Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 00 (2010) 1–8
Procedia Computer Science

International Conference on Computational Science, ICCS 2010

Flexible delivery of visualization software and services

Jason Wood[a,*], Jungwook Seo[a], David Duke[a], Jeremy Walton[b], Ken Brodlie[a]

[a]School of Computing, University of Leeds, Leeds LS2 9JT, UK
[b]NAG Ltd, Wilkinson House, Jordan Hill Road, Oxford OX2 7DE, UK

Abstract

An important issue in the design of visualization systems is to allow flexibility in providing a range of interfaces to a single body of algorithmic software. In this paper we describe how the ADVISE architecture provides exactly this flexibility. The architecture is cleanly separated into three layers: user interface, web service middleware and visualization components. This gives us the flexibility to provide a range of different delivery options, but all making use of the same basic set of visualization components. These delivery options comprise a range of user interfaces (visual pipeline editor, tailored application, web page), coupled with installation choice between a stand-alone desktop application, or a distributed client-server application.

Keywords: visualization, service oriented architecture

1. Introduction

Visualization algorithms have been the focus of much innovation and development over the past 25 years, with improvements in their capability and performance being presented annually at the IEEE Visualization conference, and many other conferences and journals. But these advances are only of value to the wider research community when the algorithms are made available through visualization systems that are easily available, easy to use, easy to tailor to different application areas and scalable across a wide range of problem sizes. In this paper we discuss ADVISE, a new framework for visualization software that is designed to be flexible in terms of both its configuration and the interface that is presented to the user, allowing the one system to serve a variety of different computing environments and a wide range of users.

Our aim in this work has been to cleanly separate three aspects of a visualization system: the algorithmic content; the management of the visualization service; and the interface to the user. Through this three-layer reference model we are able to present a single body of visualization techniques in a number of different configurations. Moreover, the framework will allow new technology advances to be introduced at each layer, without disrupting the other layers. Thus new interfaces to the same collection of algorithms can be added; new algorithms can be introduced which are still accessed via the same interfaces, and so on.

Our paper begins with a historical review. Visualization systems have evolved in parallel with general developments in computing. The impact has tended to be cumulative, in the sense that the early approaches to visualization

*Corresponding author
Email addresses: jason@comp.leeds.ac.uk (Jason Wood), jungwook@comp.leeds.ac.uk (Jungwook Seo), djd@comp.leeds.ac.uk (David Duke), jeremy.walton@nag.co.uk (Jeremy Walton), kwb@comp.leeds.ac.uk (Ken Brodlie)

service provision linger on, while new approaches are introduced alongside them. We present our historical perspective in the following section, before introducing ADVISE and its three layer model in § 3. An early prototype of ADVISE has already been presented elsewhere [1]; the contributions of this paper are a demonstration (§ 4) of the flexibility of ADVISE in terms of configuration and interface. We conclude in § 5 with some suggestions for further work, including the extension of the use of ADVISE to visual analytics through the incorporation of statistical algorithms.

2. Historical Perspective

We reflect on the development of visualization systems, from the point of view of both the *user* (in terms of how they create the visualization using the system) and the *computing infrastructure* (in terms of the hardware and software technologies that underpin the system).

In the early days, during the 1960s, visualization was 'programmed' by scripting a sequence of library calls to a graphics package. The scripting might be performed using the programming language - often Fortran - or some domain-specific language associated with a larger application, as was typically the case with statistics packages. Examples of early Fortran visualization systems include NCAR Graphics in US and GHOST in the UK. The computing infrastructure in the 1960s was generally a (relatively) high-performance mainframe accessed from a terminal; this evolved into the workstation/PC world in the 1980s with the same scripting interfaces. This approach is still popular with users today, as evidenced by the success of script-based environments such as VTK and Matlab, whilst NCAR Graphics is still finding active usage.

The late 1980s saw major developments in the design of visualization systems. The dataflow reference model of Haber and McNabb [2] elegantly decomposed the visualization process into a pipeline of smaller, re-usable chunks or modules: different visualizations could be created from the same set of modules, which represented a clear separation between the algorithmic content and the visualization process. The designers of AVS [3] - an early example of a so-called modular visualization environment (MVE) - recognized that the conceptual paradigm could form the basis of an attractive user interface; this realization led to the development of a number of MVEs besides AVS (including IRIS Explorer and IBM Data Explorer) in which the pipeline construction process was exposed to the user. A key aspect of each system was a well-defined data model, which was required to ensure that the output from one module could connect to the input of another. Internally these systems often retained a scripting interface (for example, IRIS Explorer used a language called *skm*, an implementation of the Lisp dialect Scheme) but this was sometimes hidden from the user. Moreover the interface and the management of the service were closely entwined with the algorithmic content, so providing different interfaces could be difficult. However, the dataflow paradigm itself has stood the test of time and is still widely used - for example, VTK uses such a conceptual approach, and we have built on the same paradigm in ADVISE.

