

Problem Solving Environments: Extending the Rôle of Visualization Systems

Helen Wright¹, Ken Brodlië², Jason Wood², and Jim Procter³

¹ Department of Computer Science, University of Hull,
Hull HU6 7RX, UK

`H.Wright@dcs.hull.ac.uk`

² School of Computer Studies, University of Leeds, Leeds LS2 9JT, UK
{`kwb`, `jason`}@`scs.leeds.ac.uk`

³ now at Research School of Chemistry, Australian National University,
Canberra, ACT 0200, Australia
`jimp@rsc.anu.edu.au`

Abstract. Visualization systems based on the dataflow paradigm are enjoying increasing popularity in the field of scientific computation. Not only do they permit rapid construction of a display application, but they also allow the simulation to be incorporated, giving the scientist the opportunity for computational steering as well. However, if these systems are to realise their full potential as problem solving frameworks, then three key requirements of support for group working, data persistence and usability must be addressed. This paper reviews our prior work on collaborative visualization and data management and reports new developments to improve user interface flexibility. These extensions are then assessed in the context of a unifying, augmented architecture, which in turn indicates scope for future work.

1 Introduction

Advances in computing in the last ten years have brought about a significant change in the modelling and simulation of complex phenomena. Increases in computer processing power, memory, disk and graphics capacity have not only brought about a corresponding increase in the size of problem that can be attempted, but have also brought about a fundamental change in *how* such problems are tackled. Calculations which were formerly run in batch mode with their output scrutinised afterwards can now be monitored whilst in progress using graphical means, or even ‘steered’ by altering their input parameters according to the current visual results. One approach, exemplified in the SCIRun system [1], is to develop a purpose-built computational steering system from scratch, giving greater flexibility at the expense of more development effort. Another is to provide the tools to instrument an existing code and visualize the results (e.g. [2]). Such work has computational steering as its main focus, whilst other projects such as [3] have addressed the interworking of component-like simulation and visualization codes across heterogenous networks.

The application of computational simulation to real world problems thus increasingly depends on the integration of a collection of different tools, utilised by a number of investigators having a variety of different skills. Furthermore, as projects grow in size and complexity, collaboration amongst co-workers who may be geographically separated must also be considered an issue. This, too, is recognised in [2] and has received attention in other forums [4], [5].

In this paper we review the rôle that commercially-available visualization systems may play in computational simulation, concentrating in particular on Modular Visualization Environments (MVEs). We begin by capturing an architectural model of these systems, which is then extended in three key ways.

2 Visualization Architecture and Extensions

MVEs, usefully summarised in [6], offer a variety of techniques for graphical output but without the need to program. Instead, the user selects code blocks, or modules, from a system-provided repository and joins them together using the mouse. Although widely used for post-processing simulation data, these systems are also interesting for computational steering because they allow the scientist to interact with the simulation code itself, either by incorporating it into the environment using an application programmer's interface, or by loosely-coupling the simulation and MVE. These systems implement the Filter, Map and Render pipeline model of visualization proposed by Haber and McNabb [7]. At the Filter stage, data input from disk or direct from some simulation code is sampled if dense or interpolated if sparse. In the Mapping stage, a geometrical representation is constructed, whilst at the Render stage this object is drawn and lit in order to produce the image. Modules only execute when the user varies one of their parameters or new data arrives.

In practice a number of individual modules may contribute to each of these stages and may comprise both serial and parallel codes. Coarse-grained parallelism can be exploited by distributing modules across a number of heterogeneous machines, with the environment handling user authentication and synchronisation of data flow. Alternatively, a computationally intensive code may be run on some remote supercomputer, with the results returned transparently for visualization at the workstation.

Although often the user interacts with the dataflow pipeline directly, systems also provide an end-user abstraction, or 'shrink-wrap' mode. Here, selected parameters of the simulation and visualization can be exported to a separate user interface, whilst the pipeline itself and other parameters remain private to the application developer. Figure 1 depicts a typical scenario, with a simulation code and filter elements running on a remote machine. In spite of their flexibility, MVEs also have their weaknesses. Firstly, although the dataflow pipeline may be distributed across several computers, logically it remains a single system for one user, with its interface, parameters and data flows intrinsically bound to it. Secondly, as the user changes the parameters of the calculation, so the data that flows changes. Data is ephemeral in a standard dataflow system and

