Composition of Engineering Web Services with Universal Distributed Data-Flows Framework based on ROA

Kewei Duan[1], Julian Padget[1], H Alicia Kim[2], and Hiroshi Hosobe[3]

[1]Department of Computer Science, University of Bath, UK

[2]Department of Mechanical Engineering, University of Bath, UK

[3]National Institute of Informatics, Japan

kd255@bath.ac.uk, jap@cs.bath.ac.uk, H.A.Kim@bath.ac.uk and hosobe@nii.ac.jp

ABSTRACT

The problem of staging data in workflows has received much attention over the last decade, with a variety of user-directed and automatic solutions. The latter are the focus of the first contribution in this paper, where we propose a simple peer-to-peer solution adapted to the needs of RESTful services. The second contribution, is the combination of the data staging mechanism with a simple service deployment mechanism, that is designed to allow applications developed for the command-line to function as (RESTful) services without modification or (in some cases) recompilation. Thus, the aim of this paper is to describe the design and implementation of: (i) a peer-to-peer data-staging mechanism, that is itself RESTful, and (ii) a service deployment mechanism, also following REST design principles, which together form the Universal Distributed Data-flows framework, for the support of data-intensive (RESTful) workflows. We evaluate the framework by means of an engineering workflow developed for multi-disciplinary design optimization. The workflow itself is specified in Taverna, which is a conventional centralized data-staging enactment system. However, by virtue of the underlying services and staging mechanisms described here, the resulting enactment is peer-to-peer (for data), which furthermore permits asynchronous staging, with potential benefits for network utilization and end-to-end execution time.

1. INTRODUCTION

With the development of Web services technology, there is an increasing trend for engineering software to be deployed as Web services and cooperate with each other under an integration framework to form a composed engineering computing environment. Numerous engineering design frameworks utilise RPC-style Web services to enable components to communicate over the network [1, 2, 3, 4, 5]. However, conventionally, Web services composition frameworks have centrally coordinated control-flows and data-flows, so the data associated with the enactment process have to be staged through a client-side management system. Such an approach can easily increase the level of data traffic and the situation is exacerbated where large data sets are concerned. Basically, the engineering software discussed here have the features of large data I/O requirements and a relatively small number of service capability invocations.

Data staging and how to control it are not new problems. Already in 1997 [6], adopted the idea of distributed data-flows in a service composition framework to improve data transfer performance, as did also [7] some years later. Similar ideas are embodied in some distributed program execution engines, such as [8, 9], to overcome the bottleneck of data transfers. Meanwhile, several workflow management systems took up a peer-to-peer style mechanism for intermediate data movement[10, 11, 12]. Although there are differences in detail between the various aforementioned solutions, there is one common aspect, namely the use of a private – by which we mean internal, or closed – mechanism (functions are exposed by a set of developer defined specific interfaces and operations) to handle data transfer. A further point in common is the need for addressability: in each case the data objects are assigned some unique label that allows them to be accessed from any location on the network that is participating in the enactment process.

In recent years, the REST architectural style[13] and REST-compliant Web services [14] have emerged and are rapidly gaining popularity due to its flexibility and simplicity. At the same time, to parallel the development of Service-Oriented Architecture (SOA) alongside arbitrary web services [14], we can observe the development of the concept of the Resource Oriented Architecture (ROA). ROA defines a specific set of web-based system design principles derived from the implementation of the REST architecture [15], namely addressability, statelessness, connectedness, and a uniform interface. Thus, based on URI addressability, resources in REST-based system can be accessed universally through uniform methods. This constitutes the architectural base for the Universal Distributed Data-flow framework we propose in this paper. Specifically, Universal Distributed Dataflow provides peer-to-peer data-staging as well as linking for RESTful Web services. Its core function *Datapool* service acts as a broker for data management, so that input and

WS-REST 2012, April 2012; Lyon, France

output (data) files are stored as resources that can be accessed through URIs and data transfers are carried out between Web services directly. Additionally, the *Service Factory* provides a means to generate RESTful services from legacy codes, which utilises the Datapool service to deliver inputs and save outputs. The objective of the Service Factory is to make deployment a simple task for non-specialist users. Thus, the aim of this paper is to describe the design and implementation of: (i) a peer-to-peer data-staging mechanism, that is itself RESTful, and (ii) a service deployment mechanism, also following REST design principles, for the support of data-intensive (RESTful) workflows. We evaluate these services by means of an engineering workflow developed for multi-disciplinary design optimization. The workflow itself is specified in Taverna, which is a conventional centralized data-staging enactment system. However, by virtue of the underlying services and staging mechanisms we describe here, the resulting enactment is peer-to-peer (for data), which furthermore permits asynchronous staging, with potential benefits for network utilization and end-to-end execution time.

