Fourth IEEE International Conference on eScience

A Web Services Architecture for Visualization

Jason Wood, Ken Brodlie,
Jungwook Seo, David Duke
School of Computing, University of Leeds
Leeds LS2 9JT, United Kingdom
{jason, kwb, jungwook, djd}@comp.leeds.ac.uk

Jeremy Walton
The Numerical Algorithms Group (NAG) Ltd
Wilkinson House, Jordan Hill Road
Oxford OX2 8DR, United Kingdom
jeremy.walton@nag.co.uk

***Abstract*—Service-oriented architectures are increasingly being used as the architectural style for creating large distributed computer applications. This paper examines the provision of visualization as a service that can be made available to application designers in order to combine with other services. We develop a three-layer architecture: a client layer which provides the user interface; a stateful web service middleware layer which provides a published interface to the visualization system; and finally, a visualization component layer which provides the core functionality of visualization techniques. This separation of middleware from the visualization components is crucial: it allows us to exploit the strengths of web service technologies in providing standardized access to the system, and in maintaining state information throughout a session, but also gives us the freedom to build our visualization layer in an efficient and flexible way without the constraints of web service protocols. We describe the design of a visualization service based on this architecture, and illustrate one aspect of the work by re-visiting an early example of web-based visualization.**

Keywords: application design; service-oriented architectures; software systems; visualization; web services

I. INTRODUCTION

Service-oriented architectures are increasingly being used as the architectural style for creating large computer applications. The major elements of an application are packaged as services that can be distributed on different hosts; these services can be combined in workflows, and re-used in various ways to create a range of applications, with data being transferred between services according to the workflow. The aim of this paper is to examine the position of visualization software systems within the world of service-oriented architectures.

Visualization is often just one element of a larger computing application. For example, numerical simulations often incorporate a visualization component in order to gain insight into the results, or to steer the simulation into an optimum region of its parameter space. In medical computing, visualization may be combined with database components to allow individual patient records to be extracted, and visually explored, compared and assessed. In the emerging field of visual analytics, visualization needs to be combined with statistical processing in order to explore trends, detect anomalies or simply reduce data to a feasible size.

Our contribution in this paper is the outline of a visualization web service. We are not concerned with other services that might be used in a larger application, but focus simply on a service providing visualization functionality. We consider issues of granularity: should there be a single visualization web service, or should the functionality be split into individual services along the lines of the modular visualization environments developed in the early 1990s? We favour the single visualization service, but retain the idea of modules and dataflow pipelines in a layer beneath the web service interface. We develop a realization of the architecture using current web service technologies.

To illustrate the architecture, we return to an example of air quality visualization that was used in a very early paper on visualization web services in 1996. There are two reasons for this: firstly, we can reflect on the limitations of web technologies at this earlier time, and the way in which advances in this field allow us to provide a different user experience today; and secondly, the simplicity of this example allows us to focus on the novelty of the system architecture, rather than the visualization itself.

II. RELATED WORK

Over the past 14 years, there has been a steady evolution in the use of web technologies for visualization – both client-side and server-side. In the case of client-side applications, many visualization applets have been developed, and are widely used today. For example, the ManyEyes [1] system allows registered users to upload datasets and select from a number of pre-defined applets. A Web 2.0 social networking aspect is added by allowing users to upload their visualizations, with comments added, to a repository from where other users can download the visualization, make their own comments and perhaps make changes. The Google visualization API [2] allows users to create their own Javascript visualization applications, and share these with other users. These client-side applications are typically used for graphs and charts where the visualization processing is straightforward.

Our interest is more in server-side approaches where larger datasets and more complex visualizations can be handled. An early example of this approach was the work of Wood *et al.* [3] in 1996: they demonstrated an air quality application in which a user selected data of interest using a web form; a CGI script was invoked to retrieve the data from a database and run a dataflow visualization system (IRIS

978-0-7695-3535-7/08 $25.00 © 2008 IEEE
DOI 10.1109/eScience.2008.51

IEEE computer society

Explorer [13]) on a server; and a 3D visualization was returned in the form of a VRML scene. We use exactly the same application to demonstrate the web services architecture proposed in this paper, in order to illustrate the advances that have appeared in the last twelve years thanks to the development of modern web service technologies.