From a computing infrastructure viewpoint, the early dataflow systems were designed to work primarily on workstations. In addition, the way in which the visualization process was broken down into a collection of interconnected modules led naturally to the distribution of applications across the network as modules were deployed onto different machines. This facility was further exploited when the emergence of larger datasets (and the increased support for the idea of embedding simulations in the pipeline to enable online computational steering) provided the stimulus for the distribution of modules between the desktop and a supercomputer. As distributed computing technology has evolved, visualization systems have also been extended to encompass MPI technology at a low level (for example, pV3 extending Visual3 [4], and Paraview extending VTK [5]); and at a higher level, Grid computing [6].

The provision of bespoke applications with tailored interfaces has increased as the use of visualization has grown. These present a selected group of control parameters to the user, hiding controls that are extraneous or unnecessary, or those whose value it would be undesirable for the user to change. Examples of the use of tailored interfaces to visualization applications include van Wijk's SequoiaView [7], which is a successful application to view disk space usage on a PC, and Wattenberg's BabyNameWizard [8].

The Web inevitably has had an impact on visualization. From the point of view of the user, it provides a browser environment within which many people feel comfortable; and from the infrastructure viewpoint, it provides, through web service technologies, a platform for distributed computing. Many examples of this genre exist: for example, Jankun-Kelly et al. [9] have ported their visualization spreadsheet to the web using servlets to perform volume render-

ing, Eick et al. [10] have used AJAX technologies to build thin-client interactive visualization applications and Wang et al. [11] have translated the concept of modules in traditional MVEs to services in a Web services world.

Similar systems have emerged for workflow design - for example, Taverna [12] provides a number of tools to compose and execute workflows. However, workflow systems tend to deal at greater levels of granularity than visualization dataflow, and lack the experimental plugging and unplugging of components that is typical of visualization.

Finally, we have seen the influence of Web2.0 in systems such as ManyEyes [13] where users create their own visualizations using a number of predefined applets, and can then share these visualizations with other users.

Thus the landscape of visualization systems is forever changing in response to the widening community of users and the continuing developments in computing technologies. Our aim in this work is to design a framework for visualization systems that is flexible enough to deliver different interface environments to different users and configurable enough to run on different computing infrastructures, all with a single base of algorithmic content. In addition, we would like our framework to be sufficiently future-proof to accommodate new requirements from the user community and exploit new advances in visualization technology and algorithm development.

3. The ADVISE Framework

The architecture of ADVISE was introduced in the paper of Wood et al. [1]. Therefore we present only a summary here, sufficient to provide a background to the discussion on flexibility in § 4 that forms the main contribution of this paper.

ADVISE is structured in three layers, with well-defined interfaces between the layers - as shown in Figure 1. This clean separation into distinct layers is key to the flexibility of user presentation and computing configuration.

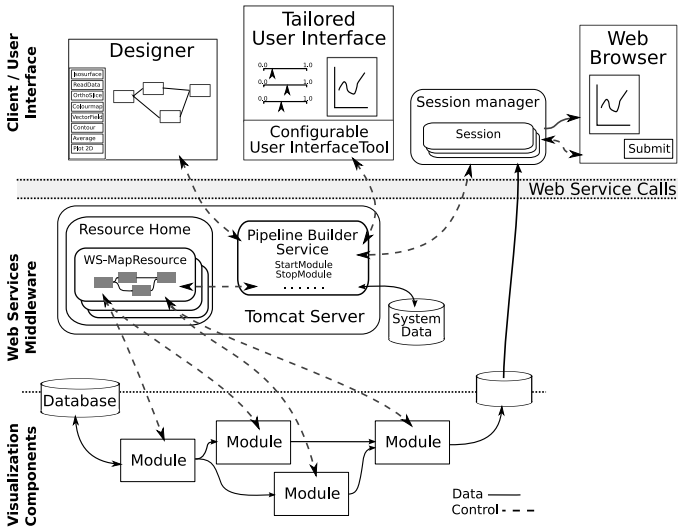


Figure 1: The ADVISE Architecture.

