Web Services Workflow with Result Data Forwarding as Resources *

Donglai Zhang, Paul Coddington, and Andrew Wendelborn
School of Computer Science,
The University of Adelaide, SA 5005, Australia
{donglai.zhang,paul.coddington,andrew.wendelborn}@adelaide.edu.au

December 21, 2010

Abstract

Under a centralized workflow model, distributed data transfer between consecutive services in a Web service workflow can save network bandwidth and avoid the centralized workflow engine being overloaded. The proposed Web Service Data Forwarding (WSDF) framework harnesses the Web Service Resource Framework (WSRF) to meet this requirement. WSDF compliant service in a workflow can forward the result to its successor service for future usage. This result forwarding function is implemented at the WSDF server level and is transparent to functional services. Our prototype system has shown that this framework can save data transfer time and significantly improve the overall performance of some workflows.

keywords: Web service workflow, stateful, data forwarding, WSRF

1 Introduction

In a distributed system, Service Oriented Architecture (SOA) [1] has been regarded as an appropriate framework for distributed components, as they are loosely coupled. Distributed services, e.g., Web services, are provided as resources to clients. To accomplish more complicated tasks, different atomic services can be integrated into a service workflow. For instance, different Web services can be composed to form a Web service workflow.

There are two different types of workflows according to the location of the workflow’s control point. If a workflow has a centralized control point, it is

---

*This is the pre-peer reviewed version of the following article: Donglai Zhang, Paul Coddington and Andrew Wendelborn. Web Services Workflow with Result Data Forwarding as Resources. Future Generation Computer Systems. Available at: http://www.elsevier.com/locate/fgcs
classified as a centralized workflow; otherwise, a decentralized workflow. This kind of workflow normally has a centralized workflow engine, and the workflow engine makes decisions on when and how to invoke the services involved in the workflow. The centralized workflow engine is also the hub to exchange data between different atomic services [2, 3, 4]. Centralized workflows are stable and easy for administration. However, in a centralized architecture, the workflow engine can be the bottleneck of the whole system, especially when there is significant amount of data transferred between services involved in the workflow. This could cause higher network resource consumption and lead to degraded performance of the whole system. For example, often a group of Web services for a specific research purpose are deployed within a local network within an institution. Users from different campuses, cities or even continents, utilize these services by invoking a workflow that is composed of these services to process their data. Under this situation, data transfer between different services can be very inefficient in going through the centralized workflow engine.

There are two types of information flows in a workflow, control flow and data flow [5] (see Figure 1). The service that is being called is the current service. The service that is after the current service and is to use the result of the current service is called a successor service. Figure 1 (a) shows a centralized orchestration [5] model. For each service, the data flow is bidirectional. Input and output data of the Web service shares the same network connection. The same applies to control flow. Different services all talk to the same workflow engine and exchange data via this engine. In Figure 1 (b), the control flow is still centralized, however, the data flow is decentralized. Data is sent from one service to another without going via the workflow engine and used by the successor service which is invoked later by the control flow. The data exchange between different services is in the choreography model [6], under which the workflow engine can avoid of being the bottleneck in the workflow.

Previous research in this area focuses on either extending functional Web services with extra capabilities [7, 8, 9] or reconstructing the workflow [5, 10]. These implementations are limited to the application level and the service server does not provide the underlying mechanism for direct data sharing between atomic services of a workflow.

Within a workflow, atomic services are integrated into a composite service, as shown in Figure 2. The composite service acts as a normal Web service in that both are invoked by a client, process input parameters and return the result to the client without saving it. In Figure 2, there is only one data input flow and one data output flow between the workflow engine and the composite service.
Any intermediate result is within the scope of the composite service. By keeping this result within the composite service, we avoid returning it to the workflow engine, therefore, avoiding the overhead of third party data transfer. We argue that the underlying problem for this model is resource sharing across atomic Web services. To keep the intermediate result of a workflow, we introduce the stateful workflow concept.

