Web Services Workflow with Result Data Forwarding as Resources

Donglai Zhang\(^1,\)∗, Paul Coddington\(^1,\)∗, Andrew Wendelborn\(^1,\)∗

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 proved that this framework can save data transfer time and improve the overall performance of the workflow significantly.

Keywords: Web Service workflow, stateful, data forwarding, WSRF

1. Introduction
In a distributed environment, 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 a resource to clients. To accomplish a more complicated task, 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 workflow according to the location of the workflow’s control point. If a workflow has a centralized control point, it

\(^*\)Correspondent author

Email addresses: donglai.zhang@adelaide.edu.au (Donglai Zhang),
paul.coddington@adelaide.edu.au (Paul Coddington),
andrew.wendelborn@adelaide.edu.au (Andrew Wendelborn)

\(^1\)School of Computer Science, The University of Adelaide
is classified as a centralized workflow. This kind of workflow normally has a centralized workflow engine, and the workflow engine makes decision 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, 5]. 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 often causes higher network resource consumption and leads to the degraded performance of the whole system. For example, often a group of Web services for a specific research purpose are deployed on servers within a company or a research institution within a local network. Users from a different campus, city or even continent, who have their own data to be processed, utilize these services by invoking a workflow that is composed from these services. Under this situation, data transfer between different services can be very inefficient in going through the centralized workflow engine.

To reduce the overhead of data transfer within a centralized workflow, we need to analyze the types of information flows within a workflow. As in [6], there are two types of information flows, control flow and data flow. Figure 1 illustrates how these two types of flows work within a Web service workflow.

Within a workflow, the service that is being called is the current service and the service that is after the current service in the workflow and is going to use the result of the current service is called successor service. Figure 1 (a) shows that with each service, the data flow is bidirectional. Data input and output of the Web service shares the same channel with different direction. The same applies to control flow. Different services all talk to the same workflow engine and exchange data via this engine. Both control and data flow is under the centralized orchestration model. If the Web service workflow includes big data transfer, it could cause big performance degradation of the workflow. 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 next service which is invoked later by the control flow. The data exchange between different services is in a choreography model. With separated control flow and data flow, the advantage is significant: the workflow engine will not be a bottleneck for intermediate data exchanging between different services, particularly, when there are multiple instances of the workflow are executed concurrently on the
same node.

Previous research in this area focuses on either extending functional Web services with extra capabilities [7, 8, 9] or reconstructing the workflow [6, 10]. These implementations are limited to the application level and not addressed from the server level. We argue the real issue within this problem is to transparently share the intermediate data between atomic services of the 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 result to the client without saving it. If each atomic service is stateless, the composite service is also stateless. On the other hand, if atomic services are stateful, then the composite service is also stateful, from the point of view that the result data of an atomic service can be kept as a resource on the composite service side. 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 composite service scope. 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.

There are different ways to keep an atomic Web service stateful. The
de facto standard for representing the state of Web services is Web Service Resource Framework (WSRF) [11]. WSRF has provided a framework such that a compliant Web service is stateful and the state information of a particular Web service instance is a resource. However, in most cases, the resource managed by WSRF is used only locally within the same Web service instance[12].

The resource sharing between stateful atomic Web services involved in the same composite service has not been researched. Theoretically, one data result generated from current service can be transported to and saved on the successor service if the latter service is also stateful. However, there is no mechanism provided within the current 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 our current work, we implemented the resource data sharing between services using the push model, as it is straightforward and easy to implement. To introduce state into workflow with standard, efficient and stable implementation, a result data forwarding/retrieving mechanism has to be provided. Therefore, changes have to be made from both Web service server and atomic services aspects. To achieve this, firstly, we believe it is vital to keep the state on each single service for result data sharing between services. Secondly, even if state can be kept on a single Web service, it is still hard for a workflow which involves multiple stateful Web services to share
their resource. We need a dedicated mechanism to meet this requirement. Currently, 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 transferring 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 implementing their own functions.

We introduce Web Service Data Forwarding (WSDF) framework, which is built on WSRF, to address these requirements. The WSDF framework is based mainly on two principles: first, in a workflow, result from current service should be transferred to, stored and processed as resource by the successor service; second, as resource sharing between different services in a workflow is a more and more common scenario, the implementation of resource transfer between services should be defined and implemented on a server level, so it can be reused by the whole community with various implementations.

Within the WSDF framework, atomic services involved within a composite service are WSRF services. A WSDF server, built on WSRF server, hosts atomic service and is responsible to forward 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. In the push model, the information includes the URI of the successor service and the URI to create and set resource on that service. A resource forwarding information schema is also defined. If a client invokes the current service while embedding resource forwarding information in the invocation request, the server first retrieves the resource forwarding information from the invocation request. After the functional service is finished, the WSDF server will forward the result to the successor service as the resource forwarding information described. The successor service accepts data sent and stores it as a resource before the invocation of this service.