The remainder of the paper is organized as follows: Section 2 presents related work about peer-to-peer data staging solutions and RESTful Web service composition; Section 3 briefly sets out the requirements for Universal Distributed Data-flows; Section 4 introduces the implementation by means of *Datapool* service and *Service Factory* service; Section 5 presents an example application in the domain of engineering design optimisation; Section 6 presents some performance figures concerning the enactment overheads and the relative cost of centralized versus decentralized for the engineering workflow. The paper finishes with Section 7 conclusions and future work.

2. RELATED WORK

We examine two strands of related work: peer-to-peer data staging solutions and RESTful service composition.

In early research on peer-to-peer data staging solutions, a distributed data management architecture for workflow management system is adopted in the distributed workflow environment Exotica/FMQM [6]. The motivation was to resolve the problem of poor performance when data-flow is embedded with control-flow. The solution, as implemented, only works on a local-area network(LAN), by means of a set of loosely synchronized replicated databases. Several other workflow management systems have adopted such an architecture, but using more recent technology, namely Triana[11] and Kepler[10], both of which applied JXTA[16] – a peer-to-peer protocol – to allow services to exchange messages. Both use an internal mechanism to identify resources in the network based on a unique ID. GridFlow [12] offers a slightly different solution at the implementation level, utilising an agent-based resource management system, whose task it is to transfer between peers realized as agents. The Globus Replica Location Service(RLS) [17], used in some Grid-based workflow management system for peer-to-peer data transfers, takes a similar approach, but with different components: it maintains a simple registry that keeps track of where data resides in the Grid environment. RLS also has a web service incarnation, called WS RLS, which is based on Web Services Resource Framework (WSRF). This works with the WS-addressing specification to identify data at messaging level. The common feature in all of these, is either the use of an extra layer of private protocols, above Internet application layer protocols, or the introduction of a potentially complicated (internal) system to achieve peer-to-peer data transfer. We believe an open solution to peer-to-peer data transfers or distributed data-flows can be achieved by simpler Web services based architecture like ROA and in which data objects can be addressed just by URIs.

The second strand is concerned with (RESTful) service composition – or workflow construction – depending on how the problem is viewed. In [18], a new language Bite is proposed to describe RESTful services composition, which is capable of describing both control-flow and data-flow. In other work, existing composition languages are applied or extended to fulfil the requirements for describing RESTful services composition: for example, JOpera [19] and BPEL [20]. All of these provide some effective methods for services composition and enactment, especially for business process [21]. However, our concern is more for the data handling issues raised by the composition of data-intensive processes. By simply using direct messaging in the form of XML and HTML forms for data transfers, we may have to confront the likelihood that control-flow and data-flow are still centrally coordinated. This may not be a critical issue for business process, especially when the size of data object contained in a message is small.

3. UNIVERSAL DISTRIBUTED DATA-FLOWS

Inspired by the principle of distributed data-flow and peer-to-peer data transfer, we aim to design a framework for Web services composition based on ROA, which is able to alleviate the communication bottleneck between client and server.

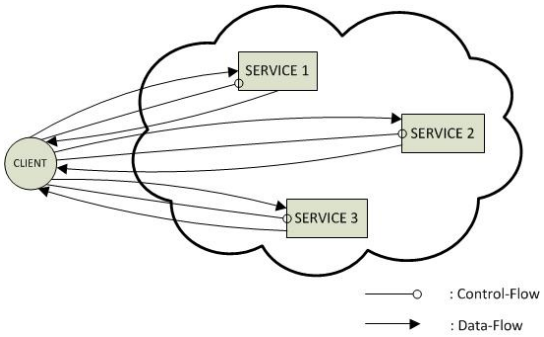

Figure 1: Centralized Data-Flows in Web Services Compostion

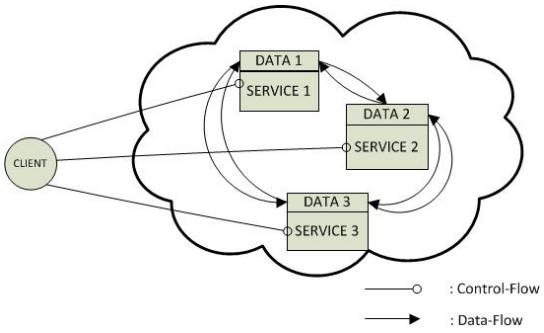

Figure 2: Distributed Data-Flows in Web Services Composition

For the objective of designing IT architectures, Pautasso et al. conclude that RESTful Web service is a better choice in the context of a cooperative environment that it is intrinsically loosely coupled [22]. Therefore, it also fits naturally with the design of a Web services composition framework, where the components of a computation process may originate from various service providers. Furthermore, because of the ROA design principle of addressability derived from the REST architecture, it is inherently supported that the resource representations can be universally located and accessed in Internet with the support of URI. With this basic factor for peer-to-peer data transfers, we can put forward a framework for supporting Web service composition with universal distributed data-flows based on ROA.