In a stateful workflow, each atomic service is stateful. The de facto standard for representing the state of Web services is Web Service Resource Framework (WSRF) [11], which provides a framework such that a compliant Web service is stateful and the state information of a particular Web service instance is a resource. A stateful workflow keeps the state of the intermediate result of Web services and shares them in the composite service. What we want is to have one data result generated from the current service to be transported to and saved on the successor service, which is stateful. However, there is no mechanism provided within the Web service framework to forward a data result from one service to the other as in the push model, or, alternatively, as in the pull model, retrieve data from current service by the successor service. In current practice, if the successor service is a stateful service, the result data can be forwarded from the current service to the successor service by adding a function from the application level. But this will lead to the situation that the data transfer depends on the specific implementation of that service. New mechanisms for result data sharing between stateful services should be built to free Web service workflow developers from being required to implement their own data transfer functions.

We propose the WSDF framework, which is based on the idea of stateful workflow, to allow efficient data sharing between services. In this framework, atomic services involved within a composite service are stateful web services. A WSDF server, built on stateful web service server, hosts atomic services and is responsible for forwarding the result data of the current service to the successor service. The information used by current service server to transfer result data is called resource forwarding information. A resource forwarding information schema is also defined. If a client invokes the current service while embedding this resource forwarding information in the invocation request, the server first retrieves this information from the invocation request; after the functional service is finished, it forwards the result to the successor service as specified by the resource forwarding information. The successor service accepts data sent and stores it as a resource before the invocation of this service. Based on these framework principles, we built a complete prototype system to prove the proposed concepts. Comparing with the normal Web service workflow system, significant
data transfer speed improvement has been achieved by WSDF workflows in long distance data transfers.

The rest of the paper is organized as follows. Section 2 gives a full definition of the WSDF framework; section 3 describes implementation details of a prototype WSDF framework; sections 4 and 5 present the testing environment, test result and analysis; section 6 discusses related work and section 7 concludes our work.

2 Web Service Data Forwarding (WSDF) Framework

We propose the WSDF framework to address the result data sharing issue between services within a centralized workflow by introducing the concept of stateful workflow. Within the WSDF framework, workflow is considered to be stateful because the result data of atomic services is kept within the composite service (as shown in Figure 2). A WSDF server forwards result data from current service to successor service according to the resource forwarding information.

2.1 Stateful Workflow

Web service workflows are composed of atomic Web services. From the client’s point of view, atomic services integrated in a workflow can be viewed as a composite service. But there is difference between them when it comes to the execution cycle and the state of the service. For each atomic service, an invocation cycle only involves a single invocation of a service operation. For a workflow, however, there are multiple invocations within an execution cycle. Each invocation represents a unique stage in the cycle, and the status of the workflow changes after the invocation: current service is executed and new intermediate data is generated. In the case of an atomic service, stateful means the state of a specific service instance can be kept; we define stateful workflow, however, to mean that the intermediate data is preserved between successive services in the same workflow. In a stateful workflow, all atomic services need to be stateful and all intermediate data is directly shared between atomic services (as shown in Figure 2).

In this research work, we use WSRF as the basic specification to build stateful web services, as the WSRF based system has provided the necessary mechanism that can readily provide the functionality to properly support the stateful web service in WSDF framework. On the other hand, the WSDF framework is not limited to the WSRF specification, any other specification(s) or model(s) that provide stateful web service can be used to build WSDF framework.

We also considered the question of whether or not it is possible to work with non-WSRF services. We believe that this could be done, but with considerable difficulty. Essentially, the problem is one of designing and implementing a distributed encoding of both the forwarding information between services and of the data that is to be forwarded, as well as providing a mechanism for storing
the data on the server side and providing a reference to it, which can be passed between the web services and the client workflow engine. We use WSRF, since it provides a high-level mechanism [11] for managing the data on the server side as state information, in a standardised way that is generally available to Web services anywhere on the Internet. In order to provide a mechanism for non-WSRF services, it would be necessary to design an alternative encoding, and in particular to provide support for it within arbitrary distributed Web services. As this would be time-consuming and difficult, we focus our work on the more tractable solution using WSRF.