The Visualization Components layer is where the algorithmic software lies. The components are implemented as re-usable modules, as in a traditional dataflow visualization system, with communication between modules using either sockets or via shared memory. It would have been possible to use web services at this layer, making each module a web service as did Wang et al. [11]. This has some attraction, allowing a worldwide collection of modular visualization services to be established, but efficiency concerns dictated our approach. By seeing visualization itself as a service (rather than a collection of modular services), we are able to avoid overheads of data conversion in passing data between services. This approach has also allowed us to use the body of visualization software from IRIS Explorer [14] to provide an initial set of modules, and more are being added.

The ***Web Services Middleware layer*** provides both the glue between the client/user interface and the visualization components, and an insulation layer which is independent of the style of interface and the content at the visualization layer. It manages the visualization service by providing methods for the basic pipeline tasks (starting modules,

connecting and disconnecting modules and so on). It holds a model of the current pipeline, using the skML [15] description language.

A key decision is to use stateful web services. This allows the model of the pipeline to be maintained throughout a period of interaction. In practical terms this delivers significant benefits: when a module parameter is changed, the stateful approach means that upstream data in the pipeline is available and does not require re-computation, as would be needed in a stateless approach.

The ***Client / User Interface layer*** makes use of standard web service calls to communicate with the middleware. This clean separation allows different interfaces to be provided, according to the differing requirements of application developers and end-users, and of novice and experienced users. This reflects the various styles of user interface that have been developed over the years (see § 2).

An important consideration in modern visualization system design is where the rendering takes place, and any system must provide both local and remote rendering. ADVISE provides the twin approach: *local render*, where the 3D geometry is rendered in the client layer; and *remote render* in which the geometry is rendered to an image in the visualization component layer, and a compressed form of that image is transferred, through the middleware, for display in the client layer. We return to this point in § 4.2, below.

In the next section we explore some aspects of the flexibility provided by this framework, presenting examples of different ways in which the framework can be configured, and examples of the different interfaces we have built.

4. Flexible Framework

The three-layer model allows us great flexibility in ways that the underlying visualization software is delivered to the user. We can imagine this as allowing variation in (a) the interface presented to the end-user; and (b) the configuration of the software across the computing infrastructure.

We illustrate the flexibility by showing a single case study - the visualization of a 2D dataset from lubrication studies [16]. The data measures the pressure values in the oil over a region where two journal bearings of an engine make contact. The visualization technique used is to display the pressure as a coloured height map with the height and colour both indicating the pressure. The modules used to achieve the visualization are shown in the following dataflow schematic diagram:

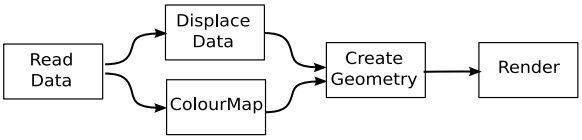


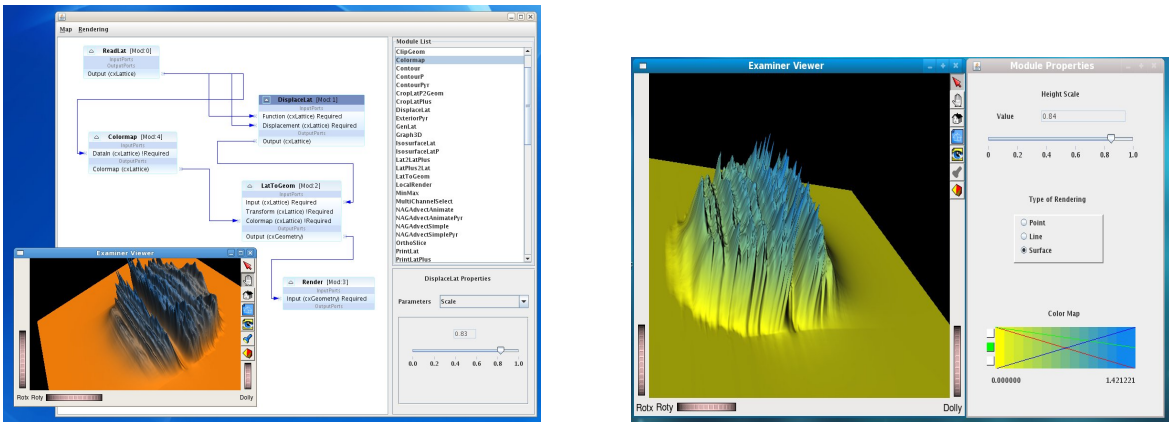
Figure 2: Lubrication application dataflow.