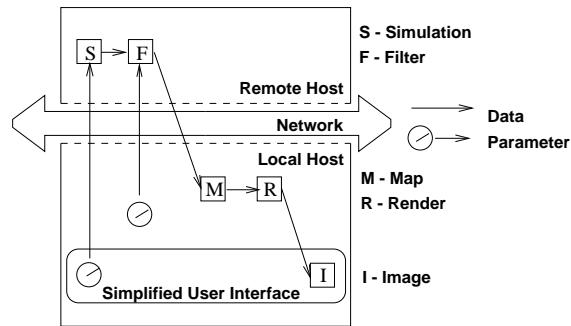


Fig. 1. MVE being used to steer a simulation running remotely

only the current state of the simulation/visualization program is ever accessible. Thirdly, complex *sequences* of actions, as may arise in computational steering, are difficult to achieve in the dataflow paradigm. Nonetheless, the general applicability of these systems and their extensibility, coupled with the infrastructure they provide for process management and distribution, warrants their consideration as generic problem solving frameworks, provided these deficiencies can be addressed.

2.1 Collaborative Working

One approach to collaborative visualization is the COVISA system described by Wood et al [8]. This system supports collaboration by allowing the selective sharing of the data, visualization parameters and pipeline building processes of an MVE. Because the MVE architecture is open, data and parameters can be captured where they flow between modules and distributed to other users. By exploiting the extensibility of the MVE we have created a set of modules, denoted by 'C' in Figure 2, that can pass data and parameters into and out of the environment. These are supported by an external server process, which is invisible to the users, for routing data between collaborators. Sharing data via modules provides a familiar paradigm and avoids the need to learn a new interface. Other modules provided allow a user to join and leave a collaborative session and to participate in collaborative map building.

This flexible approach means that users are free to choose at which points in the pipeline data and parameters are shared. Each team member works with their own copy of the visualization system, sharing data and control parameters as resource limitations and security considerations dictate. For example, in a compute-intensive application which generates a lot of data, only the geometrical representation coming from the mapping stage need be shared in order for co-workers to see the visualization (Figure 2). This configuration also means the simulation code and raw data remain private to the simulation owner, though co-workers can contribute to steering and visualization choices by exporting parameter values to their colleague's environment.

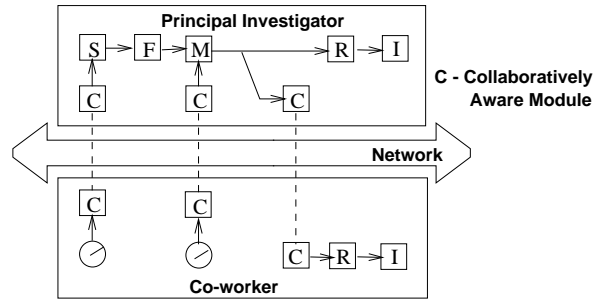


Fig. 2. Typical collaborative configuration, with shared elements reflecting resource and security constraints

COVISA is by no means the only collaborative visualization system available, for example, COVISE [9] offers collaboration by running a single instance of the base visualization system, but with multiple user interfaces, each accessing the whole pipeline. This can be achieved in COVISA by one user running all of the modules in their environment, with the others sharing control of the visualization parameters and receiving the visualized result. Another, HIGHEND [10], has multiple instances of a visualization system, one per user, each with its own user interface. The systems share synchronisation information so consistency is maintained. The equivalent in COVISA would be for each user to run the same set of modules and share control parameters. The architecture adopted here thus allows COVISA to emulate the style of collaboration offered by these other approaches, but with the key difference that users can work independently on some parts of the pipeline and collaborate over others. Whilst our implementation is for IRIS Explorer, a similar architecture has also been used to extend AVS [11].

2.2 Data Persistence

Interactive systems can bring improved insight to a problem, but the significance of any particular result is rarely appreciated at the time it is observed. More usually, hindsight plays a large part in understanding what has gone before. The need to record the progress of an investigation is thus important and has been addressed previously: the GRASPARC project [12] captured both data and parameters in the form of a tree, whose branch points signified a return to some previous simulation attempt; Mulder and van Wijk [13] have linked simulation parameters to graphical objects in the visualization, in order to see how data changes over time.