To maintain state in the workflow, the other key issue is to share result data between different atomic services. The WSRF specification enables Web services to be stateful, but it doesn’t directly address the data sharing issue and it is hard for multiple stateful Web services to share their resources in a standard way. If the successor service is a WSRF service, the data can be forwarded from current service to successor service by adding functions from the application level. However, this approach is ad hoc, and will depend on the implementation of the forwarding function, which can be written in many different ways. It is desirable for Web service developers to have a better standardized programming environment to build their workflows and this is our aim in this work. The WSDF framework implements the data forwarding between services from the server level, i.e., when the control flow invokes the current service with the resource forwarding information, the server, rather than any application service, will take the responsibility to forward the data and this is transparent to the application service.

2.2 Resource Forwarding Information

Within the WSDF framework, the control flow of a workflow is centralized. The workflow engine sends a service invocation request to an atomic service while embedding the resource forwarding information in the request message. This forwarding information includes where and how the generated result can be forwarded to the successor service. To separate this message from parameters used by the current service (application service), the namespace, wsdf, is defined to distinguish the resource forwarding information:

http://cs.adelaide.edu.au/2008/05/wsf

An XML schema for resource forwarding information is also defined within WSDF framework, as shown in Figure 3. The targetNamespace of this element is the wsdf namespace. The element includes serviceURL which is the URL of the successor service. The createOperationURL and setOperationURL are the URLs for creating an Endpoint reference (EPR) of a resource instance and setting a resource with the generated EPR on successor service respectively. An EPR conveys the information for both accessing a Web service endpoint and identifying messages sent to and from Web services of an individual service.
instance [12]. The Endpoint reference is defined by the Web Services Addressing (WS-Addressing) specification[12] which specifies XML elements to identify Web service endpoints and to secure end-to-end identification in messages.

To forward result data to a successor service as a Web service attachment [13], the attachResourceForward element is defined. This is especially useful when the current service generates a big data set as result. If the data is saved as a resource in memory, it could exhaust the memory of the server and degrade the server’s performance. An alternative is to save the data into a temporary file. The property FILE_NAME_AS_RESOURCE is to indicate whether or not the result data is real data or a file reference (i.e. file name) that points to the real data. The property, ATTACHMENT_FORMAT, indicates the format used by the server to forward result data. It is the client’s responsibility to inform current service if a temporary file name has been saved as a resource by setting FILE_NAME_AS_RESOURCE. If this property is set to true, the server will treat the resource as a file name and forward the file content to the successor service. Otherwise the resource content will be directly forwarded.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://cs.adelaide.edu.au/2008/05/wsdf"
>
  <xs:element name="wsdf_information">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="serviceURL" type="xs:string"/>
        <xs:element name="createOperationURL" type="xs:string"/>
        <xs:element name="setOperationURL" type="xs:string"/>
        <xs:element name="attachResourceForward" type="xs:string">
          <xs:attribute name="FILE_NAME_AS_RESOURCE" type="xs:string"/>
          <xs:attribute name="ATTACHMENT_FORMAT" type="xs:string"/>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 3: XML schema for resource forwarding information

### 2.3 Successor Service

In the same way as the current service, the successor service should also be a stateful web service, in this case, a WSRF service. By default, it should provide setResourceProperty operation to set the content of a resource [11]. It should also provide a create operation to create a resource instance of the service and
return an EPR that points to the created instance. The Web service client can use the default resource operation \textit{setResourceProperty} to set the content of a property, however, to support data sent via Web service attachment, a service needs to define separate \textit{setAttachAsResource} operation, as it is not provided by WSRF framework.

2.4 WSDF Architecture

The WSDF framework is built on WSRF. It includes Web Service engine, WSDF client and WSDF services. WSDF inherits the resource management mechanism from WSRF. Every WSDF service is first-of-all a WSRF service. A functional service within a WSDF framework consumes a resource and generates a response to a client request. It also provides basic services such as creating the resource reference and setting the resource.