Current Web Service Description Language (WSDL)[13] describes the service information of a Web service. It is based on the client-server model. To forward result data, which involves a third party, we define extra WSDL syntax for WSDF services. Particularly, the new syntax adds forward to the WSDL operation element of a Web service to represent the third party transfer function of that service. To implement the forward function, a WSDF server carries out the result forwarding task after the functional service is
executed. Necessary information for data forwarding (e.g. URL of the destination) is embedded and sent by client within the service invocation request. A WSDF server retrieves this message and saves it temporarily before the functional service is invoked.

The successor service also needs to implement extra interfaces to create a resource instance for the service, and set the resource content to be used by this service.

Based on these framework principles, we built our implementation. We choose WSCORE (Java) [14][15] as the WSRF implementation on which to build the WSDF server, as it is open source and widely used. We have also built a complete testing system to prove the proposed concepts. Details of building this system have been illustrated and performance achieved by this prototype system has been compared with the normal Web service workflow system with a big range of data size. Significant data transfer improvement has been achieved with long distance data transfers.

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. We introduce the concept of state into workflow. Within WSDF framework, workflow is considered to be stateful as the result data of the atomic services is kept at the composite service side (as shown in Figure 2). The state of the workflow changes after each invocation of its atomic service. To enable the workflow to have state, each atomic service should be a stateful service. They exchange data in a standard way and save the result data as resource. The WSDF framework is built on WSRF. The workflow engine is still the control point of the whole workflow. Every invocation message it sends to an atomic service could include resource forwarding information to where the current service should forward the result data. The resource forwarding information within an invocation request is distinguished from functional parameters by using wsdf namespace. An XML schema has also been defined for the resource forwarding message. Successor service within the workflow should provide service interface to create resource instance for current service to forward the result data. Standard WSRF interface or user defined service interface should be used to set the content of resources on the successor service.
2.1. Stateful Workflow

Web service workflows are composed by atomic Web services. From the client’s point of view, atomic services integrated in a workflow can be viewed as a new composite service. But there is difference between an atomic service and a composite service when it comes to the execution cycle and the state of the service. For each atomic service, an invocation cycle only involves a single operation invocation of that service. 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. With an atomic Web service, stateful means the service can keep the state of a specific service instance; on the other hand, by defining stateful workflow, we mean that the intermediate data is preserved between successive services in the same workflow. To achieve that, all atomic services within the composite service need to be stateful to keep state information on the composite service side. By defining stateful workflow, intermediate data is directly shared between atomic services (as shown in Figure (2)) and does not need to be sent back to the client side. Therefore, data transferring via the workflow engine can be avoided.

Within the WSDF framework, every atomic service is stateful as each service is a WSRF service. To keep the state of the workflow, the other key issue is to share result data between different atomic services. The current WSRF specification enables Web services to be stateful, but it can’t solve the data sharing issue. Although the WSRF framework is not limited to a single Web service, it is hard for multiple stateful Web services in a workflow to share their resource in a standard way. If the successor service is a WSRF service, the resource can be forwarded from current service to successor service by adding functions from the application level to forward the result data. However, this approach will totally depend on the implementation of the current service. The interface signature of this forwarding application can be various and Web services developers deserve a better programming environment to build their workflow. WSDF framework suggests that the data forwarding between different services should be implemented from the server level, i.e., when the control flow invokes the current service it should also inform this service to where the result data should be forwarded. Then the server, rather than any application service, will take the responsibility to forward the data and this is transparent to the application service.

As the WSDF framework asks the current service to forward result data
generated from the current service to its successor service, the data forwarding is directed, i.e. only the current service can invoke and pass parameters to the successor service. The result data forwarding service provided by the successor service is an atomic service and will not invoke any other services. So there is no opportunity for these services to form a cycle between them. Therefore, no no-determinism will be introduced by the WSDF model as discussed in [16, 17].

2.2. WSDL Syntax For WSDF

Web Service Description Language (WSDL) [13] describes the Web service interface in a standardized way. WSDF enables result data forwarding functionality from Web service server level and defines extra description syntax to describe it.

Web service is based on client-server model and WSDL is used to illustrate the relationships between the client and the server. Within WSDL, a port type defines both abstract operations and messages involved within it. There are four transmission primitives that WSDL supports—one-way, request-response, solicit-response and notification [13]. All of them are used to support communications between two participants, i.e., between the client and the server. Figure 3 shows the WSDL grammar of a request-response operation.

