

VISUAL BEANS PROJECT

**Role of Component Technologies and Open Implementation Techniques in Adaptable
Cooperative Visualization**

FINAL REPORT

GR/M82011 CCLRC RAL
GR/M81779 University of Lancaster

Christopher Cooper, David Duce
Oxford Brookes University

Julian Gallop, Ian Johnson
Rutherford Appleton Laboratory

Gordon Blair, Geoff Coulson
University of Lancaster

BACKGROUND AND CONTEXT

Computer-assisted visualization is an essential tool in many scientific, engineering and medical applications. Visualization is also a collaborative activity. Computer support for collaboration is increasingly important and offers significant opportunities for fostering new styles of collaboration rather than replacements for existing styles. In the Visual Beans proposal we described our vision of the future in these words “collaborators should be able to join collaborative working sessions in a flexible way, in particular supported by automated dynamic adaptation to a dynamic environment in which processing power, display capabilities and networking resources may alter”.

The project proposal took the view that the realisation of this vision could be addressed by using component technologies and middleware as a framework for the construction of distributed collaborative visualization (DCV) systems. The proposal recognised a number of important trends in middleware which could be exploited and developed: these included reflective facilities such as interception wrappers, explicit bindings, and the increasing role of componentisation in distributed systems.

In the visualization field previous work had often been based on dataflow-based visualization systems; for example, in DCV extensions to the commercial AVS/Express and Iris Explorer systems. RAL developed such an extension to AVS/Express in the EU funded MANICORAL project and Brodlie’s group in Leeds developed an extension to Nag’s Iris Explorer. Other approaches in the literature included CSpray (USA), and COVISE (Germany). A limitation of these approaches was their closed nature, and the need for all participants in a collaborative session to run an identical copy of the base visualization system, thus limiting adaptation to heterogeneous environments and interworking with other systems.

The issues posed by heterogeneous environments have assumed greater prominence during the project through the emergence of relatively high performance mobile devices (PDAs such as the Compaq iPAQ) which combine support for wireless networking with significant (in the context of visualization) CPU power, memory and display capabilities. While such devices may not be powerful enough to compute a visualization they are certainly capable of displaying pre-computed visualizations.

World-wide programmes in e-Science and Grid technology also emerged during the course of the project. Due attention has been paid in the project to its relationship to these programmes.

KEY ADVANCES

Introduction

Throughout its course, the project took a very practical approach to its investigation of the above areas. Regular 3 monthly meetings were held, involving all the participants, and a number of exchanges and extended visits between institutions were arranged. These arrangements helped ensure a highly coordinated approach at the level of detailed design and implementation. Project steer was also contributed by BT Labs in the early stages of the work.

The initial stage of the project established a common understanding of the issues in DCVs, component technology and middleware. These early meetings addressed difficult questions including an appropriate granularity for componentisation and the choice of a baseline visualization system. As the project developed, an overall architecture emerged, coupled with a representative set of visualization-related components, and scenarios from which this set was derived. A significant proportion of the project was concerned with construction of the middleware platform and components. The regular project meetings, supported by email and telephone contact, resulted in a highly interactive and integrated project. The development work was divided according to expertise, with Lancaster taking responsibility for the middleware platform and RAL for DCV componentisation. The key issue of the design of interfaces and component wrappers was undertaken as a joint activity.

During the course of the project the two PIs at RAL (Duce and Cooper) moved to new posts: Duce to a full-time chair at Oxford Brookes University, and Cooper to a part-time chair at Brookes and a part-time post as network strategist at UKERNA. But it has proved to be perfectly practical for Duce and Cooper to continue their PI roles from their new posts at Oxford Brookes. The two other RAL staff engaged in the work (Gallop and Johnson) remained at RAL and the research grant continued to be held by RAL.

Overall architecture

The architecture adopted was based on state-of the-art component-based middleware concepts. More specifically, the project developed and extended a middleware platform called TOAST that was initially a product of a PhD project at Lancaster. TOAST abstracts over the underlying heterogeneous distributed environment, and facilitates the task of designing, programming, and managing distributed applications by providing a simple, consistent and integrated component-based programming model. The current version runs on Windows based PCs and iPAQ hand held devices.

In more detail, TOAST extends a standard CORBA-based environment to provide independently deployable components, explicit bindings (supporting both conventional request/ reply and stream-based communication), and support for the reconfiguration of component topologies at run-time (e.g. to support adaptation). Independently deployable components can support any number of request/reply and stream interfaces. Stream interfaces support data-types such as visualisation data, animated graphics, or audio/ video streams. To connect or 'bind' interfaces, the TOAST programmer explicitly creates a so-called binding component which offers an interface through which the programmer can subsequently manage and exert dynamic control over the binding. For example, methods can be called to adjust the quality of service of the binding—perhaps to vary the latency and throughput achievable, or to start and stop the flow of data. Finally, TOAST component topologies are inherently reconfigurable: components can be dynamically added, removed, and replaced at run-time. An explicit 'component graph' representation of the current topology is provided as a basis for such reconfiguration. Furthermore, the programmer can call a standard method on any component to cause it to migrate to a designated target location. This migration is accomplished as transparently as possible to the migrated component and any bindings it is currently involved in.