Data persistence is difficult to achieve in the dataflow model, however. Abram and Treinish [14] have proposed caching data, whilst van Liere et al [15] cite this problem as a motivating factor towards a completely new visualization architecture. Another approach by Wright et al [16] aims to be more flexible than a simple cache, but nonetheless uses a dataflow approach. Based on the GRASPARC tree idea, it includes this into the pipeline by providing an additional

module called HyperScribe. Figure 3 shows the module being used to capture simulation data for a gas turbine study, resulting from steering the ambient temperature, T . The first series of results comprises twelve distinct runs carried out at 600K to 1150K in 50K increments. As each new data set is computed the user stores it on disk by specifying an 'Add' event on the module's graphical user interface (GUI), in the process creating the circles (simulation 'snapshots') which build up the central portion of the tree from left to right. Visualization shows the optimum operating conditions to lie somewhere between 700K and 800K, so additional runs are made to fill in data between these temperatures and the extra snapshots are added to form the side branches. The entire set of results is then retrieved for visualization using 'Recall' events, again on the HyperScribe GUI.

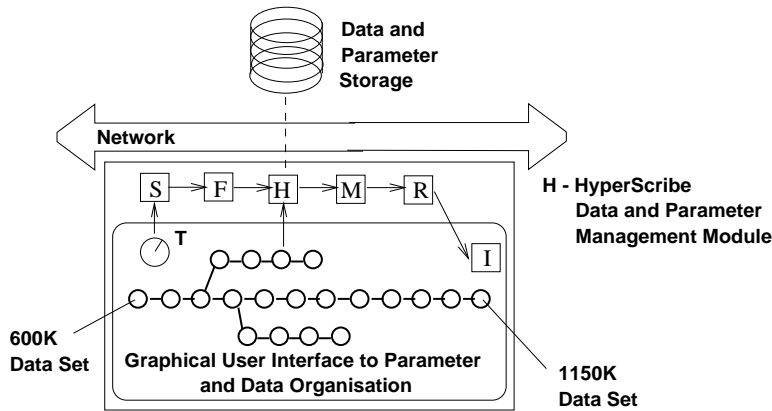


Fig. 3. HyperScribe data management

Figure 4 shows the results of tests in which users were asked to read in 12, 16 or 20 files of results and visualize them, either direct from disk or using Recall events on a tree. On average, using HyperScribe gave a 63% reduction in the time taken but, more importantly, during the 42 task instances observed, 6 errors were noted and all occurred when working with the results files directly. One reason could be a progressive loss of concentration by the user during time-consuming, repetitive tasks, in which case we might expect HyperScribe to increase nearly three-fold the size of problem that can reasonably be tackled. With planned enhancements to the HyperScribe interface, this figure could be improved still further.

2.3 Pipeline Management

Interactive systems often exhibit a repetitive element [17] and computational steering is no exception. For example, consider the following study of a chemical reaction which varies over time:

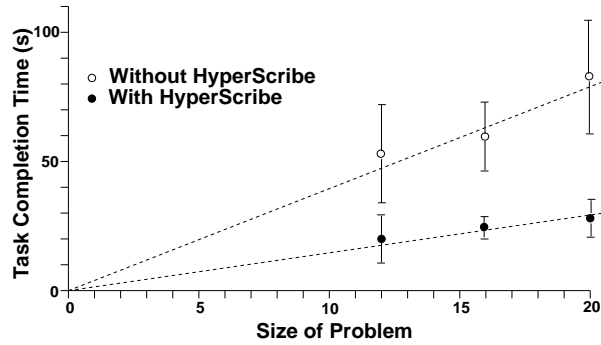


Fig. 4. Time taken to recall and visualize a 12-, 16- and 20-snapshot tree, compared with reading in the equivalent results stored as a set of files

1. Calculate new data starting from $t = 0$ and view the whole time series
2. Crop the viewed data to the portion of interest between $t = t_1$ and $t = t_2$
3. Restart the calculation from $t = t_1$, changing parameters.

In the dataflow model this requires three separate pipeline configurations which may be used repeatedly as parameters such as ambient temperature, computational tolerance and timestep are steered. Connecting and disconnecting modules manually becomes laborious and mistakes may result in data being lost. Additionally the end-user abstraction, or shrink-wrap mode, cannot be used since this hides the pipeline. Our most recent MVE extension, in the form of topology definition and management modules, seeks to support such composite activities by mapping whole configurations to a single menu item or button (Figure 5). Using the standard MVE GUI, the user constructs each pipeline just once and