- **Web service engine in WSDF framework.** The primary encoding specification for a WSRF Web service is SOAP [14]. A WSDF SOAP engine is an extended WSRF SOAP engine that understands the resource forwarding information. When the WSDF engine receives request message from a client, it reads the resource forwarding information from the request message and saves it temporarily. Then it invokes the functional service. When the service finishes, the WSDF engine checks to see if the result received needs to be forwarded to any successor service. If so, it will create the resource instance(s) and set content of the instance on the successor service(s). Data forwarding between different services within WSDF uses Web service invocation to keep the whole system in a unified model. The URLs that are used to compose these invocations should be the ones that the Web service server obtained from the request message. If the resource forwarding information contains multiple successor services, the result will be forwarded to each of them respectively. The WSDF SOAP engine also supports Web service with attachment [13, 1] to transfer large data efficiently.

- **WSDF application services.** WSRF implementation provides resource management mechanisms. The WSDF application service can utilize standard WSRF resource management functions such as \textit{setResourceProperty}. It needs to implement the \textit{create} interface for creating a resource instance and the \textit{setAttachAsResource} interface to support Web service with attachment.

- **WSDF client.** The WSDF service client composes a WSDF request based on a basic WSRF request. The difference is that if a WSDF service client needs the result to be forwarded to the successor service, it should know the URLs to create and set resource on the latter service. These URLs are used to compose the resource forwarding message element as part of the SOAP envelope of the service invocation request.

3 WSDF Implementation

The WSDF implementation includes three main parts: WSDF service server, WSDF service and client. Our implementation also supports Web service with
attachment to improve large\textsuperscript{1} binary data transfer speed for computational Web services.

3.1 WSDF Server

As WSDF is built on WSRF, we chose WS-core \textsuperscript{[15]} to build the WSDF service server. WS-core is the Web service server of the Globus \textsuperscript{[16]} software toolkit, which implements WSRF and has been widely used by the research community. By using the WS-core source code obtained from WS-core project\textsuperscript{2}, we rebuilt the SOAP engine for WSDF service and named it as WSDF-axis. Beside the functions that a WSRF server has, the WSDF Web service server also provides the ability to interpret resource forwarding information embedded in the invocation request message and forward result data after it is generated by a back-end application. After data forwarding, the EPR pointing to the resource will be returned to the current service and finally to the client for the next invocation. If the client does not specify any resource forwarding information, the WSDF server will return the result directly back to the client. Figure 4 illustrates how the WSDF-axis engine works when a functional service is invoked on the server side, according to the following steps:

Step 1: input SOAP message including resource forwarding information, resource information and service information is sent from the client to the WSDF Engine.

Step 3: create the context of current service instance.

Step 4: resource forwarding information is retrieved from the SOAP message envelope and saved temporarily for future usage.

Step 5: invokes the back-end application.

Step 6, 7 and 8: by interacting with the WSRF resource context created in step 3, the back-end application gets the resource property value of the current service instance, carries out the functional processing and resets the resource property in the context. The result is then returned to the SOAP engine.

Step 9: SOAP engine carries out the resource forwarding task. It first checks the resource forwarding information. If there is information about one or more

\textsuperscript{1}But less than 2GB. This is decided by the maximum value of the int type in Java language.

\textsuperscript{2}WS-core version of Axis is a “enhanced” version that supports WSRF functions.

Figure 4: SOAP Engine Functional Service Invocation In WSDF Framework
successor services, the result will be forwarded to each of them respectively and saved as a resource. Two steps are involved within the creation and setting of the resource. First, the SOAP engine sends a request to successor service to create a new resource instance on that service and receives an EPR to the instance. Second, the engine sends a second request to the successor service to set the result as the resource referred by the previous created EPR. The SOAP envelope within the request is composed by the SOAP engine using information that is contained within the resource forwarding information. No Web service stub is needed.

Step 10: if the result from current service has been forwarded to successor service, return EPR to the client, otherwise, return the result to the client.

3.2 WSDF Client

Under WSDF, a client sends an invocation request to the server by composing the SOAP envelope directly. Developers need to compose these SOAP envelopes explicitly in the client applications. As we have extended the standard WSDL to support the forward function in WSDF [17], in the future, stubs could be built automatically from WSDL. The SOAP envelope should also contain resource forwarding information.