However, within a Web service workflow environment, data needs to be transferred not only between the client and the current server, but also, to other server on which the successor service is located, i.e. the third party. Limited by the transmission primitives, WSDL does not support third party data transfer. As a more and more common scenario, we believe third party data transfer should be supported by WSDL. With WSDF framework, a WSDF server supports result data forwarding to the third party, to meet the requirement of sharing data between different services within a workflow. We also suggest <wsdl:forward> as a new element of <wsdl:operation> to represent the action of result data forwarding to a successor service. The suggested WSDL is shown in Figure 4 and the forward element illustrates that the service supports third party resource forwarding between Web services. Note, in Figure 4, the value of <wsdl:output> is different from the former <wsdl:output> element in Figure 3, the former element is the output from the application service of the web service, the later output element is endpoint reference(s) that returned from the successor service(s).
Figure 3: WSDL Grammar For a Request-Response Operation [13]

Figure 4: Suggested WSDL Grammar with Third-party Data Forwarding Element

9
2.3. Resource Forwarding Information

Within WSDF framework, the control flow of a workflow is still centralized. The workflow engine sends service invocation request to an atomic service while embedding resource forwarding information in the request message.

To address the requirement of forwarding the resource, a client of WSDF needs to send certain resource forwarding information within the invocation message, to the atomic service. This information includes where and how the generated result can be forwarded to the successor service. It is embedded into the SOAP envelope which contains invocation message to the current service. To separate this message from parameters used by current service (application service), a namespace called wsdf is defined to distinguish the resource forwarding information. The XML schema for resource forwarding information is also defined within WSDF framework. SOAP message elements within the wsdf namespace and compliant to resource forwarding information schema are only interpreted and used by WSDF server as http://cs.adelaide.edu.au/2008/05/wsdf.

In Figure 5, the XML schema for resource forwarding information is defined. 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. The Endpoint Reference is defined by Web Services Addressing (WS-Addressing) specification[18]. WS-Addressing specification defines XML elements to identify Web service endpoints and to secure end-to-end identification in messages[18]. Endpoint reference is defined by this specification to convey the information for both accessing a Web service endpoint and identifying messages sent to and from Web services of an individual service instance. To forward result data to a successor service as a Web service attachment [19, 20], the attachResourceForward element is defined. The ATTACHMENT_FORMAT property indicates the format used by the server to forward result data. The other property FILE_NAME_AS_RESOURCE is to indicate if the result data is real data or a file reference (i.e. file name) that points to the real data. This is useful when the current service generates a big data set as result. The current service can save the data as a resource in the memory, but as this could exhaust the memory of the server, which is not efficient. The alternative way is that the current service creates a temporary file and saves the
result data set into this temporary file. The file name, rather than the real data, will be saved as a resource. However, the WSDF server will not be able to know if the resource is a file name or the real content, as it will not know the implementation details of a specific application service. On the contrary, the client knows how current service handles the result data and needs to inform the server of the current service if a temporary file name is saved as a resource by setting FILE_NAME_AS_RESOURCE. With this information, the server can read out data from the file has its name saved as a resource, and forward the real content to the successor service. If a service with two different implementation versions, and they save the result as resource in different ways (i.e. set the result as a resource vs. save the result in a temporary file and set the file name as a resource), the client can decide which one to use and let the server know how to handle them respectively by setting the FILE_NAME_AS_RESOURCE in the invocation request.

2.4. Successor Service

There are two services involved in each data exchange between the atomic services: current service and successor service. Same as the current service, the successor service should also be 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 setResourceProperty to set the content of a property, however, to support data sent via Web service attachment, a service needs to define separate setAttachAsResource operation, as it is not provided by WSRF framework.

2.5. WSDF Architecture

The WSDF framework is built on WSRF. A Web Service engine that supports WSRF has been extended to support result resource forwarding. This WSDF Web service engine interprets the Web service data forwarding message which conforms to the resource forwarding schema and stores this information temporarily. After the functional service has been completed, the Web service engine will forward the generated result data to the successor service. If there is no successor service, the result data will be sent back to
The server knows if the current service is the last service according to the resource forwarding information it received.
and gets a result, it invokes the successor service to create a service instance reference, and sets the result as a resource of that service instance.

2. WSRF defines resource life time management interface. WS-ResourceLifetime (WSRF-RL) specification [11] is part of the WSRF specification. This specification defines operations such as Destroy (destroy a resource) and SetTerminationTime (set termination time of a resource) to manage the life time of resource properties. Any implementation of WSRF (e.g. WS-core) also provides an interface for the user to manage the resource life time. In WS-core[15], wsrf-destroy and wsrf-set-termination-time are command line interface for users to manage the life time of resource. Within WSDF framework, a service and its client can utilize this mechanism to automatically manage the life time of a resource (e.g. the client sets a termination time for the result data to be saved on the successor service as a resource).

