 87
Elapsed time  6691 | 2944 | 2747 | 2616 | 2571 | 2549 | 2659 | 2591 | 2541 | 2593
```

Figure 4. Measurement tool

D. *Interpretation of results*

These measurement results seem to confirm our hypothesis. Namely, the time elapses are shorter for the DSD method in both cases. In measurement 1, the difference is a factor 20; in measurement 2, the difference is a factor 43.

Our explanation for these differences that are in favor of the DSD method, is the location of the implementation code. The fact that in the DSD method this location is on the client side means that computation happens in one place instead of two (less computation executed) and there is less need for remote invocation. This means there is no time spent on marshalling and unmarshalling of objects, which creates a considerable efficiency gain. The fact that in measurement 2, the difference is a factor 43, compared to a factor 20 in measurement 1, suggests that the higher the number of simultaneous calls, the greater the efficiency gain of the local invocation.

VI. Discussion

In this section we discuss the answers to our research questions, identify threats to validity and point at issues that merit future research.

A. *Research questions*

RQ1 Can the performance of service applications be improved through replacing traditional remote invocation by an alternative method of local invocation? Yes, performance can be improved by using an alternative method of local invocation. The experiment presented in this paper, which makes use of our proposed Distributed Service Delegates architecture clearly shows this.

RQ2 How can the difference in performance between the traditional and the alternative method be explained? As discussed in section V.D, the difference is explained by means of the location of the implementation code. In the DSD method, computation happens by the client delegate, which is located on the client platform. In the traditional web services method, computation happens on both the client and the server side. The DSD method reduces the need for remote invocation, which means that less time is spent on marshalling and unmarshalling of objects, which creates a considerable efficiency gain.

RQ3 Under which circumstances is it sensible to apply the alternative method and what are the performance trade-offs? Our experiment shows that in the case of a data service local invocation is considerably more time efficient than traditional remote web service invocation. Therefore, we suggest that in the case of SOA projects where web systems are evolving to support services, organizations should carefully weigh the different performance trade-offs of the intended evolution. In the case of intra-organizational service set-up, where interprobability is not a complicating factor, it may be worth considering to use local service invocation instead of the common remote method invocation. This is certainly advisable in the case of straightforward data or validation services. In each situation the performance versus interprobability trade-off should be weighed in order to decide whether or not to use local invocation. Lastly, we suggest that organizations take a critical look at their governance set-up. It may turn out that the inclination of different business units to keep tight control over their systems is a drawback, and that the organization as a whole would benefit from a more open governance set-up in which business units allow easy access to their services by means of the client delegate method proposed in this paper.

B. *Threats to validity*

The internal threats to validity are, firstly, the fact that our experiment was performed on one platform, functioning as both provider and consumer platform. This was done to achieve more accurate timing and eliminate the network overhead, which is not relevant to our study that intended to prove the significance of the code location. [simulation of 100 clients vs. real users]. The second internal threat to validity is the fact that the measurement on the used platform, although it was a dedicated platform, includes operating system activities. However, this drawback was minimized by means of measuring 100 repetitions for both measurement 1 and 2. Thirdly, bad programming of the experiment code could influence the performance. This is very unlikely, however, in the case of this straightforward scenario, of which the code is available.

The external threat to validity is the fact that one experiment by definition is only representative to a limited extent. However, the validity of our experiment is strengthened by the fact that the business scenario is inspired by a real-life industry case [2] and by the fact that we measured performance for both 1 client and 100 clients retrieving results simultaneously.

C. *Future research*

The experiment discussed in this paper showed the advantage of using the DSD local invocation method in a situation of data services in an intra-organizational context. Future research is needed to obtain better insight into the scope of this finding in composite service contexts and when more complex business scenarios are at stake. Situations where interprobability is a strong requirement also need further investigation. Research will have to examine how the DSD method can be adapted to be applicable in such situations, and develop guidelines on which business functions can be delegated under which circumstances. In the case of interprobability this may require designing delegates for specific platforms (DSD is not meant to be exclusively applicable to Java, even if the experiment in this paper is limited to a Java platform) or finding alternative suitable solutions. Future research will also look into the possibilities of extending this solution to mobile devices.

A second area interesting for further research concerns testing. As discussed, testing and integration testing often pose challenges to service-oriented projects due to the mutual dependency of service consumers and service providers. We expect that the DSD method has advantages in this respect because the DSD tests against local delegates. Therefore, the consumer does not need to wait for the provider's systems to be up and running in the test/integration environment.

A third area for future research is to compare Remote Method Invocation (RMI) without web services to our proposed DSD.

Fourthly, in our experiment we did not include an embedded database within the client delegates in order to fully eliminate the remote database invocation. In our future research we wish to investigate the possibilities of embedding the database. This will bring with it challenges related to exposing sensitive data in the client delegate. Research can further explore how encryption can be incorporated into the DSD method to address this challenge.

VII. CONCLUSION

In this paper, we analyzed some of the performance issues related to service-oriented computing using our experience with large SOA migration projects involving web system evolution. We concluded that in certain situations the commonly used methods of remote web service invocation have inefficiency drawbacks. We designed an alternative method, called Distributed Service Delegates (DSD) which is based on local invocation by client delegates. We tested the DSD by means of an experiment which involved the retrieval of a data service. The experiment results showed clear performance gains for the DSD method when compared to traditional web service methods. We consider the following to be the key contributions of this paper:

- An analysis of some of the performance and modularization problems involved in traditional Service Oriented Architectures
- The formulation of Distributed Service Delegates, an alternative approach emphasizing client-side computations.