Incorporation of visualization in TOAST

The project wrestled with three issues: the level at which sharing should take place (mentioned in the Background section), the granularity and choice of components, and the programming model in which components would be realized. The first issue was the most straightforward. Based on earlier work in the MANICORAL project our judgment was that sharing should not be restricted to any one level in the system, but should rather be supported at a range of levels from application data to rendered image. This is reflected in the choice of components in Visual Beans.

The second and third issues occupied some time. Our initial view was that components would be fairly fine-grained, reflecting, for example, the notion of a module in the modular visualization environment. Over time our thinking evolved and we finally came to the view that components should be more coarse-grained. In particular, we found that coarse-grained components provided a better basis for adaptation experiments. The particular set of components that was implemented is discussed below.

Modular Visualization Environments are based on a dataflow programming model. Modules "fire" under the control of a central executive when input data become available. Other visualization systems use different models, for example the publish/subscribe design pattern, in which objects execute in response to notifications. The programming model eventually chosen for Visual Beans was based on the Model-View-Controller paradigm. Visualization components were constructed using the VisAD visualization system. A number of candidate systems were considered including the commercial AVS/Express and Iris Explorer systems and Open DX. VisAD has an internal structure based on publish/subscribe which is not naturally sympathetic to the dataflow/stream model that underlies TOAST. The choice of VisAD was influenced by a number of factors including availability of source code (VisAD is written almost entirely in Java) and a wish to explore how a system that is not naturally a dataflow system could be wrapped as dataflow components.

The key achievements in this area were the stance taken on the granularity of components; the development and exploitation of TOAST's flexible reconfiguration facilities—particularly component substitution and component migration; and the definition of interfaces and wrappers to define key visualization components implemented in VisAD.

Experiments

A number of experimental systems were constructed as demonstrators and proof-of-concept implementations of the TOAST framework and visualization components.

The first system we constructed consisted of a single visualization component and a separate slider control component. The idea was that the slider could be used to select a particular image from a sequence of images,

visualizing particular time steps in a data set. Although the application was not particularly interesting of itself, the achievement was significant in that it demonstrated that a visualization application constructed using VisAD could be wrapped as a TOAST component and its control interface could be bound to a separate slider component.

The second demonstrator consisted of a more complex collection of visualization components: data source, visualization engine, greedy/lazy slider component, viewer component, splitter component and web server. The visualization components were constructed and wrapped according to the model-view-controller paradigm. The slider component was implemented in C++ rather than Java (demonstrating language heterogeneity) and provided the user with a choice of greedy or lazy slider, the former providing a stream of updated position values as the slider was moved by the user and the latter a single value when the user had determined a new slider value. The significance of this component was that the visualization engine would attempt to recompute the visualization for each new data value received. In the case of the greedy slider the visualization engine component would have difficulty keeping up with the stream of input values unless a powerful processing engine were available. Experiments were then carried out that involved the transmission of audio streams (to facilitate informal collaboration) through TOAST alongside the visualization setup. In this extended scenario, changing to a lazy slider was necessary in order to avoid degrading the audio performance to an unacceptable level. The key point here was the demonstration of adaptation mechanisms. Although adaptation was performed under user control in this demonstration, it could equally well have been performed under the control of a monitoring component that implemented some adaptation policy, using the same programming interfaces as embedded in the slider component.

Having achieved this second demonstration, we developed our scenario to accommodate component migration. This was done by introducing an additional component, a visualization engine manager which acted as a proxy for the visualization, so that data source, control and output viewer components were bound to the visualization engine manager rather than to the visualization engine directly. A separate control interface on the visualization engine manager component could cause the visualization engine to migrate to a different host while maintaining the necessary bindings between the visualization engine manager and visualization engine. A key problem overcome here was the maintenance of the state of the visualization engine component between old and new locations.

Having demonstrated component migration, two additional components were developed, a data filter component and a simple JPEG filter. The data filter could be inserted between a data source and a visualization engine. The effect is to coarsen the grid on which the data are defined. This can be used to reduce the volume of data supplied to the visualization engine. The JPEG filter component could be inserted between the visualization engine and a splitter component, or splitter component and viewer, in order to reduce the volume of data transmitted to the viewer by compressing the JPEG images at a higher factor. With these additional components we were able to demonstrate scenarios involving limited computing resources (thus necessitating reduction of the data volume supplied to the visualization engine) and limited display resources, for example we experimented with the integration of an iPAQ PDA connected by a low bandwidth link (necessitating reduction of the volume of data to be displayed for selected participants).