Figures 1 and 2 illustrate the essential difference between a centralized mechanism and Universal Distributed Dataflows. Figure 1 shows that both control-flow and data-flow are centrally coordinated for each Web service invocation. There is a high risk that the client or central controller becomes the bottleneck for data communication among computation components in this case. In Figure 2, the dataflows are distributed among Web services directly rather than passing through central controller. The I/O data communication carries out in Internet between each related Web services. Each computation service is associated with a data management service situated at the same physical location or domain name. The client can also obtain the complete data objects from the Internet whenever it needs. Hence, each service provider takes care of the task of data storage instead of the client. Furthermore, each data object must have the capability to be identified and accessed universally through the Internet.

In order to implement the universal distributed data-flows in Web services composition framework, there are several basic design requirements to be considered. First of all, the addressability of data objects. By following the design principles of ROA and the URI protocol, we are straightforwardly able to achieve the requirement of enabling data objects being identified and accessed universally in Internet. We observe that it is possible to improve data communication performance by using asynchronous data transfer. In the circumstance that assuming the prerequisite of one service invocation may involve outputs from different sources (services), the references (URIs) to output data objects do not have to be collected by client side and transfer to next service synchronously. However, the data transfer mechanism should allow the next service aware of each data reference as soon as each previous service being accomplished. Concrete example is shown by Figure 3 in Section 4. We can understand this as distributing the data-flows further within each service invocation. It is also a requirement that an appropriately authorized user should be able to manipulate only their own data objects, in order both to secure a given workflow's data, but also to avoid unintentional conflicts with other users. Finally, we require a means for users to be able to deploy services that can be "joined up" via the datapools, without the need to take any special action or needing to modify existing applications so they can operate in this environment.

4. IMPLEMENTATION

We will now describe the implementation of the two main components of the universal distributed data-flows system,

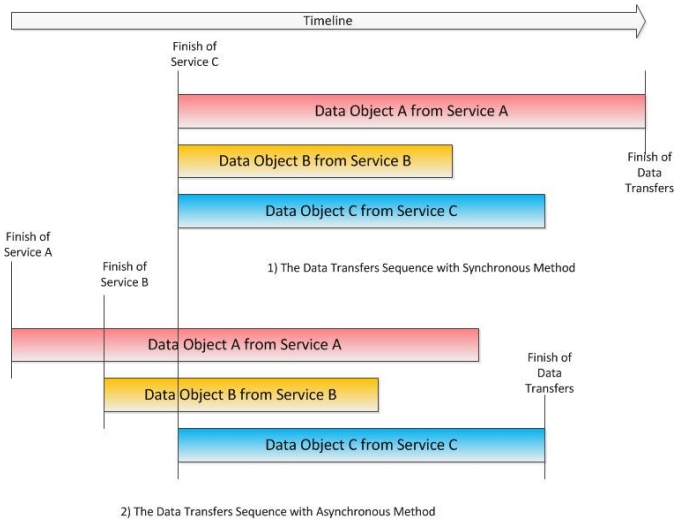

Figure 3: Comparison of two different data transfers management methods

the Datapool service and the Service Factory.

4.1 Datapool service

We can make two quite obvious remarks about data flows within a workflow: (i) for a given service invocation, the data flow rarely involves the client or central controller, which means that data flows can (normally) be distributed, and (ii) it is not uncommon that the necessary data objects (inputs) may come from different sources, suggesting that data transfers can be initiated asynchronously before the actual execution of a service. These constitute the two properties that the datapool service needs to satisfy.

1. Distribution of data between services is greatly simplified through the use of URIs (references). Hence the basic function of the Datapool is to provide data storage for association with a particular service, as shown in Figure 2, whereby data objects are uploaded into a particular Datapool, such as during initialization, or are transferred from one Datapool to another as a result of the control flow. In each case, the object will be given a name derived from the Datapool's URI.

   For the purpose of enabling data being transferred from client to Datapool or between two Datapools, firstly, it exposes an interface for users to inject documents of any format into the repository as either textual data or arbitrary binary data. Datapool service automatically allocate URIs and organize them in the form of a collection of URIs. Secondly, client can submit the URI of the data object to Datapool. Datapool will automatically obtain and allocate a new URI to it. Afterwards, before the execution of service, client merely need to provide the URI of the corresponding local Datapool to Web service, which contains all the necessary inputs. Web services will automatically extract and consume the data it needs based on data object's name.