The work of Wood *et al*. was followed by a number of similar CGI-based server-side applications, which typically used Java applets rather than HTML forms to provide an improved user interface [4, 5]. A further step was taken by Jankun-Kelly *et al*. [6], who ported their visualization spreadsheet application to the web using servlets to create volume rendering, Javascript to provide the interface and Grid technologies for user authentication and file transfer. Recently Eick *et al*. [7] have demonstrated how thin client visualization applications can be developed by using AJAX and other web technologies similar to those deployed in Google maps.

Our aim in this paper is to explore how the dataflow visualization concept which underlies many popular commercial visualization systems can be migrated to modern web service technologies. In dataflow visualization, elementary processing steps in a visualization pipeline are provided as a set of modules; users can select the modules they need to compose a particular visualization, and connect these modules together in an appropriate network using a visual editor. The idea of implementing dataflow visualization as web services was pioneered by Charters *et al*. [8] and by Wang *et al*. [9]. In their work, the modules of a modular visualization environment become web services. In Wang's case, the notification feature of web services was used to pass the data between the services in the pipeline: the input port of one service 'notifies' the output port of a connected service. A difficulty of equating modules with services is that data transfer must be implemented as XML messages, which involves a considerable overhead. As will be explained later, in our architecture only the pipeline management is a web service, while the individual modules are processes that can communicate using more efficient protocols.

The VisTrails project [10] allows users to build visualization pipelines from modules constructed using the VTK library. The execution control of the pipeline is handled within the VisTrails interface, but for the web-based applications with which we are concerned here, we believe it is preferable to have execution control independent from the user interface.

There is a growing body of research on workflow, particularly within the Grid community. Triana [11] is a workflow-based graphical problem-solving environment that allows distributed components such as Grid jobs or web services to be composed using a visual editor. Taverna [12] provides tools for workflow composition and execution, particularly for bioinformatics applications. These workflow systems can be applied to visualization, but typically the granularity of the workflow components tends to be rather larger than is appropriate for visualization. Moreover they often lack the interactivity and ability to 'plug, play and throw-away' that is characteristic of visualization dataflow programming.

III. Service-oriented Architectures for Visualization

Service-oriented architectures offer a new paradigm for the construction of distributed applications. Here, the application designer orchestrates published computing components (services) into a workflow to fulfill a specific requirement. To support this design, services provide published interfaces using the Web Services Description Language (WSDL) that describe the functions they offer and the data types they require and provide. This approach allows for the re-use of components for the construction of sophisticated applications.

This paradigm shows clear parallels with the visualization dataflow reference model. However, as mentioned in the previous section, the simple approach of implementing each module in a dataflow pipeline as a web service has limitations of performance and reliability. Our approach seeks to avoid these limitations whilst retaining the benefit provided by the standard open interface of web services. We essentially provide visualization as a service as opposed to providing a set of visualization services.

Our architecture can be viewed as a simple three-layer model (Fig. 1) where the client communicates with the web services layer using standard web service calls, but the web services layer communicates with a lower layer of visualization components using a proprietary communications mechanism. This approach gives us one of the benefits of web services, namely a published interface allowing anyone to access the provided services, but allows us to implement the visualization components in a more efficient way.

A. *Visualization Components Layer*

The visualization components layer contains the computation and data associated with the visualization pipeline. The elements that constitute this layer could be implemented as re-useable modules that use socket communication to share data between them in a similar way to web services, but the data can be passed without the

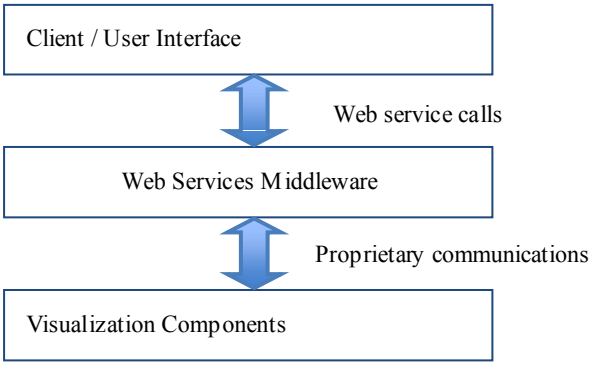

Figure 1. Basic architecture.