4.1. Interface flexibility

We begin by discussing the variety of user interfaces that can be created. These form different instances of the client layer, and are targeted at different classes of user.

4.1.1. Designer - Visualization Scientist

This tool (see Figure 3 (a)) provides a visual programming facility, as found in many existing MVEs [2, 14] to allow the construction of dataflow pipelines (maps). It allows modules to be selected from a list and connected into pipelines, with a 'plug-play-throw away' style of working. This is targeted at the visualization scientist who will experiment with different forms of visualization in order to achieve the most appropriate view of their data. The tool is written in Java and on start-up connects to an ADVISE middleware service from which it receives a list of available visualization components (modules) that is displayed in a panel within the application window. The user drags modules from the list into a workspace, causing the middleware to start an instance of each module in the components layer on the server. The user is then able to connect the modules together in the workspace; as connections are made or broken in this layer, the tool uses web service calls to communicate these actions to the middleware that



(a) (b)

Figure 3: Example interfaces in ADVISE to visualize the lubrication data. (a) A map is constructed by selecting modules from the repository on the right and connected together in the edit pane on the left. One of the modules has been selected; its widgets are shown in the pane at bottom right. The window at bottom left has been created by the module which is rendering the geometry; it shows the lubrication data as a coloured height map. (b) A tailored interface to the lubrication application.

passes the requests to the components layer on the server. The behaviour of each module is controlled by a collection of parameters, and the user sets values for these via standard widgets such as dials and sliders. As data passes along the pipeline, modules may choose to alter aspects of their parameters (e.g. change a range in which a parameter is valid due to the range of the current data set). These changes are delivered to the user interface through the use of web service notifications.

The output from the tool is a map, described in the skML language (see § 3), which can be saved and restored. In addition to the desktop user interface, we have also built a version that runs as an applet within a browser (see § 4.2 below).

4.1.2. Tailored Application - Scientific End User

Many users will prefer an interface that has been specially tailored to their particular application and which contains only the controls that they require. In addition, they are not interested in the structure of the underlying pipeline. Within ADVISE, visualization developers can create interfaces that hide the pipeline, and present a simple interface to the user that includes the visualization itself together with a set of widgets that control key parameters. The designer tool is used initially to create and save the underlying map, and identify the properties to be exposed in the interface. When the application is run, the map is restored in 'hidden' mode and executes in the same way as any other map. This style also gives the opportunity to express the interface in the context of the application, rather than using visualization terminology. Figure 3 (b) shows the lubrication application presented using this style of interface.

4.1.3. Web Browser Interface - Wide Community of Users

Most users are comfortable with an interface that runs within the Web browser. We have developed an applet version of the designer tool, which can run in this fashion. Thus, in Figure 4 we see a map for the lubrication example being created, working within a popular browser environment.

The Web browser interface becomes arguably still more potent with the tailored application approach, where simple interfaces to visualization web services can be provided. We discuss this further in § 4.2.

4.1.4. Script Interface - Power User

Finally, there will be some users who prefer to access the visualization software more directly through a scripting interface. In ADVISE, we provide a low-level interface to allow experienced users to build pipelines using a small set of commands that operate directly at the visualization components layer. It is also feasible to provide a scripting interface in the client / user interface layer (similar to the various interfaces described above) although this has yet to be implemented.

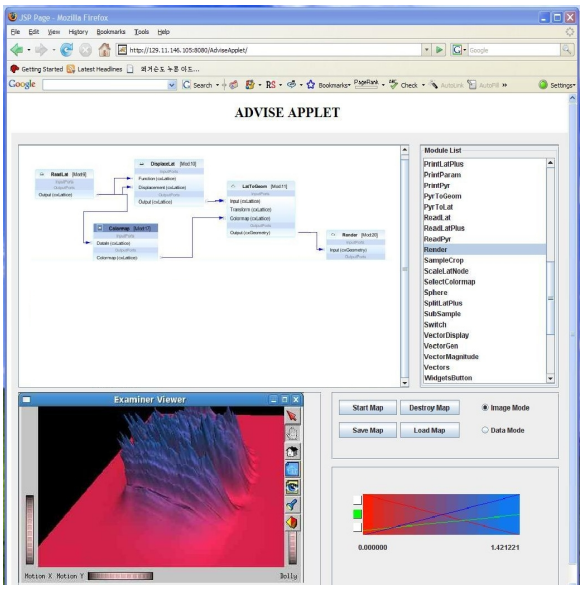


Figure 4: ADVISE web browser interface

4.2. *Configuration flexibility*

The other axis of flexibility (see § 4) relates to the way in which software execution has been distributed.