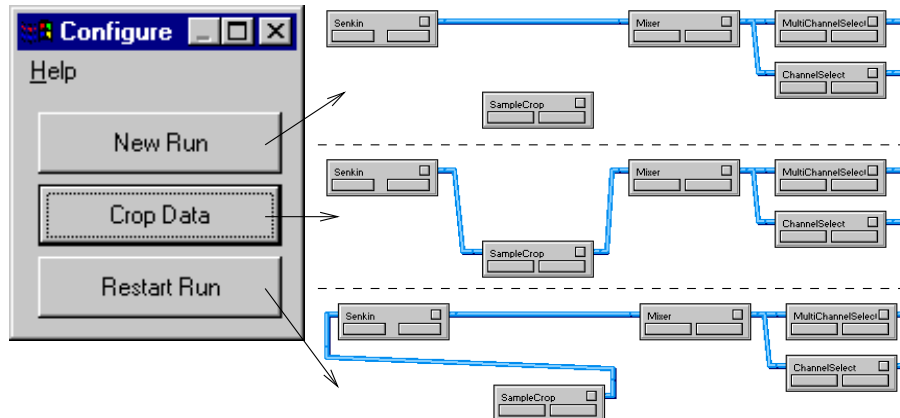


Fig. 5. Separate pipeline configurations map to a single button each

gives it a meaningful name, which is stored in a file by the topology definition module, along with the configuration. Pipeline configurations held in the file can be edited, replaced and added to for flexibility. Thereafter, pressing a particular configuration button causes the topology management module to input to the environment a script of IRIS Explorer command line instructions which changes the pipeline. The particular sequence of instructions needed is determined automatically given the module's knowledge of the current and target configurations. An application developer can choose whether to leave the pipeline available for additional manual interactions, or whether to export the configuration menu to a simplified user interface using shrink-wrap mode.

2.4 Augmented Architecture

Combining all these extensions, we can now propose an augmented architecture, Figure 6, whereby all the elements can work together. For example, Hyper-

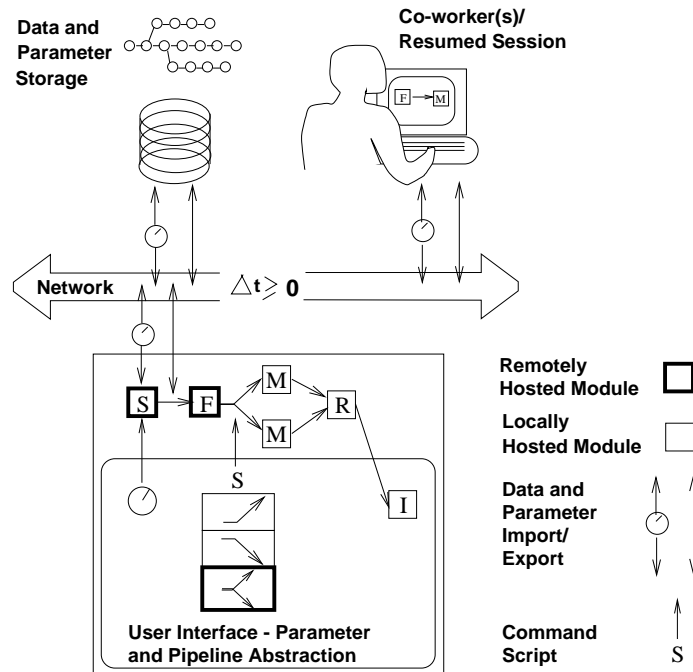


Fig. 6. Augmented MVE, with a simplified user interface giving access both to selected module parameters and different pipeline configurations. Data and parameter import and export allow different steering runs to be recorded and/or passed to co-workers, either synchronously or asynchronously

Scribe's GUI is realised using the geometry data type, and it is a simple matter

to export this to one's co-workers using a collaboratively aware module, in order to cooperate over a single database of steering results. If distant members of a team work asynchronously, anyone can pick up a database of results made earlier by a colleague by running HyperScribe remotely on that person's machine – the only collective decision needed is which particular machine to save the results on.

Our aim throughout is to work within the existing capabilities of the MVE, and this is especially important at the user interface. Here, by providing additional modules to perform the various operations, we avoid placing an additional learning burden on the user. This strategy also means our additional features can become part of a simplified interface constructed for an end-user. For example, a collaborative working application can be set up which hides the communicating modules, or different pipeline configurations can be presented simply as buttons, in a manner analogous to the abstraction of selected parameters already described.

3 Conclusions

In this paper we have reviewed past and current efforts to extend existing Modular Visualization Environments, drawing these together into an augmented architecture which demonstrates the potential for such systems to become Problem Solving Environments. A key principle of our approach has been to utilise existing MVE features, which allows the different enhancements to interwork as required.