overhead of converting it into an intermediate form such as XML or base64 encoded binary (which would be required if web services were being used for this layer). Alternatively, for processes that are running on the same machine, data can be passed via shared memory. Here, a reference to a data object is passed between processes without copying the data, thus reducing the memory footprint of the visualization process. Since visualization typically involves large sets of data, this efficiency saving can be extremely important. If a compute cluster were provided as the hardware system on which to execute the visualization processes, then it would be possible to run parallel visualization components that communicate through MPI.

B. *Web Services Middleware Layer*

There are two current models for web services, *stateless* and *stateful*. As its name suggests, a stateless service performs a task for a given set of input data and retains no information that can be used in a subsequent call to the same service. By contrast, a stateful service allows data generated by a call to that service to be retained and used in subsequent calls to that service. In this case, the calling client uses a unique reference to the service to identify the previous instance of the service.

While stateless services provide an elegant simplicity of design, an architecture for a visualization service constructed using them would be inefficient because any request for the results of a given pipeline would require complete re-execution (in the absence of any cached results from previous executions). Using stateful services in this architecture, the client is able to establish and re-use a visualization pipeline by interacting with the same visualization service instance over a period of time. When a new result is required, previously cached data can be used to shortcut the re-execution of the whole pipeline.

The web services middleware layer provides the glue between the client / user interface layer and the visualization components layer. It offers a standard interface to the visualization pipeline implemented within the visualization components layer. It provides methods for all the basic tasks associated with constructing and interacting with visualization pipelines (e.g. start/stop pipeline component, connect/disconnect pipeline component, set parameter value for pipeline component etc.) as well as being able to load and save visualization pipeline descriptions from/to a file. The middleware layer maintains a model of the visualization pipeline associated with a particular service instance, and is responsible for starting processes in the visualization components layer in response to requests from the client layer, as well as routing interaction requests from the client (e.g. set a parameter value) to the appropriate underlying visualization component.

C. *Client / User Interface Layer*

Our three-layer architecture cleanly separates the user interface from the middleware and the visualization components. This facilitates the development of a range of different interfaces which use the visualization service as the engine to generate the visualization result. For example, a simple interactive visualization application could be delivered over the web using a web browser as a user interface (see section 5). Alternatively, visualization could be part of a larger application; in this case visualization could be incorporated by using web services to access one or more predefined visualization pipelines. Data would be passed to the service and a visualized result returned for display within the application. While these two examples deal with potentially static pipeline descriptions, there is no reason why an application could not be constructed that allowed the dynamic construction of a network of visualization processes. The user could then interact with these processes in the same way that modules are used in a conventional dataflow visualization toolkit [13], changing parameters and connections to effect new visualization results. Unlike these toolkits, however, the user can create an alternative interface without having to abandon the visualization functionality provided.

IV. REALIZATION OF THE ARCHITECTURE

A. *Visualization Components Layer*

We have chosen to implement the dataflow model as the realization of the visualization components layer since it complements the web services paradigm. We have taken existing module code from IRIS Explorer [13] and have replaced its existing framework and GUI with a new environment in which these modules can function. It currently provides a mechanism for allowing modules to make connections to each other at the request of some external agent (in our case the web services middleware), and for the setting of parameter values. It reuses IRIS Explorer's socket communications library for passing data between modules: much of the existing IRIS Explorer module API has been ported to the new environment as well. Taking this approach has provided us with a large set of ready-made visualization functionality. New modules can still be written (using, for example, other visualization libraries such as VTK [14]) and added to the system to extend its functionality.

While using socket communications means that data is copied and passed between processes much as with standard web services, using the native communications library allows data objects to be traversed and transmitted without needing to be re-packaged and then subsequently unpacked on arrival. This gives us efficiency improvements over web services.

The new environment, in addition to providing functionality for modules to execute and communicate, also incorporates a firing algorithm that enables modules to decide for themselves when to execute (e.g. when new data is passed to them).