3.3 WSDF service

A WSDF service is first of all a stateful web service. It is different from a normal web service, in which only uses the parameters passed in by the client and processes these values directly. For stateful web services, there are two different types of parameters, one is the normal parameter, directly sent from the client as in the normal web service; in addition, there is extra resource information that is stored in the request, which represents a different set of parameters that has been saved on the current service side and is to be used by this service. These two types of parameters are represented and stored differently: this is best represented by using stateful web services.

A WSDF service should implement two operations in addition to its computational service. First, a create operation, for the client of a WSDF service to create a resource instance on that service. By invoking the create operation, a resource instance is created for that service and an EPR should be returned to the client. Second, an operation for the client to set the resource state, such as the WSRF operation, setResourceProperty [11]. Further, to enable the WSDF service to support Web service with attachment for binary data transfer between different services, a separate resource property operation setAttachAsResource should be implemented. When the operation is invoked, it gets the data from the attachment of the invocation message and sets as the resource. The setAttachAsResource operation will be implemented in the way that it saves the attached data in a temporary file and location of the file is saved as the resource property.
3.4 Building Workflow With WSDF framework

To demonstrate the data forwarding mechanism within a WSDF workflow, and to test the performance of the system, we built both a WSDF service named RGB\(^3\) and its client. The service provides create, setAttachAsResource and convert operations. Convert is a functional operation that takes the content of a .bmp image file as input and changes the color of its pixels: red to green, green to blue and blue to red; the create operation is used to create an EPR for a service instance on this service and the reference is sent back to the client; setAttachAsResource operation is used to set the attachment of the request as a resource which is to be processed by the convert operation. If there is forwarding information within the convert request, the server forwards the result to the destination(s); otherwise, the result data is returned back to the client. In our implementation, three RGB services are deployed on different WSDF servers. A workflow which is composed of multiple segments of service invocations in a pipeline style, similar to the one shown in Figure 2, is built to carry out the computation. After an execution cycle of these services, the final image is sent back to the client. The returned image should be identical to the original file.

4 Testing

We use the WSDF engine prototype and the workflow we built in section 3.4 to demonstrate the advantages of the WSDF framework.

**Services and data size.** Different services will take different processing time. To keep time constant, we use the same RGB Web service each time (see section 3.4). In a WSDF workflow, the more services involved, the more data transfer takes place within the composite service side and the more improvement we expect to see from the WSDF framework compared to standard web service without data forwarding. We test our WSDF framework with different workflows that are composed of 3, 6, 9 and 12 services respectively. We also provide different data sizes to be processed by the workflow. Data size in our experiments ranges from 100KB to about 2GB (as we use Java based WSDF-axis engine to carry out the experiments), including 100KB, 500KB, 1MB, 5MB, 10MB, 50MB, 100MB, 500MB, 1GB and 2040MB.

**Emulation environment for testing.** We use WANem \([18]\) to build a WAN emulating environment to carry out experiments \([17]\). The emulator is configured as a gateway between client and servers, as well as gateways between servers. We set different latency and bandwidth on these gateways to emulate inter-continental, and intra-continental network respectively. Similarly to our previous experiments \([19]\), we set the inter-continental network latency to 110ms and set the bandwidth to 20Mbits/sec. The intra-continental network latency is set to 10ms and the bandwidth is set to 40Mbits/sec. For local network, we did not use the emulator but ran the experiment directly on a LAN. The bandwidth of the LAN is 100Mbits/sec and the latency is about 0.2 milli-seconds.

\(^3\)The name is an acronym for red, green and blue
**Basic Service Time (BST) Consumption** One purpose of this test is to compare the performance difference between WSDF workflow with normal Web services workflow. To eliminate the variation in execution time of the computational service, which differs according to the service selected, but is the same in each approach, we introduce the Basic Service Time (BST). The BST refers to the time taken by the service to do the computational work, which is algorithm specific, and excluding any input/output operations. Comparisons between WSDF workflow and normal Web service workflow are mostly based on their execution time excluding the BST.

**Hardware.** We use five Linux boxes for the test and all of them have two Intel(R) Pentium(R) 4 CPU 2.80GHz with 1GB memory. Three of them are used as servers and one of them is used as client or workflow engine. We use the last one as the gateway to emulate remote network connection.