3. Web service Engine in WSDF framework. The primary encoding specification for Web service is SOAP [21]. A Web service engine is also called a SOAP engine. The implementations of SOAP engine include Axis[22, 23], XFire[24] and gSOAP [25]. A SOAP engine reads the input message and processes it. Within the WSDF framework, a SOAP engine needs to be extended to understand the resource forwarding information that implements the corresponding schema. When the SOAP engine receives request message from a client, before invoking the functional service, the engine should read the resource forwarding information from the request message and save it temporarily for later usage. Then it invokes the functional service. When the functional service finishes, the SOAP engine checks to see if the result received needs to be forwarded to the successor service. If so, it will create the resource instance and set content of the instance on the successor service. 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. For large data transfer, Web service with attachment [19, 1] can be used to achieve better performance. This attachment information can also be embedded and retrieved from the Web service resource forwarding
information. The client or the workflow makes a decision according to the ability of the successor service if the result data should be sent via attachment or not, and informs the current service in service request. For example, if a successor service wants result data to be sent to it in the attachment part of a Web service invocation, then the client needs to specify that in the resource forwarding information (by setting the wsdf:attachmentResourceForward element in resource forwarding information). So the server can forward the result data as a Web service attachment to the successor service.

4. WSDF Application Services. As we have explained, WSDF is built on WSRF and the WSRF implementation provides resource management mechanisms. The WSDF application service, as with other WSRF services, can use standard WSRF resource management functions such as setResourceProperty and getResourceProperty. As shown in following section, application services can also provide extra implementations to meet their own requirements, such as supporting of Web service with attachment.

5. WSDF Client. The WSDF service client composes 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 the resource. These URLs are used to compose message element. This message element will be put into the SOAP envelope as the parameter elements used for the Web service. If the functional service supports Web service attachment, the client can also specify the attachResourceForward part in the resource forward information element.

2.6. Workflow Within WSDF Framework

Figure 6 shows an example where there are three services within the workflow and illustrates the steps for service execution and data forwarding. The following steps are mostly based on standard WSRF operations, apart from the ones involving communication between different services, which are specific to WSDF.

- **Step 1:** From the workflow engine, client sends request to create an endpoint reference of a new service instance.
- **Step 2:** Invocation (control flow) sent from client to service.
Figure 6: Workflow Within WSDF Framework

- Step 3: WSDF service creates resource instance and returns the resource reference (EPR) to the client.
- Step 4: Client sets the resource by sending the EPR and the input data to the same service.
- Step 5: Input data flow between the client and the service.
- Step 6: Service interprets reference information embedded in the request header and saves input data as resource.
- Step 7: Composing WSDF request by using the created and set resource to invoke computational function.
- Step 8, 18, 28: Invocation is sent to target service.

- Step 9, 19, 29: Server retrieves resource that contains data from the resource context by using EPR contained in the invocation request.

- Step 10, 20, 30: Invoking functional service.

- Step 11, 21: Forward the result generated from current service to the its successor service. This step implements WSDF specification, which is different from WSRF.

- Step 12, 22: Create data resource instance on the successor service and save the current service’s result to this resource.

- Step 13, 23: The created EPR to the data resource on the successor service is returned back to current service.

- Step 14, 24: The EPR of the data resource is returned to the workflow engine by the current service, which is now completed.

- Step 15, 25: Control flow travels between different part of the workflow components that contains the EPR.

- Step 16, 26: EPR is passed within workflow engine from current part to its successor part.

- Step 17, 27: EPR is forwarded to the client to compose WSDF request. Together with resource forwarding information, the WSDF request is to invoke the first service. In step 27, as it is the last step, there is no resource forwarding information.

- Step 31: Send the output data of the last service directly back to the client.

- Step 32: Client receives the result of the workflow and exits.

3. WSDF Implementation

The WSDF implementation includes three main parts: WSDF service server, WSDF service and WSDF client. As WSDF is built on WSRF, we chose WS-core [15] to build the WSDF service server. WS-core is the Web
service server of the *Globus* [14] software toolkit, which implements WSRF and has been widely used by the research community. Our implementation meets the WS-I basic profile [26] standard, which is proposed by the Web services Interoperability Organization, an open industry organization, to establish best practices for Web services interoperability [27]. The implementation also supports Web service with attachment to improve large\(^3\) binary data transfer speed for computational Web services.

### 3.1. Implementation Details

#### 3.1.1. WSDF server

In our implementation, we build WSDF Web service server based on *WS-core*, which is built on **Axis1.1** [22]. By using the *WS-core* source code obtained from *WS-core* project\(^4\), we rebuilt the SOAP engine for WSDF service. We name the modified SOAP engine 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, saves this information temporarily, and forwards result data after it is generated by a back-end application. After data forwarding, result data is saved as a resource on the *successor service* and referenced by an EPR generated by that service. The EPR 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 in the request, the WSDF server will return the result directly back to the client.

Figure 7 illustrates how the WSDF-axis engine works when a functional service is invoked on the server side.

Steps in Figure 7:

- **Step 1:** Input SOAP message including *resource forwarding* information, resource information and service information.