B. *Web Services Middleware Layer*

The web services middleware layer is realized using the Web Service Resource Framework (WSRF) [15] for stateful and transient web services and ***WS-Notification*** [16] for event delivery. It is implemented using the Globus Toolkit 4 Core [17] and the Apache Tomcat Server [18]. WSRF has

recently emerged from the collaborative efforts of the Web Service and Grid Computing communities, and includes a number of XML-based specifications. Those that we have employed in this work include ***WS-ResourceLifetime*** [19], ***WS-ResourceProperty*** [20], and ***WS-ServiceGroup*** [21]. The main aim of WSRF is to separate stateful entities such as the ***Resource Home*** and ***WS-Resource*** from web services by employing ***WS-Addressing*** [22]. Clients access a specific ***WS-Resource*** using its End Point Reference (EPR) which contains the resource key. The communication between clients and the visualization service is implemented using Simple Object Access Protocol (SOAP) binding stubs [23]. The visualization service is implemented by running a stateful web service – called Pipeline_Builder_Service – within a Tomcat server. This service currently provides a collection of methods to manipulate pipelines, such as `CreateMap`, `StartModule`, `MakeConnection`, `SetParameter` and so on, as shown in Fig. 2.

Clients interact with Pipeline_Builder_Service by invoking the `CreateMap` method to create a ***WS-MapResource*** for which they receive a unique EPR. The ***WS-MapResource*** created in the ***Resource Home*** contains a model of the visualization pipeline implemented as a set of Java objects. These objects are responsible for starting and stopping modules in the underlying visualization components layer at the request of the client, and for maintaining two-way communications to exchange information with the other layers. The client makes subsequent calls to its ***WS-MapResource***, identified by the EPR, to start and stop modules, make and break connections, set parameters of modules within the pipeline or to save the pipeline description to a file. These requests modify the model of the visualization pipeline held in the ***WS-MapResource*** which forwards any required change to the underlying visualization modules.

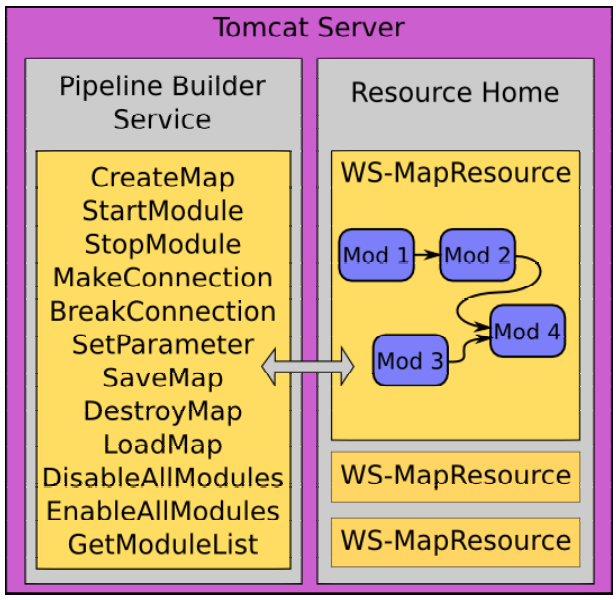

Figure 2. Visualization middleware layer.

In addition to the service methods that allow manual construction of a visualization pipeline, clients can load a previously constructed pipeline from a file. The XML file format used is based on the extended skML format described in [24] with some further additions allowing the identification of visualization results. WSRF allows the lifetime of a resource to be set independently of whether a client is connected, which allows clients to create long-running visualization jobs that can be interacted with over a period of time. So long as the EPR of the ***WS-MapResource*** is maintained, the client can check in and out to monitor progress.

C. Client / User Interface Layer

Our architecture provides for an open interface to the web services layer, which allows for the creation of a variety of user interfaces. To date, two different styles of interface have been developed and tested. First, a prototype graphical user interface designed to operate in the manner of a traditional dataflow environment has been constructed. It offers the basic operations of starting, stopping and connecting visualization components by dragging and dropping graphical representations of modules and wires onto a workspace. When first started, a module list is retrieved from Pipeline_Builder_Service and displayed on the left hand side of the interface. The user launches a module by dropping it onto the workspace, at which point the interface receives a list of input / output ports with data types and a list of parameters with initial values which are used to populate the user interface. This approach allows for the independent development of the server component of the system, adding new modules or modifying existing ones and delivering updates to users at runtime without requiring them to re-install the client. Once a pipeline is constructed and executing, geometry is delivered to the client's desktop for viewing. Fig. 3 shows an example of this interface, as implemented in the .NET Framework.