2. Data-flows in service invocation form a special case of the above, because the data sources for a given ser-

	Methods	URIs
D.	POST	http://.../datapool/{Datapool_Name}/obj/{Data_Object_Name}
	PUT	http://.../datapool/{Datapool_Name}/obj/{Data_Object_Name}
	PUT	http://.../datapool/{Datapool_Name}/obj?DO_URI={Data_Object_URI}
	GET	http://.../datapool/{Datapool_Name}/obj/{Data_Object_Name}
	GET	http://.../datapool/{Datapool_Name}/obj
	DELETE	http://.../datapool/{Datapool_Name}/obj/{Data_Object_Name}
	DELETE	http://.../datapool/{Datapool_Name}/obj
S.F.	POST	http://.../service_factory/service/{Service_Name}
	PUT	http://.../service_factory/service/{Service_Name}
	GET	http://.../service_factory/service/{Service_Name}?DP_URI={Datapool_URI}
	GET	http://.../service_factory/service/{Service_Name}/contract
	DELETE	http://.../service_factory/service/{Service_Name}

Table 1: URIs of Datapool and Service Factory modules

vice may come from several different sources. Consequently, each (output) data object can be transferred asynchronously to the consuming service as soon as the producer has finished.

For example, this process is illustrated in Figure 3, where there are three data objects from three different Web services that need to be transferred to another service as inputs through three different connections. We assume that in two situations, the same data object transfer takes the same time. The length of each bar represents time. Under synchronous data transfer, because the data references are controlled through the client, data transfer only starts when the last service finishes. However, in the asynchronous method, the transfers start asynchronously when each service finishes. From Figure 3, we can clearly notice that the asynchronous method can bring about an earlier completion of the whole data transfer process.

The services of Datapool are accessible through URIs as shown in Table 1. Users can maintain several pools for different processes or for different stages in one process, and only the owner of the pool can control the data objects stored in it and decide who can access them, which provides a degree of security for user data. Data resources in the Datapool have unique URIs so that different users cannot unintentionally conflict each other.

The universal distributed data-flows framework allows data transmission process to happen concurrently rather than serializing the process through the client. The overall efficiency gains are illustrated by the example in Figure 4, where several Datapools are located on different servers. In this scenario, we assume that there are two services located in different servers in one sequential execution process. There are two corresponding Datapool services on each server as well. SERVICE 1 needs INPUT1 and INPUT2 as input files and SERVICE 2 needs output from SERVICE 1 and

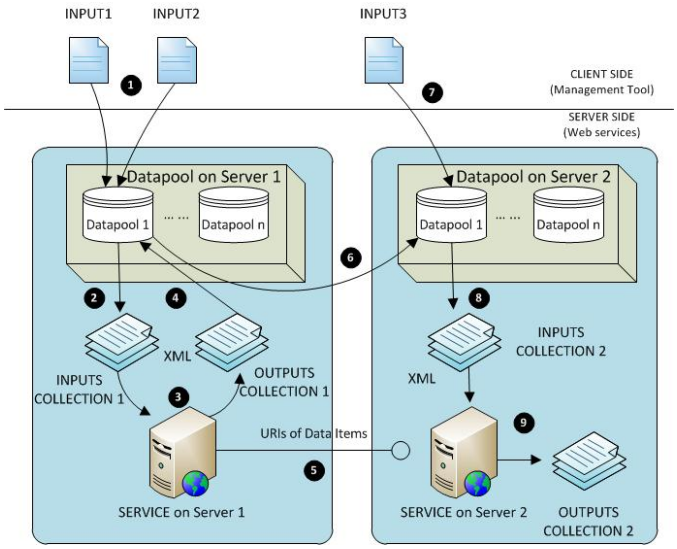

Figure 4: An execution process example based on Datapool service

INPUT3 as input files. INPUT 1 to 3 are original data objects from CLIENT SIDE management tools. OUTPUTS COLLECTION 1, which contains intermediate data objects generated by SERVICE 1, does not need to be transferred to and from the CLIENT. Data transmission only happens between Datapools in each SERVER. URIs of data objects, which are universal addresses of data, are transferred from server to server under the control of CLIENT SIDE management tools. Therefore, in Figure 4, data objects can be pulled from SERVER 1 to SERVER 2 in step 6. Meanwhile, step 7 can be executed concurrently with step 6 as well as step 1. This mechanism can save network bandwidth resource because of the separation of data flow and control flow, which let them work in asynchronous way.

4.2 The Service Factory Web Service

Legacy codes contain much scientific and engineering knowledge. In addition, such programs are often written in a variety of different languages, although typically not in "modern" web-cognisant languages such as Java, but rather FORTRAN or C, or package-based languages such as Matlab or SPSS. In order to fill the gap between non-specialist user and the deployment of RESTful Web services based on Universal Distributed Data-flows, a generic application wrapping mechanism is provided. In the case of SOAP-based services, the service provider typically needs administrator privileges to be able to deploy the service. There can also be technical issues such as restarting the service container and perhaps even some manual reconfiguration of the server. Our aim here is to be able to deploy new web services in a more plug-and-play style, where the service provider should only need to be a user rather than an administrator of the server. Furthermore, the deployment activities should not be limited to the local machine. The idea is that the process of deploying and publishing web services can be achieved through web services via online operations. This design can simplify this process for non-discipline specialist user and allows easy access to geographically distributed software and hardware resources.

```
<WS_Info>
        <Service name="FE_C"
        URL="http://.../REST/service_factory/get/FE_C
        ?DP_URI={Datapool_URI}"/>
        <Datapool
        URL="http://.../REST/datapool/post
        /{Datapool_Name}/{Data_object_Name}>
        <Inputs>
                <Input name="ProDef" MIME="text/plain"/>
                <Input name="Density" MIME="text/plain"/>
                <Input name="OptVar" MIME="text/plain"/>
                <Input name="DecimalPlaces" MIME="text/plain"/>
        </Inputs>
        <Outputs MIME="text/xml">
                <Output name="StEnergy"/>
        </Outputs>
</WS_Info>
```