- **Step 2:** Other SOAP engine processes.

- **Step 3:** Set the context of current service instance.\(^5\)

\(^3\)Less than 2G bytes. This is decided by the maximum value of the *int* type in Java language.

\(^4\) *WS-core* version of Axis is a “enhanced” version that supports WSRF functions.

\(^5\)Each WSRF service instance has its own context, which is set by the SOAP engine. The SOAP engine sets the individual service instance context according to *ReferenceParameters* value in EPR which is sent by the client.
- Step 4: Retrieve resource forwarding information from the SOAP message envelope and save them temporarily for future usage. The information could contain information for multiple successor services, as shown in Figure 9.

- 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 current service instance, carries out the functional processing and finally 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 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 resource referred by the previous created EPR. The SOAP envelope
within the request is composed by the SOAP engine using information that 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; else, return the result to client.

Computational Web services often need to transfer binary data between different participants. The binary data can either be transferred by applying encoding schemes such as Base64 [28] or by using Web service with attachment [19, 20]. Our implementation supports Web service with attachment, as it will save both network resources as well as computational resources and is much faster than using encoding schemas such as Base64.

3.1.2. WSDF Client

Currently, a WSDF 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. In the future, stubs can be built automatically from WSDL by using WSDF-compliant stub generator, as with current tools to easily convert applications to Web services, which will benefit developers. The SOAP envelope should also contain resource forwarding information. To achieve better Web service interoperability, the implementation uses document-literal style SOAP message that follows WS-I [26] specification.

3.1.3. WSDF service

A WSDF service should provide two more operations besides its computational service. First, a create operation, for the client of a WSDF service to create a resource instance on that service before the functional service is called. By invoking the create operation, a resource instance is created for that service and an EPR should be returned to the client. Second, the operation for the client to set the resource. SetResourceProperty [11], as a default operation for WSRF service, can be used to set the content of a resource. However, to have the WSDF service to support Web service with attachment for binary data transferring between different services, a separate resource property operation setAttachAsResource should be implemented.

An invocation to the setAttachAsResource operation has the binary data to be sent embedded in the attachment part of the invocation. When the
operation is invoked, it gets the data from the attachment of the invocation message and sets as the resource. Or, the `setAttachAsResource` operation can 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.

Figure 8 illustrates an EPR example pointing to a resource instance. There are two parts within each EPR [18]. First, the URI which indicates the location of the Web service, in this example it is

```
http://129.127.10.133:8010/wsrf/services/RgbWSDFAttService
```

Second, a reference parameter, which in this example is called `RgbWSDFAttKey` (stands for key of RGB WSDF service with Attachment support) with an integer value is used to distinguish current data resource instance from other resource instances of the same service.

```
<ns1:RgbWSDFAttReference
   xsi:type="ns2:EndpointReferenceType"
   ......
xmlns:ns2="http://www.w3.org/2005/08/addressing">
<ns2:Address xsi:type="ns2:AttributedURI">
  http://129.127.10.133:8010/wsrf/services/RgbWSDFAttService
</ns2:Address>
<ns2:ReferenceParameters
   xsi:type="ns2:ReferenceParametersType">
  <ns1:RgbWSDFAttKey>25626358</ns1:RgbWSDFAttKey>
</ns2:ReferenceParameters>
</ns1:RgbWSDFAttReference>
```

Figure 8: Endpoint Reference (EPR) example

### 3.2. Building Workflow With WSDF framework

To develop a simple example to demonstrate a workflow with data forwarding mechanism within WSDF framework, and to test performance of the system, we built both a Web service named `RGB` and its client that meets WSDF specification. The service is called `RGB` (The name is the acronym stands for red, green and blue) and it takes the content of a `.bmp` image file as input and changes the color of the pixels in the file. It provides `create`,
setAttachAsResource and convert operations and is deployed on three WSDF servers. Convert is a functional operation to convert colors of images. 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. The setAttachAsResource operation is used to set the attachment of the setAttachAsResource request as the resource which is to be processed by the convert operation. A similar operation setResourceProperty is defined in WSRF as the interface to set content of a resource, but it does not support attachment data setting. When the request for convert operation is received, the server will process the previously saved data and generate the result data. After the processing, red color component of pixels will be changed into green; green to blue and blue to red. If there is forwarding information within the convert request, the server forwards the result to the destination; if there is no forwarding information in the request, the server will return the result data directly to the client. For our test, three RGB services are deployed on three WSDF compliant servers.

A workflow which is composed of three segments of service invocations is built to carry out the computation. The task of the workflow is to have a 16 bit .bmp image to be processed by three RGB services (RGB_A, RGB_B and RGB_C) one after another and return only the final image back to the client. The returned image should be identical to the original file.

There are three parts within the workflow, each part contains a RGB service client and is responsible for invoking the service on one server.