Second, an interface using a web browser has been

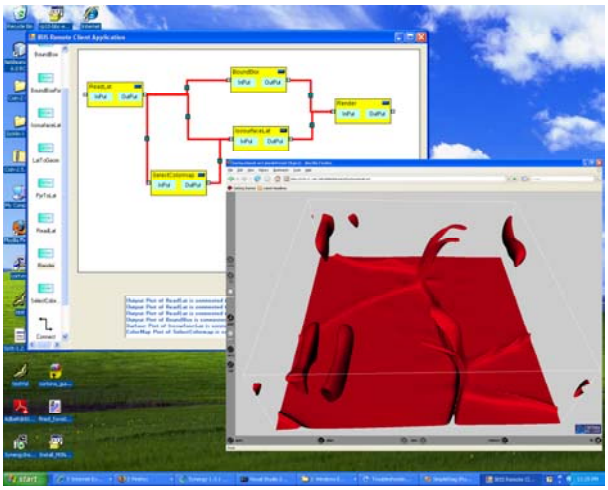

Figure 3. Prototype user interface, showing a pipeline generating an isosurface from volume data, displayed in a VRML viewer.

developed and is used as part of the demonstrator in section 5. For this we have split the client into two parts: the first part is simple HTML that appears in a user's browser and which provides them with a constrained set of visualization options, while the second part is the session manager, which is hosted on a separate Tomcat server [18]. This is accessed across the network and converts simple requests from the browser into web service calls that are directed to the middleware layer.

This session manager (see Fig. 4) is implemented using three technologies, namely Java Server Pages (JSP), Java Server Faces (JSF) and Java managed beans, in order to store data and efficiently interact with both clients and the visualization service. JSF provides standard, reusable components for creating user interfaces as web applications, encapsulating the event handling and component rendering logic. The components in the user interface are capable of interacting with a number of managed beans such as a Session Bean and a Request Bean to store and retrieve data. The managed beans can be set up with three different levels of scope: *Application*, *Session* and *Request*. Application scope is for all clients as long as the web application is running, Session scope is for each client and Request scope is for each request from a client between web pages.

V. Reworking an Early Web-Based Visualization

In a previous paper Wood *et al*. [3] described an implementation of a server side web-based visualization system that used IRIS Explorer. This system was demonstrated using an example visualization application that delivered visualized results for air quality data. The user was able to select a set of data based on a combination of location and chemical pollutant over a selected time period. In addition, a set of visualization options was chosen before the request was submitted. The visualization was delivered in the form of a VRML file that could be inspected on the user's desktop.

The server side implementation used the IRIS Explorer desktop visualization tool to realize the visualization service. In practice, a copy of IRIS Explorer was running at all times with the visualization pipeline active. When a user made a request, the data file was prepared, and then the user's parameters were passed to the pipeline by communicating with a special module through a socket connection. The pipeline executed and created the VRML output, flagging its completion through the use of a lock file. Any subsequent request by that user – for example, switching visualization type – required a complete re-execution of the process. No re-use of any previous data in the pipeline was possible. In this way it is similar to what we would think of today as a stateless web service.

This demonstrator has been re-worked using the implementation of our new architecture. To the viewer, the system appears as before, with the same data and visualization options being offered, and with VRML being returned. The changes are all in the backend systems.