4.2.1. *Single host*

In the simplest case we can run all three layers on the same host computer, as would frequently be done when using a commercial visualization system today. The visualization service provider will supply a 'standalone' package via web download, for the user to install on their computer.

4.2.2. *Distributed visualization*

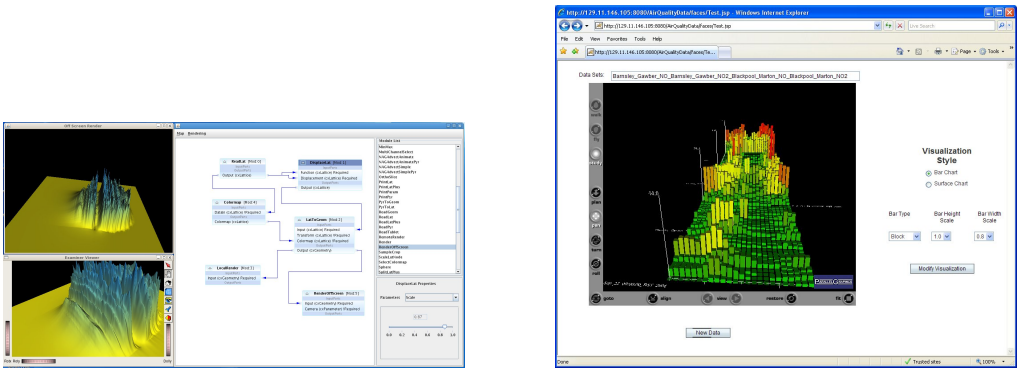
The use of Web services in the ADVISE architecture gives us flexibility in the way the layers are distributed. A typical configuration is to place the visualization components and middleware on a server, and place the client / user interface layer (using any of the interfaces of § 4.1) on the client desktop. To support this separation we have developed a remote rendering component for ADVISE, called ***RemoteRender***, that is capable of utilising remote rendering hardware. It takes geometry from the pipeline and, for a given camera viewpoint, renders an image. This image is delivered to the middleware which subsequently forwards images to clients through Web Services notifications. The middleware supports forwarding requests for camera viewpoint changes so allowing users to interact with the rendering of their data remotely. Alternatively, we also support using local rendering hardware where a module called ***LocalRender*** is started on the user's machine and connected to the map in the same way as remote modules thus allowing geometry to be passed to the local machine for rendering.

Figure 5 (a) shows the lubrication example with both kinds of rendering.

In this scenario, the visualization service provider would supply an organisation with the software to install on a server while client software - including local rendering, if necessary - would be installed on each user's machine.

A special case of distributed visualization is where we use the web browser as a client. In this scenario, the visualization service provider can host all the software, and there is no software distribution required. The lubrication example is not appropriate here; instead, we have developed an air quality visualization service using ADVISE [1]. Figure 5 (b) shows this style of working.

To further support distributed visualization, the ADVISE middleware can be configured in a number of modes (in addition to basic client/server). In all modes the middleware is placed on a client facing host, as described above, to provide a simple single point of contact to access the visualization services that it supports. The placement of the visualization components, however, can be configured to use a number of compute resources that may be hidden



(a) (b)

Figure 5: (a) Lubrication example showing both local rendering (bottom) and remote rendering (top). (b) ADVISE used to provide an air quality visualization service.

from the outside world. In one mode these hosts are allocated to clients as they connect to allow each user their own compute resource. In another, users are allocated multiple resources with the middleware splitting the visualization pipeline by placing different modules on different resources. A further mode using multiple resources, which is still under much development, sees the modules of the pipeline replicated on a number of hosts and a portion of the data fed to each pipeline with the rendered result from each host combined into a single image using a modified version of the RemoteRender module. In this last case, the middleware presents the system as a single pipeline to the user interface, handling the complexities of multiple pipelines internally.