1. In the first part, the client first invokes the create operation and generates a resource instance on the server side. An EPR pointing to that data resource instance is returned to the client. Then, the client invokes setAttachAsResource operation to send the .bmp file as a Web service attachment to the server. The server will save this attachment into a file and set the file name as the resource. Finally, the client will invoke the convert operation. As the workflow wants the generated result to be forwarded to the next RGB service, the forwarding information of the second RGB service should also be composed into this convert operation request. During the conversion, the operation will retrieve the file name from the resource, read in binary file and convert the red, green and blue colors respectively. Then write the result back to another temporary file and save this new file name as the resource. The file name is returned to the SOAP engine for it to forward the file
to RGB_B. After forwarding the result as resource of RGB_B, the SOAP engine will return the endpoint reference of this resource to the client and the first part of the workflow finishes.

2. In the second part, the client will invoke the service on RGB_B. As this client has got the EPR of data resource on the second service, it does not need to carry out invocations of create or setAttachAsResource operations carried out by RGB_A service. Instead, it can directly invoke the convert operation on RGB_B. Meanwhile, as the workflow needs the third service to process the result of the second operation, the client needs to encode the third service information into the convert request as resource forwarding information, so the second server can forward the result generated by RGB_B to RGB_C. As with the first service, the EPR of the resource on the third service will be returned to the second segment of the workflow and passed on to the third part of the workflow.

3. In the third part of the workflow, the client will invoke the convert operation directly as the resource required for that operation has been saved on RGB_C by RGB_B. If the client does not want the result generated by RGB_C to be forwarded to another service, it can invoke the convert operation directly without composing any other forwarding information into the request. Since RGB_C can not find any resource forwarding information in the request envelope, after carrying out the operation, the WSDF engine will return the result data directly back to the client as Web service attachment.

On completion of the whole workflow, the client will get a converted .bmp file. Figure 9 shows a segment of request code with Web service resource forwarding information. The XML element with name forwardInfo is defined within the wsdf namespace and contains the necessary information for the WSDF server to compose a SOAP envelope on the fly and forward the result data.

The value of <wsdf:namespace>, http://rgbwsdfatt.com, is the namespace of the successor service to be invoked for data forwarding, and the serviceURL of the successor service is 

The URLs for create operation and setAttachAsResource operation are given in the wsdf:createOperationURL and wsdf:setOperationURL respectively. The wsdf:attachmentResourceForward element indicates the data should
be forwarded to the successor service as Web service attachment. This element has two attributes FileNameAsResource and attachmentFormat. FileNameAsResource shows the resource property on current server is going to be saved as a file and file name is saved as the resource property. Therefore, for the WSDF server to forward the data, it should not forward the resource property directly, but take the resource property as a file name and read real data from this file. The attachment format is shown in the other attribute attachmentFormat which is MTOM in this example. If there are multiple successor services, the client can attach corresponding forwardInfo about these services and the server will forward the data to them as well.

Not all services need the result data to be forwarded (e.g. if the current service is the last service within a workflow). Figure 10 is a segment of sample code without the resource forwarding information. As there is no forwarding information, it does not have wsdf:forwardInfo SOAP element as shown in Figure 10 and it is only a normal Web service invocation request message. With this request, the WSDF-axis engine simply returns the generated result back to the client as a normal Web service server does.

4. Testing

We use the prototype WSDF engine and the workflow environment built on the WSDF schema to demonstrate the advantage of the WSDF framework.

4.1. Testing Methodology
4.1.1. Basic Service Time Consumption (BST)

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 based on their execution time excluding the BST.
Figure 9: SOAP Request with resource forwarding Information

4.1.2. WAN environment for testing

We use WANem [29] to build a WAN emulating environment to carry out experiments. The emulator is set as gateway between client and servers, as well as gateways between servers. By applying different configurations to the gateways, we tested the workflow under different environments.
Figure 11, shows the emulation of a network environment in which three web services are located within a single Ethernet network and the workflow engine is remotely located to these web services in WAN. To achieve that, we install five Linux boxes in a single Ethernet network. The WANem Gateway, as shown in the Figure 11, is a Linux box that runs the WANem software. Host1, Host2 and Host3 represent the servers that host the web services. The solid lines represent the real network connections in a local area network. The connections between Web service providers and the workflow engine go via WANem Gateway (gateway), in order to emulate the remote connection as shown by the dashed line. We can change the network settings on gateway to emulate different network conditions. For example, if we want to emulate an inter-continental connection between the workflow engine and the Web service providers, we can set the delay time on the gateway to half of the round trip time (RTT) between the client and the service provider, then add the gateway to the route of both providers and workflow engine.