When users open the User Interface page in a web browser, a new session is started for each client. Users are able to not only send requests for a data set from a particular location during a certain time period, but also set options in the visualization pipeline. Upon receiving the first request, the session manager creates a 'map manager' object, which is responsible for interacting with the visualization service, hosted in a separate Tomcat server, as shown in Fig. 5. A 'data driver' object is created to obtain the data set from the air quality data server [25]. Once the requested data has been accessed, the map manager invokes the visualization service to create a ***WS-MapResource*** for a pipeline on behalf of the client. The ***Resource Home*** creates a ***WS-MapResource*** and returns its EPR to the map manager, which can then manipulate it using methods such as `LoadMap`, `MakeConnection`, `BreakConnection` and `SetParameter`. The map manager requests the appropriate map to be loaded and sends then the input data stored by the data driver to the ***WS-MapResource*** as a URL using the `SetParameter` method. The data is passed through the pipeline to generate a VRML file, whose location (in the form of a URL) is stored in a data server and passed to the Request Bean through the map manager. After the map has been executed, the results page retrieves this location from the Request Bean and displays the scene in a VRML viewer in the client's browser.

Once the initial pipeline is loaded, the client can continually interact with the stateful web service through the map manager without creating a new pipeline. Whenever the client sends a request, the map manager can interact with the ***WS-MapResource***, which contains all the stateful objects. For instance, when the client changes a parameter value that affects a module delivering the visualization output, the map manager simply uses the `SetParameter` method to change the option in the existing pipeline. The

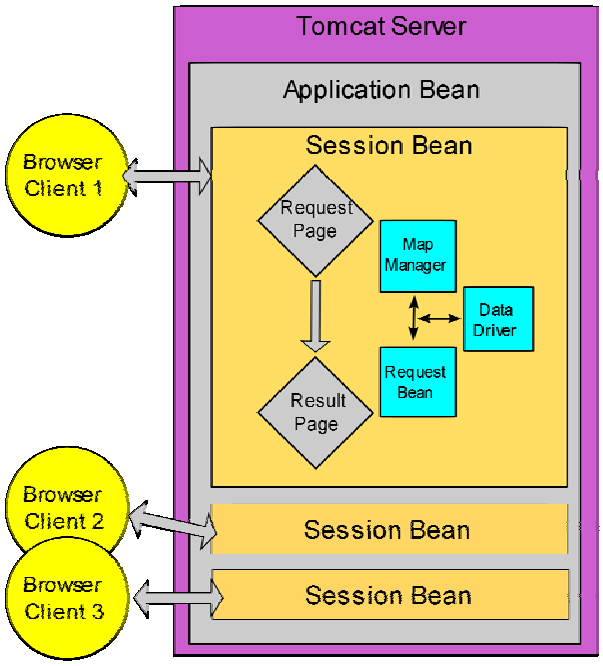

Figure 4. Session manager for web-based visualization client.

system is able to determine which elements of the previous execution can be re-used and only those parts of the pipeline that need updating are executed to generate a new visualization. Fig. 6 shows an example where the user has selected two sites and two pollutants and initially chosen to view them as a surface. The surface is displayed in a new page using a VRML viewer. This page also provides controls to change the visualization type to a 2D histogram; selecting this option takes the viewer to the third page shown in the figure.

Once the user stops interacting with the web browser, the expiry of the time limit associated with the Session Bean in the session manager forces the map manager to destroy the ***WS-MapResource*** in the visualization service. In this way, clients are able to run a stateful web service, which has a lifetime associated with it.

In our architecture, notifications are implemented in accordance with the ***WS-ResourceProperties*** and ***WS-Notification*** specifications. Clients asynchronously receive notifications for both parameter values calculated in the visualization layer (such as data range) and also the firing status from each module process in order to effectively interact with the ***WS-MapResource***. Each module automatically notifies the middleware of the changed parameter value and the middleware then updates it as a ***WS-Topic***. The user interface receives the changed value and updates it. Alternatively, clients are capable of selecting a number of pipeline properties, subscribing to each ***WS-Topic*** with the EPR. For example, clients can subscribe to the status of the whole pipeline.