5 **WSDF Performance Analysis**

The WSDF framework has introduced stateful workflow, a new concept, for data sharing between workflow services. Within WSDF framework, Web services share intermediate data directly in a distributed way. Therefore, data transfer between WSDF services is more efficient when compared to normal Web services that do not do data forwarding.

In section 5.1, we compare the data transfer time between WSDF and Web Service framework by deriving analytical formulas for network data transfer. These formulas illustrate that how the WSDF workflow system saves data transfer time over normal Web service workflow system. Then, in section 5.2, we give experimental results and analysis.

5.1 **Data Transfer Time Comparison**

WSDF framework brings time saving advantage over Web service workflow by reducing the amount of data transfer between the client and the servers. We compare the total time used for network data transfer (referred to as transfer time) in both Web service workflows and WSDF workflows and analyse the advantage of WSDF framework.

\[
T: \text{Overall transfer time for normal Web service workflow.} \\
T': \text{Overall transfer time for WSDF workflow.} \\
DI_i, DO_i: \text{Input and output data of the } i\text{th service respectively.} \\
BW_{C,i}: \text{Bandwidth of the network connection between client and the } i\text{th service.} \\
BW_{i,i+1}: \text{Bandwidth of the network connection between the } i\text{th and } i+1\text{th service.}
\]

Suppose a workflow has \( n \) services and a successor service always uses the previous service’s result data. In equation (1), \( T \) is the sum of transfer time used for input data \( DI_i \) and output data \( DO_i \) to transfer between the client
and the server via network connection for each service. The bandwidth of the $i$th connection is $BW_{C,i}$. \footnote{For simplicity reasons, we ignore other factors that might affect the transfer time, such as latency. Here, we suppose these factors only have a relatively small affect to the overall transfer time.}

$$T = \sum_{i=1}^{n} (DI_i / BW_{C,i} + DO_i / BW_{C,i}) \quad (1)$$

The overall transfer time in WSDF framework is composed of three parts: first, transfer time for input data from the client to the first service; second, total transfer time for services from the first to the $n$-1th to transfer output data from the current service to its successor service; last, the output data transfer time from the $n$th service to the client.

$$T' = (DI_1 / BW_{C,1}) + \sum_{i=1}^{n-1} (DO_i / BW_{i,i+1}) + (DO_n / BW_{C,n}) \quad (2)$$

Comparing the difference between $T'$ and $T$, when the output data from the current service is the input data of the successor service (i.e. $DO_i$ equals $DI_{i+1}$).

$$T - T' = \sum_{i=1}^{n-1} ((DO_i / BW_{C,i}) + (DO_i / BW_{C,i+1}) - (DO_i / BW_{i,i+1})) \quad (3)$$

Equation (3) illustrates the difference between the two frameworks. For both WSDF and normal Web service workflows, from the first to the $n$-1th service, each service transfers the same intermediate data $DO_i$ from one service to the next, but via different network paths. $(DO_i / BW_{C,i}) + (DO_i / BW_{C,i+1})$ represents the transfer time under Web service framework: the intermediate data $DO_i$ is first sent from the $i$th service to the client via a network with bandwidth $BW_{C,i}$, and then sent from the client to the $i+1$th service via a network with bandwidth $BW_{C,i+1}$, i.e., the data transfer goes via a third party (the workflow engine); within a WSDF framework, the transfer time is $(DO_i / BW_{i,i+1})$, the intermediate data $DO_i$ is sent from the $i$th service to the $i+1$th service directly via a network connection with bandwidth $BW_{i,i+1}$. As mentioned in section 1, if the servers in a workflow are located in a LAN and connected by high bandwidth network, then the bandwidth $(BW_{i,i+1})$ is much larger than the ones between the servers and client $(BW_{C,i})$.