4.1.3. Different Latency and Bandwidth setting for Workflow Environment

In a real world workflow system, clients and services are distributed in various physical locations. This often means the network properties (e.g., bandwidth and latency) vary between different parts of the whole workflow. The latency and bandwidth between client and services, as well as between services, can affect the network performance benefits brought by WSDF framework. Our experiments are designed to test performance improvement in a real environment, so we set different latency and bandwidth to emulate inter-continental, intra-continental and local area network respectively.

As shown in Figure 11, a Linux box that has been installed with WANem software works as the gateway that emulates the environment. We set the
remote network emulation with WANem latency and bandwidth of the gateway via the network interface provided by this gateway. Any two participants, e.g. participants A and B, that want to emulate such bandwidth and latency condition between them, should add this gateway on its path to the other participant. For example, with A, it should add the gateway on the path to B and vice versa.

Similarly to our previous experiments [30], 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 do not set any latency and the bandwidth is 100Mbits/sec. In this, we can investigate three different network environment.

4.1.4. Service

Different services will take different processing time. To avoid these differences, we use the same RGB web service (see section 3.2) for all services.

A workflow can involve multiple services. Within a WSDF web service workflow, if each service uses the result from the previous service, only the first and the last service needs to send the data between the workflow engine (i.e. client) and the service. Other services can forward their data to their successor service. Normal Web service workflow, on the other hand, needs to
send data between Web services and their workflow engine. The more services in a workflow involved, the more efficient we expect the WSDF framework to be.

We will test our WSDF framework with four different workflows. They are composed of 3, 6, 9 and 12 services respectively.

4.1.5. Data Size

The data size that a workflow processes has an impact on total performance of the workflow, so we provide different data sizes to be processed by the workflow. With bigger data size, a WSDF workflow can save more time when compared with a normal web service workflow. However, the percentage of the time saved on data transfer will not change much.

We have tested different data sizes to show that our system is suitable for both small and large data sizes. The data size in our experiments ranges from 100kB to 2GB (as we use Java based WSDF-axis engine to carry out the experiments). We use 100k, 500k and 1m bytes files as small size files; 5m, 10m 50m and 100m bytes files as middle size files; 500m, 1G and 2G bytes files as big size files.

4.2. Experiment Environment

We now describe our test environment, and other aspects of our tests of data transfer using web service.

The computers used for the tests have the following specifications. We have four Linux boxes and all of them have two Intel(R) Pentium(R) 4 CPU 2.80GHz with 1,034,604kB memory. Three of them are used as servers and one of them is used as client or workflow engine. We also use a Linux box as the gateway to emulate remote network connection.

Based on the testing methodology discussed in the previous section, we carried out both normal Web service based RGB workflow and WSDF framework based workflow. Due to the variability of network performance on the shared networks that we used for our tests, all the tests were run multiple times, and the results plotted in the figures are the average value.

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.

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

\[
T = \sum_{i=1}^{n} (DI_i/BW_{C,i} + DO_i/BW_{C,i})
\] (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 \( n-1 \)th to transfer output

\(^6\)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.
data from the current service to its successor service; last, the output data transfer time from the nth service to the client.

\[ T' = \left( \frac{D_1}{B_{C,1}} \right) + \sum_{i=1}^{n-1} \left( \frac{D_O_i}{B_{i,i+1}} \right) + \left( \frac{D_O_n}{B_{C,n}} \right) \]  \hspace{1cm} (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. \( D_O_i \) equals \( D_{I,i+1} \)).

\[ T - T' = \sum_{i=1}^{n-1} \left( \left( \frac{D_O_i}{B_{C,i}} \right) + \left( \frac{D_O_i}{B_{C,i+1}} \right) \right) - \left( \frac{D_O_i}{B_{i,i+1}} \right) \]  \hspace{1cm} (3)

Equation (3) illustrates the difference between the two frameworks. For both WSDF and normal Web service workflows, from the first to the nth service, each service transfers the same intermediate data \( D_O_i \) from one service to the next, but via different network paths. \( \left( \frac{D_O_i}{B_{C,i}} \right) + \left( \frac{D_O_i}{B_{C,i+1}} \right) \) represents the transfer time under Web service framework: the intermediate data \( D_O_i \) is first sent from the ith service to the client via a network with bandwidth \( B_{C,i} \), and then sent from the client to the \( i+1 \)th service via a network with bandwidth \( B_{C,i+1} \), i.e., the data transfer goes via a third party (the workflow engine); within a WSDF framework, the transfer time is \( \left( \frac{D_O_i}{B_{i,i+1}} \right) \), the intermediate data \( D_O_i \) is sent from the ith service to the \( i+1 \)th service directly via a network connection with bandwidth \( B_{i,i+1} \). 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 \( B_{i,i+1} \) is much larger than the ones between the servers and client \( B_{C,i} \).