Improving the functionality of existing systems holds twofold benefits: firstly, we can build on work already completed to improve these; secondly, extending current systems with an established user base allows early exploitation of the ideas – IRIS Explorer alone, for example, has several hundred users. Perhaps more importantly, improving the usability of current systems allows experience gained on large-scale projects to benefit a range of other, smaller-scale endeavours. This class of applications includes those where the demand for megaflops may not be so great, but where, if a solution is to be found efficiently, the requirement for cooperative work, data and process management tools is just as real as in any large Problem Solving Environment. Thus we envisage in the coming years a growing recognition that the concept of high performance includes not only the processing power applied to some problem, but also the effectiveness of the user in solving it.

Acknowledgements

We particularly thank NAG Ltd and the UK EPSRC for funding; JP thanks Mr and Mrs G Procter for their support whilst at Leeds. Case studies, simulation code and application expertise were generously provided by BG plc, the School of Chemistry and the Department of Fuel and Energy at the University of Leeds,

UK, and Sandia National Laboratories, Livermore, USA. Finally, our thanks to the anonymous reviewers for their valuable suggestions for improvement.

References

1. C Johnson, S G Parker, C Hansen, G L Kindlmann and Y Livnat. Interactive simulation and visualization. *IEEE Computer*, **32**(12):59–65, IEEE Computer Society (1999)
2. J A Kohl and P M Papadopoulos. Computational steering and interactive visualization in distributed applications. ORNL/TM-13299, Oak Ridge National Laboratory, Oak Ridge, USA. (1999)
3. Efficient coupling of parallel applications using parallel application workspace. <http://www/acl/lanl.gov/PAWS/>, Los Alamos National Laboratory, California, USA. (2000)
4. K W Brodlie, D A Duce, J R Gallop and J D Wood. Distributed co-operative visualization, in Eurographics State of the Art Reports, A Augusto de Sousa and R Hopgood (editors), Eurographics98 Conference, pp27 - 50. (1998)
5. K W Brodlie and J D Wood. Volume graphics and the Internet, in M Chen, A E Kaufman and R Yagel (editors), *Volume Graphics*, pp317–331. Springer Verlag, (2000)
6. G Cameron. Modular visualization environments: Past, present and future. *Computer Graphics*, 29(2):3–4, (1995)
7. R B Haber and D A McNabb. Visualization idioms : A conceptual model for scientific visualization systems. In B Shriver G M Nielson and L J Rosenblum, editors, *Visualization in Scientific Computing*, pages 74–93. IEEE, (1990)
8. J D Wood, H Wright and K W Brodlie, Collaborative visualization, Proceedings of Visualization '97, pages 253–259. IEEE, (1997)
9. A Wierse, U Lang and R Ruhle, Architectures of Distributed Visualization Systems and their Enhancements, Eurographics Workshop on Visualization in Scientific Computing, Abingdon (1993)
10. H G Pagendarm and B Walter, A prototype of a cooperative visualization workplace for the aerodynamicist, *Computer Graphics Forum*, **12**(3):C485–C496. Eurographics, (1993)
11. D A Duce, J R Gallop, I J Johnson, K Robinson, C D Seelig and C S Cooper, Distributed Cooperative Visualization - The MANICORAL Approach, Eurographics-UK Chapter Conference, Leeds (1998)
12. K W Brodlie, A Poon, H Wright, L A Brankin, G A Banecki and A M Gay, GRASPARC: A Problem Solving Environment Integrating Computation and Visualization, Proceedings of Visualization '93. IEEE, (1993)
13. J D Mulder and J J van Wijk, 3D computational steering with parametrized graphical objects, Proceedings of Visualization '95, pages 304–311. IEEE, (1995)
14. Greg Abram and Lloyd Treinish. An extended data-flow architecture for data analysis and visualization. *Computer Graphics*, **29**(2):17–21, (1995)
15. R van Liere, J Harkes and W de Leeuw, A distributed blackboard architecture for interactive data visualization, Proceedings of Visualization '98, pages 225–231. IEEE, (1998)
16. H Wright, K W Brodlie and M Brown, The dataflow visualization pipeline as a problem solving environment, Eurographics Workshop on Scientific Visualization '96, pages 267–276. Springer-Verlag, (1996)
17. J Nielsen, Usability engineering. Academic Press, (1993)