We define the percentage of time saving from WSDF to be:

$$P = \frac{T - T'}{T} \times 100 \quad (4)$$

To further simplify the formulas, we make the following assumptions: first, all the input and output data in a workflow have the same size, represented by
Based on these assumptions, the percentage of transfer time saved by the WSDF framework over Web service workflow is as following:

\[ P = \frac{D \times \sum_{i=1}^{n-1} (2/BW_{C,S} - 1/BW_{S,S})}{D \times \sum_{i=1}^{n} (2/BW_{C,S})} \times 100 \]  

(5)

In equation (5), if \( BW_{S,S} \) is much larger than \( BW_{C,S} \), then the maximum transfer time saving ratio of WSDF framework will be nearly \( n - 1/n \).

5.2 Experimental Results and Analysis

We have carried out comprehensive experiments to compare WSDF workflow performance with Web service workflows. According to our experiments, the performance of WSDF workflow shows great advantages on time saving for data transfer in both local network scale and Internet scale (for more detailed results, please refer to the technical report [17]).

In a LAN environment (Figure 5), WSDF workflows have advantages over Web service workflows in most cases; and in remote environments (Figure 6, 7), WSDF workflows show clear advantages over Web service workflows. In a LAN, the WSDF framework saves time from data transfer, and at the same time introduces extra time for resource creation and management (see section 3.1). In a LAN environment, this extra time could be higher than the time saved from data transfer by WSDF workflow (e.g. 100KB files with 3 services).

In a remote environment, network connection conditions between services are much better than the ones between the workflow engine and the services and the time saved from data transfer is much larger compared to the extra time introduced by WSDF. The WSDF workflows save a significant amount of time.

Performance differences also exist between different data sizes. The overhead for resource instance creation and management of a small data set are much
more significant compared with a large data set. With the same number of services, a WSDF workflow can save more time when it is processing larger data. For example, in the intra-continental environment, a WSDF workflow with 12 services can save about 33% transfer time of web service workflow with the input of a 100KB file, 60% with input file of 1MB and 70% for input file of 10MB. The increase of performance becomes steady when the input size increases further, as the resource creation time is relatively small compared to the whole processing time.

Comparing the time saving for local, intra-continental and inter-continental services, inter-continental WSDF services show the most improvement and local network WSDF services, the least.

As shown in Figure 8, in a LAN environment, the network time (i.e. total time minus BST time) saving (as defined in equation 4) for RGB workflow with six services for a 100MB file is about 34%; in an intra-continental environment, the time saving is about 64%; in an inter-continental environment, it is about 68%. If we put the related experimental data into equation (5), the estimated
transfer time saving for local service is 42%, for intra-continental service is 67% and for inter-continental service is 75%, which is reasonably close to our experimental results, as we are not considering the overhead brought by WSDF framework. The workflows with 3, 9 or 12 services present a similar trend, where the slower the connections between client and the service providers are, the bigger improvement could be achieved by WSDF framework. This meets our expectation: with WSDF workflows, as most data transferred does not need to be sent back to the client via a low bandwidth network, it takes less time for the workflow to complete.

Figure 9 shows that the more services involved within a workflow, the higher percentage of time is saved from WSDF framework. From Figure 2, we can see that only the first input data and the last output data transfers are between the client and the stateful composite service. When there are more services involved in a composite service, more data exchanges are within the composite service, which improves the overall performance of the workflow. Also, in equation (5), the more services involved within a WSDF workflow, the higher percentage of time will be saved by the WSDF workflow, therefore, higher performance will be achieved.

Figure 8: Time Saving Comparison

Figure 9: Performance Improvement with More Services
6 Related Work

According to the usage of workflows, they can be divided into business workflow and scientific workflow. Scientific workflow by its nature is more data flow oriented. Business workflow is more control flow oriented. However, within a centralized workflow model, data sharing between different Web services is a common research area for both types.

In [5, 20], the relationship of control flow and data flow within a workflow system is illustrated. Within a centralized control-flow, distributed data-flow (1CnD) model, different autonomous services, created by wrapping software application with a mediator that supports ASAP (Autonomous Service Access Protocol), establish data-flow directly between them. The centralized controller coordinates the control flow between different autonomous services. This suggested service model is a general model that can apply to any application which has been wrapped with a mediator. However, as we focus on Web service workflow and Web service is the application interface, we prefer to take advantage of Web service architecture to separate the data flow and control flow in its context. We believe it is not a good choice to encumber Web services with an additional mediator.