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

\[ P = \frac{T - T'}{T} \times 100 \]  \hspace{1cm} (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 \( D \); second, all bandwidth between client and the servers are the same, represented by \( B_{C,S} \); last, all bandwidth between servers are the same - \( B_{S,S} \). Based on these assumption, the percentage of transfer time saved by the WSDF framework over Web service workflow is as following:
\[ P = \frac{D \sum_{i=1}^{n-1} \left( \frac{2}{BW_{C,S}} - \frac{1}{BW_{S,S}} \right)}{D \sum_{i=1}^{n} \left( \frac{2}{BW_{C,S}} \right)} \times 100 \]  

(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 [31]).

In a LAN environment (Figure 12), WSDF workflows have advantages over Web service workflows in most cases; and in remote environments (Figure 13, 14), 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 3.1.1). 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 have the most improvements and local network WSDF services have the least improvements.

Figure 15: Time Saving Comparison

As shown in Figure 15, 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 16 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.

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 [6, 32], the relationship of control flow and data flow within a workflow system is illustrated. FICAS (Flow-based Infrastructure for Composing Autonomous Services) is introduced as an infrastructure for service composition that supports distributed data-flows. 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), can establish data-flow directly between them under the centralized controller in the FICAS run-time environment. The centralized controller coordinates the control flow between different autonomous services. The data flows are formed between autonomous services without going via the central controller. The FICAS autonomous 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
We believe it is not a good choice to encumber Web services with an additional mediator that supports ASAP protocol.

In [33, 10], the authors described algorithms to convert a workflow, which is typically in BPEL4WS [34] language specification, into smaller units, where each unit works as a centralized workflow system. These units are run on 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 improve the overall throughput. This approach is especially good for data driven workflow. But this algorithm also makes the whole system to be more complex and potentially less stable. First, a new communication system needs to be built for distributed units to talk to each other; second, where the units should be located is still an open problem; finally, more exceptions are expected when comparing with a centralized workflow system. In our model, extra control points are not necessary and stability of the workflow is not compromised.

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 targets of 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 similar 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 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
In [35], researchers try to implement direct data transferring by using other underlying mechanisms instead of Web services. However, this approach lacks flexibility as client and server are tightly coupled.

Web Service Choreography Description Language (WS-CDL) [36] has provided a standard way to provide a unique global space for Web service collaboration to define data type and behavior within the collaborative scope. However, it does not address data exchange issue between collaborating services.

7. Conclusion and Future Work

Within a service oriented framework, to share the intermediate data between different services, we introduce the WSDF framework. Though it is based on Web service with WSRF specification, this framework can be extended to any other services with their own state specification. We also implemented a prototype system to prove the WSDF concept.

The main contributions of our work are:

1. We introduce the state concept into workflow, especially, Web service workflow. The composite service which is composed by atomic services in a workflow is recognized as stateful if all atomic services are stateful. The benefit of a stateful workflow is that it can keep the state of the intermediate data between different services: as each atomic service is stateful, result data shared by other services does not need to be sent back to the centralized workflow engine, but only to be transferred between different atomic services, stored and retrieved as resource. Under this framework, distributed data flow between atomic services is possible. From the client’s point of view, each resource on a single atomic service which can be uniquely identified, stored and retrieved is a resource of that composite service. By using the concept of composite service, it is easy to understand that when more services are involved in the WSDF workflow, more data exchange happens within the composite service entity and significant network transfer time can be saved.

2. As it is becoming a common scenario that result from 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 information for result data forwarding 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.

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

According to our implementation, we also observe the following:

The WSDF workflow can perform much better than Web service workflow within intra-continental, inter-continental environment and in most cases within local network environment.

WSDF workflow with more services involved will save higher percentage on network transfer time.

WSDF model suits workflows that involve significant data transfer.

With distributed data flow between different services, WSDF framework can avoid possible performance bottlenecks. If multiple workflow engines are carried out concurrently on a single computer (e.g. a researcher tries to run multiple workflow simultaneously on his/her computer), the network can be overloaded if a large amount of data is exchanged via the centralized workflow engine. Limited by time, we did not carry out related experiments, but these experiments can 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.

In the future, we plan to implement the pull model of WSDF framework. Currently, the WSDF framework only introduces push model for data sharing between different atomic services in WSDF framework. We believe that pull model can bring other advantages, such as more flexibility, comparing with the push model. For example, if there are multiple successor service candidates, when current service is invoked and the workflow engine has not decided which successor service candidate is going to be used, the current service can save the result data of the service as a local resource and give back the reference of the resource to the client. After the workflow engine has finished selecting the right candidate as the successor service provider, it can invoke the operation on the successor service by passing the resource reference on the current service. With this approach, current service needs to provide corresponding service operation to support data transferring when
the successor service is trying to retrieve the resource with the given resource reference.

We will also build client side stub generation tools as currently we manually compose the client side invocation messages. With these tools, it will be much easier for workflow programmers to compose a WSDF workflow.

References