Figure 5: Service information in XML

```
<WS_Definition>
  <Service_Name ifPublic="no">Aerosolve</Service_Name>
  <Application_Name>aerosolve</Application_Name>
  <Inputs>
    <Input name="aero_input_dat"/>
  </Inputs>
  <Outputs>
    <Output name="aero_output_dat"/>
    <Output name="aero_forces_dat"/>
  </Outputs>
  <STDOut name="stdout"/>
</WS_Definition>

<WS_Definition>
  <Service_Name ifPublic="yes">Instal2asp</Service_Name>
  <Application_Name>instal2asp</Application_Name>
  <Inputs>
    <Input name="DomainFile" qualifier="-d"/>
    <Input name="IAL"/>
  </Inputs>
  <Outputs></Outputs>
  <STDOut name="lp"/>
</WS_Definition>
```

Figure 6: The configuration files of RESTful web services

It is the purpose of the Service Factory to provide this function and in so doing, to let issues around maintenance, security and data management be entirely transparent to the applications provider. Consequently, the methods used to deploy and manage RESTful Web service resources are themselves RESTful Web services rather than local tools.

Thus, the primary function of the service factory is to provide the function of file uploading, by means of the POST operation to the first URI of the service factory in Table 1. All the related files, application itself and metadata about this service are nested into a MIME MultiPart entity for uploading. After this process, the URI of the application service, which is a HTTP GET method, is automatically generated. It is shown in the third row in section Service Factory in Table 1. Here the Datapool URI indicates the Datapool where the application service can find the necessary input data. The output data objects of the service will be allocated a URI as the input data object aforementioned as well. This GET method of application service will return a XML file, which contains all the output data objects' URIs. As a result, it is even possible to access the content through Web browser by typing in the URI directly.

The descriptions of Web services need to be exposed as

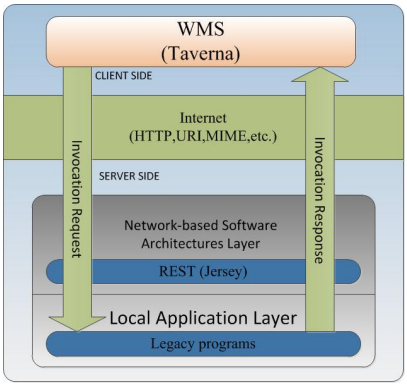

Figure 7: Application Architecture of the Engineering Web Services Composition

URIs for the purpose of service discovery. The web services user or the application that needs to invoke them also can retrieve the information about how to use the Web services via these interfaces. For each application service, there is a pair of URIs corresponding to it. One is the execution method mentioned in last paragraph; the other is the HTTP GET methods URI link to retrieve the metadata of the corresponding application service. The URI for metadata retrieval service is shown in the fourth row in Service Factory section of Table 1. An example of the metadata returned is shown as Figure 5. This URI is exposed to search engine for discovery purposes. Here, we can see all the necessary information to access a service called FE_C. It contains the URI template of the Datapool used for data delivery and the type of each I/O data object.

Therefore, the provider can upload relative files of the application and its configuration file through those provided URIs and methods. The configuration file is written in the form of XML, which is an obvious choice for ease of machine processing. Two examples of the XML files are presented in Figure 6. The configuration file is designed for application command generation, service interface generation and service permit initialization. This series of operations simplifies the process of resource deployment and execution. Because the tools used to deploy Web services are Web services as well, uploading of application related data can be carried out remotely and under the control of user rather than local system administrator.

5. COMPOSITION OF ENGINEERING WEB SERVICES AND RESULTS

Using the Datapool and Service Factory services, we show how a composition of engineering web services is deployed in conjunction with the widely-used service container and workflow management system (Taverna [23]). Datapool services and the Service Factory service are developed in Java and deployed using Jersey [24], which is a REST software framework for REST Web services development. The whole application architecture is shown is Figure 7. Service invocation is achieved directly using a standard Internet application-layer protocol (HTTP), identifiers (URIs) and data encodings (MIME), standardized service deployment (JAX-RS) and simple local programs.