- An empirical study illustrating the trade-offs involved when choosing between traditional SOA and DSD architectures.

In addition, our paper indicates four areas for future research.

References

[1] G. Lewis, D. B. Smith, and K. Kontogiannis, "A research agenda for service-oriented architecture (soa): Maintenance and evolution of service-oriented systems," *CMU/SEI Technical Note*, 2010, http://www.sei.cmu.edu/library/abstracts/reports/10tn003.cfm.

[2] K. A. Nasr, H.-G. Gross, and A. van Deursen, "Realizing service migration in industrylessons learned," *Journal of Software Maintenance and Evolution: Research and Practice*, pp. n/a–n/a, 2011. [Online]. Available: http://dx.doi.org/10.1002/smr.540

[3] T. Erl, *SOA Design Patterns*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2009.

[4] G. Lewis, "Is soa being pushed beyond its limits?" *Microsoft Architecture Journal*, 2009, http://msdn.microsoft.com/en-us/architecture /aa699422.aspx, (accessed 10 April 2011).

[5] C. Kohlhoff, R. Steele, B. B. Rd, and N. Bay, "Evaluating soap for high performance business applications: Real-time trading systems," 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/ summary?doi=10.1.1.8.4280

[6] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the limits of soap performance for scientific computing," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, 2002.

[7] W. R. Cook and J. Barfield, "Web services versus distributed objects: A case study of performance and interface design," in *Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 419–426. [Online]. Available: http://portal.acm.org/citation.cfm?id=1172963.1173077

[8] T. Takase and K. Tajima, "Efficient web services message exchange by soap bundling framework," *Enterprise Distributed Object Computing Conference, IEEE International*, vol. 0, p. 63, 2007.

[9] I. Kumaran and S. I. Kumaran, *Jini Technology: An Overview*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[10] I. Rozman, M. B. Juric, I. Golob, and M. Hericko, "Qualitative and quantitative analysis and comparison of java distributed architectures," *Softw. Pract. Exper.*, vol. 36, pp. 1543–1562, November 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1181951.1181953

[11] M. B. Juric, I. Rozman, and M. Hericko, "Performance comparison of corba and rmi," *Information and Software Technology*, vol. 42, no. 13, pp. 915 – 933, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0B-40YSBJD-2/2/5db4239a4063c6d3f8e7489b74adf7fa

[12] C. Erik, C. Francisco, M. Greg, and W. Sanjiva, "Web services description language (wsdl) 1.1," 2001. [Online]. Available: http://http://www.w3.org/TR/wsdl

[13] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38 –45, november 2007.

[14] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," in *Proceedings of the 22nd international conference on Software engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 407–416. [Online]. Available: http://doi.acm.org/10.1145/337180.337228

[15] S. Shirasuna, H. Nakada, S. Matsuoka, and S. Sekiguchi, "Evaluating web services based implementations of gridrpc," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, 2002, pp. 237 – 245.

[16] D. Davis and M. Parashar, "Latency performance of soap implementations," in *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, May 2002, p. 407.

[17] S. Chen, B. Yan, J. Zic, R. Liu, and A. Ng, "Evaluation and modeling of web services performance," in *Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 437–444. [Online]. Available: http://portal.acm.org/citation.cfm?id=1172963.1173079

[18] W. Edwards, *Core Jini*. Prentice-Hall, 1999.

[19] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[20] C. Roberto, M. Jean-Jacques, R. Arthur, and W. Sanjiva. (2007, June) Web services description language (wsdl) version 2.0 part 1: Core language. [Online]. Available: http://www.w3.org/TR/wsdl20/, (accessed 10 April 2011)

[21] G. Martin, H. Marc, M. Noah, M. Jean-Jacques, N. Henrik Frystyk, K. Anish, and L. Yves. (2007, April) Soap version 1.2 part 1: Messaging framework (second edition). [Online]. Available: http://www.w3.org/TR/soap12-part1/, (accessed 10 April 2011)

[22] A. Tanenbaum and R. van Renesse, "A critique of the remote procedure call paradigm," in *'Proceedings of the EUTECO 88 Conference' Elsevier Science Publishers B. V. (North-Holland), Vienna, Austria*, 1988, pp. 775–783.

[23] U. Saif and D. Greaves, “Communication primitives for ubiquitous systems or rpc considered harmful,” in *In 21st International Conference on Distributed Computing Systems Workshops (ICDCSW 01*, 2001.

[24] G. Canfora and M. Di Penta, “Service-oriented architectures testing: A survey,” in *Software Engineering*, ser. Lecture Notes in Computer Science, A. De Lucia and F. Ferrucci, Eds. Springer Berlin / Heidelberg, 2009, vol. 5413, pp. 78–105, 10.1007/978-3-540-95888-8_4. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-95888-8_4

[25] ——, “Testing services and service-centric systems: Challenges and opportunities,” *IT Professional*, vol. 8, pp. 10–17, March 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1137241.1137432

[26] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti, “Whitening soa testing,” in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE ’09. New York, NY, USA: ACM, 2009, pp. 161–170. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595721

TUD-SERG-2011-021
ISSN 1872-5392