In [21, 10], the authors described algorithms to convert a workflow into smaller units, where each unit works as a centralized workflow system. These units are run on servers at different locations with direct communication between them. The expected benefit from this approach is to avoid the central point of the workflow orchestration becoming a bottleneck and significantly improve the overall throughput. This approach is especially good for data driven workflow. But this algorithm also makes the whole system more complex and potentially less stable.

The proxy model is suggested in [7] and [9]. In [7], a hybrid architecture is built. A proxy is a piece of middleware closely coupled to a functional service as a gateway. It has three major functions: delegating the invocation of the functional service; managing input/output data storage and sending the result data between workflow components. A set of Application Programming Interfaces (APIs) are given to interact with the proxy service. This research work has pointed out some research issues in the data sharing problem between Web services in a centralized workflow, such as result data storage, forward and retrieving. The APIs given in the paper are Java language specific and do not fit Web services written in the sense of programming language neutrality. A related but different model is introduced in [9], in this paper, the trigger also stands as a proxy to delegate the Web service’s invocation and result data forwarding to its consumer. While sitting closely to the Web service, the trigger acts as a buffer of input parameters and waits until all parameters arrive, then it will trigger the corresponding service. After getting the output of the service, the results are sent only to where they are needed by the trigger. In this paper, no standard description is provided for the trigger service and we expect different implementations will not be compatible.

The general drawback of the proxy models is that they address the data
sharing problem from an application level. Either the trigger or the proxy service is defined as a normal function service, so the programmer needs to maintain these services for themselves. Our model focuses on data sharing between services at server level and hides the details of data sharing from users.

7 Conclusion and Future Work

7.1 Conclusion

We introduce the WSDF framework to share the intermediate data between successive services in a service oriented workflow. Though it is based on Web services with WSRF specification, this framework can be extended to any other services with their own state specification. The main contributions of our work are as follows.

We introduce the state concept into workflow, especially, Web service workflows. The benefit of a stateful workflow is that it can keep the state of the intermediate data between different services. Result data shared by other services does not need to be sent back to the centralized workflow engine, but needs only to be transferred between different atomic services, stored and retrieved as a resource. Under this framework, distributed data flow between atomic services is possible.

As it is a common scenario that result from a current service is used by a successor service in a Web service workflow, we suggest data sharing between different services in a workflow should be implemented at server level rather than application level. A WSDF server retrieves necessary resource forwarding information from the invocation request and carries out the data forwarding on behalf of the client transparently. We built a WSDF server, an extension of the WSRF server, to prove this concept.

We define wsdf namespace and resource forwarding information schema to describe the information used for data forwarding between different WSDF services.

For our WSDF implementation, we also observe the following results from experimental tests: WSDF workflows perform better than Web service workflows in both local area network (LAN) and wide area network (WAN). WSDF workflow with more services involved will save more on network transfer time, as most data transfer occurs between services within the composite service. Compared to normal Web service workflows, WSDF workflows have better performance when the client (workflow engine) is remotely located from the services. The WSDF model also suits workflows that involve significant data transfer. Workflows that process bigger files save more network time over workflows that process smaller files.
7.2 Future Work

Currently, the WSDF framework only uses *push* model for data sharing between different atomic services in WSDF framework. We believe that a *pull* model can bring more flexibility. For example, if there are multiple successor service candidates, when the current service is invoked and the workflow engine has not decided which candidate is going to be used, the current service can save the result data of the service as a local resource and give back a resource reference to the client. After the client selects the right successor service provider, it can invoke the operation on the successor service by passing the resource reference on the current service.

With distributed data flow between different services, WSDF workflows avoid possible performance bottleneck. If a researcher runs multiple normal workflows simultaneously on his/her computer, the network can be overloaded if a large amount of data is exchanged via the centralized workflow engine. We did not carry out related experiments, but these experiments could be implemented in the next stage of our research work. We will also carry out experiments with this framework in the cloud environment to figure out how this framework can utilize the cloud infrastructure for better performance.

Currently we manually compose client side invocation. In the future, we will work on tools for client side stub generation, which will be more convenient for WSDF workflow programming.

References