We note that our web service can be entered into the Universal Description, Discovery and Integration (UDDI) registry, enabling other users to discover and interact with it, by making use of the published WSDL interface. This can be done on a variety of platforms and languages (for example, the Windows .NET client, as shown in Fig. 3). In addition, instances of ***WS-MapResources*** created by other users can be found and monitored by the Monitoring and

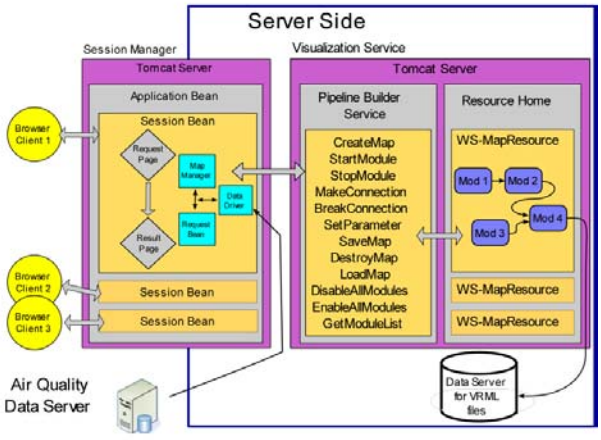

Figure 5. Architecture for air quality web service.

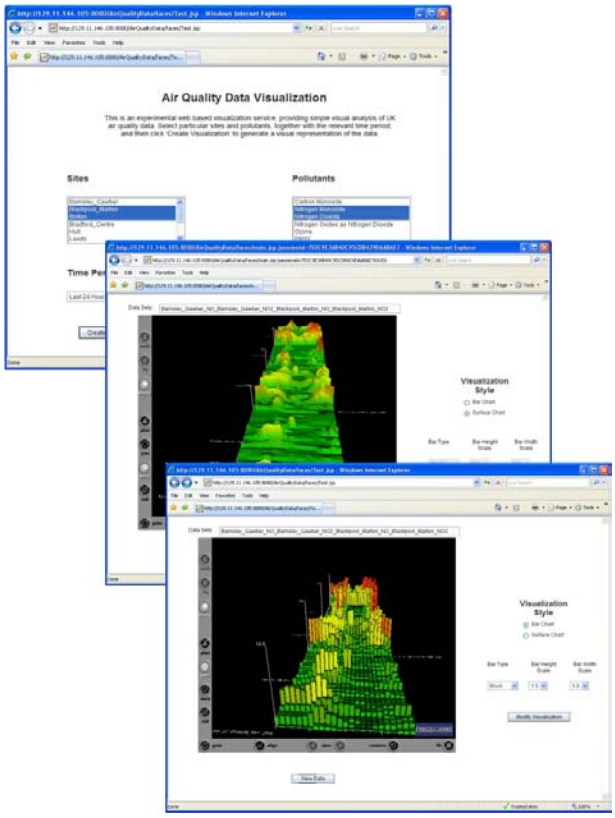

Figure 6. Visualization of air quality data showing data selection page (top) surface view (middle) and histogram view (bottom).

Discovery Service [26] within the GT4 using ***WS-ServiceGroup*** and ***WS-Notification***.

In the original 1996 implementation, each visualization request required the creation and execution of the pipeline from scratch (14 modules, 19 connections) – even when only a small parameter change was involved. By contrast, our new implementation only involves this cost for the first request; by using stateful web services, on subsequent requests involving parameter changes, the pipeline is re-used and only two modules need to be re-executed.

VI. CONCLUSIONS

We have presented an architecture for a visualization system using web services technologies and demonstrated its use for developing web-based visualization applications. Using a web services middleware layer allows us to present a published web services front-end to client applications, while still giving us the opportunity to develop an efficient visualization engine underneath. Additionally, using stateful services for the middleware layer gives us the ability to perform a sequence of interactions over time with the same visualization pipeline and hence gain the benefit of re-using cached data.

Rather than presenting a set of visualization services, where each individual visualization component is a service, we have considered the problem at a coarser level of granularity and offered visualization as a service. We

maintain the workflow paradigm used by the application designer by allowing the visualization service to be configured using a dataflow pipeline (workflow) description.

ACKNOWLEDGMENT

This work was carried out within the ADVISE project funded by the Technology Strategy Board of the UK Department for Innovation, Universities and Skills; partners in the project are NAG Ltd, VSNi Ltd and the University of Leeds. Thanks to Roger Payne and Ian Channing of VSNi for helpful discussions, and to Haoxiang Wang who laid the foundations for this work during his PhD project at Leeds.

REFERENCES

[1] F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss and M. McKeon, "Many Eyes: a site for visualization at internet scale," IEEE Transactions on Visualization and Computer Graphics, Vol 13, No 6, pp. 1121–1128, 2007.