5. Conclusion and Further Work

As visualization continues to evolve as a discipline, there is a growing body of high quality algorithmic software - the intellectual heart of the subject. The goal for the designer of visualization systems is to make this software accessible to the user community, through the variety of interfaces and in the variety of standalone and distributed configurations that exist today. This paper has argued that the flexible framework of ADVISE contributes to this goal. Its three layer model cleanly separates the key aspects of visualization system design - the visualization components layer comprising the algorithmic content; the middleware providing the management; and the client / user interface layer handling the presentation. This model then allows us to deploy a wide variety of visualization services from a single body of software: different configurations of software across hardware resources are supported, and different interfaces are readily provided to cater for a range of users.

In a separate study, to be reported elsewhere, we are exploring the performance implications of the architecture: what does this flexibility cost?

We are carrying out this research within a UK government-funded project to study visual analytics. Our next step is to augment the visualization modules which have already been ported to the system with a new set of modules that contain statistical algorithms. The latter incorporate functionality from the GenStat package, and the incorporation of statistical techniques will allow for the development of improved analysis methods for users' data. Moreover, the flexibility of our framework will allow domain-specific applications to be developed using a variety of delivery mechanisms.

We are also keen to extend the system to the analysis of very large datasets. The inclusion of shared memory is a step in that direction, but more work is needed here.

Finally our use of Web services will enable us to exploit Cloud Computing technologies, such as provided by the Amazon EC2. Indeed we are close to a time when we can expect to see 'visualization as a service': ADVISE can support a scenario whereby the visualization is generated on the cloud and returned to the user through a browser interface.

References

[1] J. Wood, K. Brodlie, J. Seo, D. Duke, J. Walton, A web services architecture for visualization, eScience, IEEE International Conference on 0 (2008) 1–7. doi:http://doi.ieeecomputersociety.org/10.1109/eScience.2008.51.

[2] R. Haber, D. A. McNabb, Visualization idioms: A conceptual model for scientific visualization systems, in: Visualization in Scientific Computing, 1990.

[3] C. Upson, J. Thomas Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. van Dam, The application visualization system: A computational environment for scientific visualization, IEEE Computer Graphics and Applications 9 (4) (1989) 30–42. doi:http://doi.ieeecomputersociety.org/10.1109/38.31462.

[4] R. Haimes, pv3: A distributed system for large-scale unsteady cfd visualization, in: AIAA paper, 1994, pp. 94–0321.

[5] Paraview, http://www.paraview.org/ (2009).

[6] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, J. Wood, Visualization in grid computing environments, Visualization Conference, IEEE 0 (2004) 155–162. doi:http://doi.ieeecomputersociety.org/10.1109/VISUAL.2004.112.

[7] J. J. van Wijk, H. van de Wetering, Cushion treemaps: Visualization of hierarchical information, in: INFOVIS, 1999, pp. 73–78.

[8] L. Wattenberg, The baby name wizard, http://www.babynamewizard.com/voyager (2005).

[9] T. Jankun-Kelly, O. Kreylos, K.-L. Ma, B. Hamann, K. I. Joy, J. Shalf, E. W. Bethel, Deploying web-based visual exploration tools on the grid, IEEE Computer Graphics and Applications 23 (2) (2003) 40–50. doi:http://doi.ieeecomputersociety.org/10.1109/MCG.2003.1185579.

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

[11] H. Wang, K. W. Brodlie, J. W. Handley, J. D. Wood, Service-oriented approach to collaborative visualization, Concurr. Comput. : Pract. Exper. 20 (11) (2008) 1289–1301. doi:http://dx.doi.org/10.1002/cpe.v20:11.

[12] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, P. Li, Taverna: a tool for the composition and enactment of bioinformatics workflows., Bioinformatics 20 (17) (2004) 3045–3054, http://dx.doi.org/10.1093/bioinformatics/bth361. doi:10.1093/bioinformatics/bth361.

[13] A. B. Viegas, M. Wattenberg, F. V. Ham, J. Kriss, M. Mckeon, Many eyes: A site for visualization at internet scale, in: Proceedings of Infovis, 2007.

[14] C. D. Hansen, C. R. Johnson (Eds.), The visualization handbook, Elsevier, Amsterdam [u.a.], 2005.

[15] D. A. Duce, M. Sagar, skml: A markup language for distributed collaborative visualization, in: Theory and Practice of Computer Graphics, Eurographics UK, 2005, pp. 171–178.

[16] C. E. Goodyer, M. Berzins, Parallelization and scalability issues of a multilevel elastohydrodynamic lubrication solver: Research articles, Concurr. Comput. : Pract. Exper. 19 (4) (2007) 369–396. doi:http://dx.doi.org/10.1002/cpe.v19:4.