For the purpose of demonstrating our multi-disciplinary

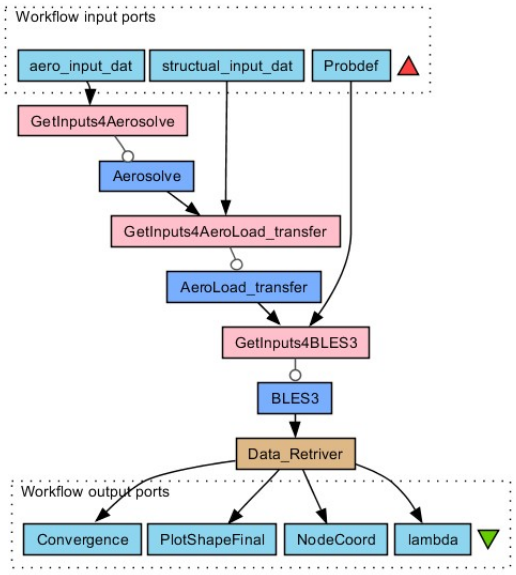

Figure 8: The wing structure optimization process built in Taverna

design optimization process as a Web services composition, we use the Taverna workflow management system [23] to carry out the tasks of composition, execution and monitoring with the support of the services located in Datapool and Service Factory. We also observe that the intermediate data movement of Taverna is categorized as centralized[25, 26], but that Taverna also has the capacity to access RESTful Web services. We demonstrate the functionality of all the components with an example created in Taverna. Figure 8 is a screenshot of the services composition design example, in which the internal stiffness distribution of a typical aircraft wing is optimized under coupled aerodynamics and structural considerations. We define the typical swept back wing of a subsonic civil airliner modelled as a 2D flat plate with 20 strips and 5 chord boxes with a 0^o twist and 22.6^o sweep angle. The root chord of the wing is 7 metres, semi-span is 16.2 metres and the taper ratio is 0.22. The flight condition is set as 0.8 MACH and 35000 feet. The wing structure defined above is input through the file aero_input_dat and depicted in Figure 9(a). For more details of this workflow application, please refer to [27].

In Figure 8, all the dark blue boxes (Aerosolve, AeroLoad-_transfer, BLES3) are RESTful Web services built by Service Factory from three command line programs, written in Fortran or C. All the pink boxes (GetInputs4Aerosolve, GetInputs4AeroLoad_transfer, GetInputsBLES3) are the RESTful Web services for Datapool, whose functions are uploading data as web resources for application services and transferring related URIs of resources. The input ports (light blue boxes) built in Taverna based on RESTful Web services are located at the top of Figure 8, and output ports (light blue boxes) are located at the bottom. One local service Data Extractor is utilised to retrieve the data based on the URI collection return by the last application service.

By using the structural loads transferred from aerodynamics loads by service AeroLoad_transfer, the final topology solution is generated by BLES3 service, which is used for structural optimization. Figure 9 depicts the initial topology and the final optimum solution. The solution is generated based on the numerical result from PlotShapeFinal output port.

(a) Initial topology (b) Final optimum topology

Figure 9: Result of topology optimization

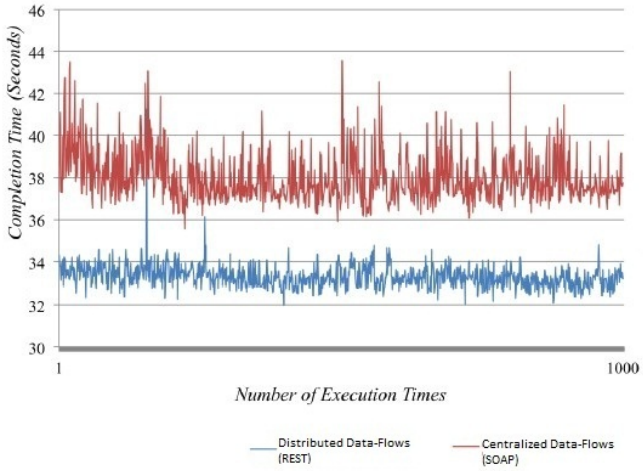

Figure 10: Comparison of 1000 continuous executions

6. PERFORMANCE COMPARISON

In order to be able to assess the performance of services deployed using our new framework, we have run the wing optimization process in two network configurations: 1. with all the programs deployed as SOAP services and controlled by a central client, via which all the data transfers pass, and 2. with the programs deployed as RESTful services, using a centralized client for control, but using the universal distributed flows framework for data transfers. We present a preliminary comparison of these two modes.

We use the same machine and network environment for all the services, namely: (i) Network: VLAN over gigbit (copper) ethernet; (ii) Client: Mac OS X Version 10.6.7: Processor 3.2 GHz Intel Core i3, Memory 4GB; Workflow Management Tool: Taverna 2.2.0; (iii) Server: Linux 2.6.32-31-server: Processors 2.27 GHz Intel Xeon E5520 (*16), Memory 24GB; (iv) VMs: Each VM is allocated 2 CPUs and 2GB Memory; Operating System: Linux 2.6.32-28-generic.