[2] Google Visualization API. http://code.google.com/apis/visualization/

[3] J. D. Wood, K. W. Brodlie and H. Wright, "Visualization over the world wide web and its application to environmental data," Proceedings of IEEE Visualization96 conference, R. Yagel and G. M. Nielson, Eds. pp. 81–86, ACM Press, 1996.

[4] M. Bender, R. Klein, A. Disch and A. Ebert, "A functional framework for web-based information visualization systems," TVCG, 6(1), pp. 8–23, January-March 2000.

[5] R. M. Rohrer and E. Swing, "Web-based information visualization," IEEE Computer Graphics and Applications, 17(4), pp. 52–59, July/August 1997.

[6] T. J. Jankun-Kelly, O. Kreylos, J. M. Shalf, K.-L. Ma, B. Hamann, K. I. Joy, and E. W. Bethel, "Deploying web-based visual exploration tools on the grid," IEEE CG&A, 23(2), pp. 40–50, 2003.

[7] S. G. Eick, M. A. Eick, J. Fugitt, B. Horst, M. Khailo, and R. A. Lankenau, "Thin client visualization," in VAST07, pp. 51–58, 2007.

[8] S. Charters, N. Holliman, M. Munro, "Visualization on the Grid: a web service approach," in Proceedings of the UK e-Science All Hands Meeting 2004, pp. 202–209.

[9] H. Wang, K. Brodlie, J. Handley and J. Wood, "Service-oriented approach to collaborative visualization", Concurrency and Computation: Practice and Experience, 20, pp. 1289–1301, 2008.

[10] C. T. Silva, J. Freire and S. T. Callahan, "Provenance for visualizations," IEEE Computing in Science and Engineering, 9 (5), pp. 82–89, 2007.

[11] I. Taylor, M. Shields, I. Wang and A. Harrison, "Visual grid workflow in Triana," Journal of Grid Computing, 3, pp. 153–169, 2007.

[12] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, A. Wijpat and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," Bioinformatics Journal, 20 (17), pp. 3045–3054, 2004.

[13] J.P.R.B. Walton, "NAG's IRIS Explorer", in The Visualization Handbook, C. D. Hansen and C. R. Johnson, Eds. pp. 633–654. Elsevier, 2005.

[14] W. J. Schroeder, K. M. Martin, W. E. Lorensen, "The design and implementation of an object-oriented toolkit for 3D graphics and visualization," Proceedings of IEEE Visualization96 conference, R. Yagel and G. M. Nielson, Eds. pp. 93–100, ACM Press, 1996.

[15] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, "The WS-resource framework," http://www.globus.org/wsrf/specs/ws-wsrf.pdf, 2004.

[16] S. Graham *et al*., "WS-Notification family of specifications (WS-BaseNotification, WS-Topics and WSBrokeredNotification)", 2004.

[17] Globus, http://www.globus.org.

[18] Apache Tomcat, http://tomcat.apache.org.

[19] OASIS, WS-ResourceLifetime, http://docs.oasis-open.org/wsrf/2005/03/wsrf-WS-ResourceLifetime-1.2-draft-05.pdf .

[20] OASIS, WS-ResourceProperties, http://docs.oasis-open.org/wsrf/2005/03/wsrf-WS-ResourceProperties-1.2-draft-06.pdf

[21] OASIS, WS-ServiceGroup, http://docs.oasis-open.org/wsrf/2005/03/wsrf-WS-ServiceGroup-1.2-draft-04.pdf .

[22] D. Box *et al*. (W3C members), "Web Services addressing (WS-Addressing)," Aug. 2004, http://www.w3.org/submission/2004/subm-ws-addressing-20040810/.

[23] SOAP 1.1, "Simple object access protocol (SOAP) 1.1", W3C, Note 08 May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[24] J. D. Wood, M. Riding and K. W. Brodlie, "A user interface framework for Grid-based computational steering and visualization," in Proceedings of the UK e-Science All Hands Meeting 2007, NeSC Sept 2007.

[25] Air Quality Data, http://www.airquality.co.uk.

[26] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," 10th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, pp. 181–184, 2001.