To provide preliminary evidence that the RESTful web services with distributed data-flows performs better than the previous solution, we performed a trial running 1000 consecutive executions for each processe in the same environment. The results are presented in Figure 10, showing that the REST workflow is faster by a clear margin.

We undertook a second experiment to explore the performance gains more deeply. We create a simple workflow

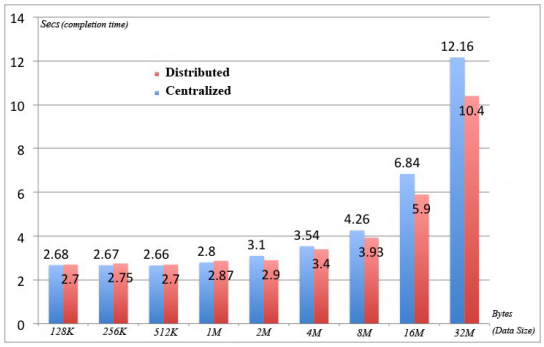

Figure 11: Results of simple workflows execution with centralized and distributed Data-flows

that just moves data from client to first service, then to the second service, and back to client. These two services are deployed in two different VMs on the same LAN as the client. Client and servers address each other by URIs. We set up two scenarios, using REST with centralized and distributed data flows, respectively. In the first, the data transferred from the first service to the second is included in the HTTP body, while in the second scenario only the URIs are transferred. The results appear in Figure 11. Each workflow in two scenarios and different data size conditions are executed 10 times to get the mean value. We can see that the lower data volumes benefit from the inclusion approach, while large data items are better handled by the URI approach. Clearly the crossover point will be dependent on numerous conditions.

7. DISCUSSION AND FUTURE WORK

This paper presents a systematic mechanism for composition of RESTful Web services based on Universal Distributed Data-flows framework. It describes the design and implementation of a peer-to-peer data-staging mechanism, that is itself RESTful, and a service deployment mechanism, also following REST design principles for the support of data-intensive (RESTful) workflows. We evaluate these services by means of an engineering workflow developed for multi-disciplinary design optimization. The workflow is specified in Taverna, which is a conventional centralized data-staging enactment system. However, by virtue of the underlying services and staging mechanisms described here, the resulting enactment is peer-to-peer, which furthermore permits asynchronous staging, with potential benefits for network utilization and end-to-end execution time. At the same time, the simple interface to the web service deployment service allows the framework to be more accessible for non-specialist users.

Based on current architecture, we still have two main issues need to consider. They the security issue of Universal Distributed Data-flow and the description and enactment issue of Web service composition. They are discussed in following paragraphs:

Security Mechanism The security mechanism mainly involves two aspects of issues. One is the security mechanism within data staging process, which is related to Datapool. The other is the risk of untrusted codes that they might be deployed through Service Factory. In the current solution, private input data uploaded by users in Datapool can only be manipulated by corresponding users, output data items generated by services can be read publicly in order to let Datapool to access data through URIs. The authorization on reading output data can be further restricted by setting output as private date and be accessed by delegating permission from user to Datapool. About the risk of untrusted codes, we also will consider to dynamically deploy each service in sandbox to deal with the issues raised by untrusted codes and mis-operation by users. Besides, we also plan to restrict the Datapool size for each user as a means to limit potential abuse of storage capacity.

Declarative Description and Enactment In the current work, the description of control-flows is maintained at client side, which is programmed (visually in the case of Taverna) or written in an imperative style. The description maintains and provides a knowledge base for the enactment of composite Web services. While this composition is itself an integral entity, we do not believe that its interpretation needs to be, so that the elements that comprise it may also become web resources within a ROA. The particular benefit we see arising from distribution of the control flow is an increased capacity for resilience in the case of either execution or communication failure. However, that also depends upon both a declarative and a semantic specification of the composition, each of which introduces a complementary element of late-binding by expressing *what* is required rather than stipulating *how* it shall be achieved. Taking into account work such as [28, 29, 30, 31] (for example), we are exploring the development of a logic based declarative description language for composition of RESTful Web services, which aims to drive the (dynamic) composition process informed by the evolving discussion around the notion of Hypermedia As The Engine Of Application State (HATEOAS). Consequently, the enactment of RESTful Web service composition will be able to be steered by providing appropriate URIs to the client, which link not only computation processes, but also data objects. The Universal Distributed Data-flows framework can act as the foundation for this mechanism.

Acknowledgements Kewei Duan is partially supported by an internship at the National Institute of Informatics, Japan (October 2011–March 2012).

8. REFERENCES

[1] Henry Ng, Suleyman Geluyupoglu, Frank Segaria, Brett Malone, Scott Woyak, and Greg Salow. Collaborative Engineering Enterprise with Integrated Modeling Environment. `http://www.phoenix-int.com/resources/CollaborativeEngineering.php`, 2003. [Online; accessed 08-August-2011].

[2] Abdulsalam Alzubbi, Amadou Ndiaye, Babak Mahdavi, Francois Guibault, Benoit Ozell, and Jean-Yves Trepanier. On the use of JAVA and RMI in the development of a computer framework for MDO. In *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2000.

[3] M. Sobolewski. FIPER: The Federated S2S Environment. In *JavaOne, Sun's 2002 Worldwide Java Developer Conference*, 2002.

[4] Ho-Jun Lee, Jae-Woo Lee, and Jeong-Oog Lee. Development of Web services-based Multidisciplinary Design Optimization framework. *Advances in Engineering Software*, 40(3):176–183, 2009.

[5] Tom Crick, Peter Dunning, Hyunsun Kim, and Julian Padget. Engineering design optimization using services and workflows. *Phil. Trans. R. Soc. A*, 367(1898):2741–2751, 2009.

[6] G. Alonso, B. Reinwald, and C. Mohan. Distributed data management in workflow environments. In *Research Issues in Data Engineering, 1997. Proceedings. Seventh International Workshop on*, pages 82 –90, apr 1997.

[7] David Liu, Jun Peng, Gio Wiederhold, Ram D. Sriram, Corresponding Aruthor, Kincho H. Law, and Kincho H. Law. Composition of engineering web services with distributed data flows and computations, 2005.

[8] Derek G. Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. CIEL: a universal execution engine for distributed data-flow computing. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, page 9, Berkeley, CA, USA, 2011. USENIX Association.

[9] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[10] Kepler Project. `https://kepler-project.org/`. [Online; accessed 17-Feb-2012].

[11] Triana Project. `http://www.trianacode.org/`. [Online; accessed 17-Feb-2012].

[12] Junwei Cao, S.A. Jarvis, S. Saini, and G.R. Nudd. Gridflow: workflow management for grid computing. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 198 – 205, may 2003.

[13] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[14] Web Services Architecture. `http://www.w3.org/TR/ws-arch/\#relwwwrest`. [Online; accessed 08-August-2011].

[15] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, Inc., 1 edition, May 2007.

[16] JXTA. `http://java.net/projects/jxta`. [Online; accessed 17-Feb-2012].

[17] GT Data Management: Replica Location Service (RLS). `http://www.globus.org/toolkit/data/rls/`. [Online; accessed 17-Feb-2012].

[18] F. Rosenberg, F. Curbera, M. J. Duftler, and R. Khalaf. Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. *Internet Computing, IEEE*, 12(5):24–31, 2008.

[19] Cesare Pautasso. Composing restful services with JOpera. In Alexandre Bergel and Johan Fabry, editors, *Software Composition*, volume 5634 of *Lecture Notes in Computer Science*, pages 142–159. Springer Berlin / Heidelberg, 2009.

[20] Cesare Pautasso. Restful web service composition with bpel for rest. *Data Knowl. Eng.*, 68:851–866, September 2009.

[21] Xiwei Xu, Liming Zhu, Yan Liu, and M. Staples. Resource-oriented architecture for business processes. In *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*, pages 395 –402, dec. 2008.

[22] Cesare Pautasso and Erik Wilde. Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design. In *Proc. of the 18th International World Wide Web Conference (WWW2009)*, pages 911–920, Madrid, Spain, April 2009.

[23] Taverna. `http://www.taverna.org.uk/`. [Online; accessed 08-August-2011].

[24] Jersey. `http://jersey.java.net/`. [Online; accessed 16-February-2011].

[25] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3:171–200, 2005. 10.1007/s10723-005-9010-8.

[26] Yun Yang, Ke Liu, Jinjun Chen, Joel Lignier, and Hai Jin. Peer-to-peer based grid workflow runtime environment of swindew-g. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 51–58, Washington, DC, USA, 2007. IEEE Computer Society.

[27] Kewei Duan, YE Vincent Seowy, H Alicia Kimz, and Julian Padget. A Resource-Oriented Architecture for MDO Framework. In *Proceedings of 8th AIAA Multidisciplinary Design Optimization Specialist Conference*, 2012.

[28] W. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23:99–113, 2009. 10.1007/s00450-009-0057-9.

[29] Giulio Piancastelli and Andrea Omicini. A multi-theory logic programming language for the World Wide Web. Technical Report 2515, Alma Mater Studiorum—Università di Bologna, August 2008.

[30] Jose Luis Ambite and Dipsy Kappor. Automatically composing data workflows with relational descriptions and shim services. In *6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, pages 15–28, November 2007.

[31] Rosa Alarcon, Erik Wilde, and Jesus Bellido. Hypermedia-driven restful service composition. In *Proceedings of the 2010 international conference on Service-oriented computing*, ICSOC'10, pages 111–120, Berlin, Heidelberg, 2011. Springer-Verlag.