The key achievements of the demonstrations were proof-of-concept implementations of the ideas developed in the project and demonstration of the flexibility of the components and interfaces in terms of realizing a variety of application scenarios through component composition.

Application to e-Science

The world-wide programmes of e-Science and Grid technology aim to create virtual organisations, encouraging collaboration among researchers and bringing together dispersed resources to bear on a single problem. The component-based approach developed in this project has definite potential to enhance this collaboration.

As an example, instead of being lodged in known datasets, data may be generated by equipment at high data rates--examples such as satellite ground stations and biological scans are well known. Data may also be produced by a long running computation on the Grid, such as a climate model or whole aircraft simulation. A group of scientists at different locations may wish to examine these datasets simultaneously and speak to each other about their findings. The component approach enables the visualization software to communicate with a separate component which is responsible for reading the data and transforming it to a form that the visualization software can read. Typically only one of the science group has a particular visualization system available. Other participants initiate their own copies of the control and display components which enable them to engage fully in the interaction. Other components handle audio and video communication whether on the participants' host

machines or on an Access Grid node. The network and computational environment will vary from one participant to another and from one occasion to another. So enabling parameter variation and component migration will allow communication to be more robust and effective.

Software

The following visualization generic components have been constructed. Note that, because of the flexible TOAST architecture, these can be easily combined in many different ways to satisfy diverse application requirements

- *Visualization engine.* Built using VisAD, this component takes input from a data source and delivers a rendered image. For demonstration purposes, we use a component tailored to visualizing the data sets used in a demonstration in the MANICORAL project. These are netCDF data files of bathymetric data of a region of the central Mediterranean. The component has a control interface, through which the range of data values to be presented can be viewed. For demonstration purposes, the output from the visualization engine is a sequence of JPEG images. A new image is delivered whenever a new control data value is supplied.
- *Splitter.* This takes the output from a visualization engine and transmits it to viewer components bound to the splitter and to a simple HTTP web server that pushes new images to its clients.
- *Viewer.* A simple JPEG viewer to view the output from the splitter. A version of this component was also developed for the Compaq iPAQ.
- *Slider.* A component written in C++ that can be bound to the visualization engine or arbitration component to control the visualization. This component provides both a greedy and a lazy slider. The greedy slider delivers a stream of values as the slider is moved. The lazy slider delivers a single value when movement ceases (in practice when the user releases the mouse button). A button is provided to toggle between lazy and greedy versions. The component was written in C++ to illustrate how components heterogeneous with respect to implementation language could interwork in the TOAST framework.
- *Data simplification filter.* A component which can be inserted between a data source and a visualization engine. The effect is to coarsen the grid on which the data are defined. This is used to reduce the volume of data supplied to the visualization engine.
- *JPEG filter.* A component which can be inserted between the visualization engine and a splitter component, or splitter component and viewer, in order to reduce the volume of data transmitted to the viewer by compressing the JPEG images at a higher factor.

Components to support continuous media were also made available. These include components to filter audio channels to require less bandwidth.

Crucially, the software produced by the project (both the generic platform and the specific demo scenarios) has been consolidated and made available to the research community (available on request from the investigators). We see this as a key outcome for a project that has been highly focused on inter-participant collaboration at the level of detailed design and implementation work.

Research impact and benefits to society

In the area of scientific visualisation, the project has shown how components can form the basis for a flexible approach to distributed cooperative visualization which supports adaptation to heterogeneous networking and processing situations. A set of components was successfully implemented and configurability and adaptation mediated by the middleware platform have been demonstrated. The project has significantly influenced the direction of future distributed cooperative visualization research at RAL and Oxford Brookes primarily as a result of the set of components and approach to wrapping developed in the project.

It is also of relevance that the project, particularly in its early stages, collaborated with Bill Hibbard, the developer of VisAD, and contributed perspectives and development directions to the VisAD community.

As well as its obvious impact in the specific area of scientific visualization, the experience of the project has significantly impacted the direction of middleware research at Lancaster. For example, our experiences fed in directly to the development of the OpenCOM component model which is being used at Lancaster to support a wide range of application areas from ad-hoc interaction in mobile environments to programmable networking.

More broadly, the project produced well-founded general recommendations for next generation middleware solutions. This includes the development of web-services based middleware for e-Science (e.g. through the

Open Grid Services Architecture). In particular, the project has proposed that future middleware should support the following: i) the modelling of communications as first class components—this permits communication components to be deployed, controlled and reconfigured in an analogous way to application components and for QoS to be controlled dynamically, ii) the equitable treatment of all media types—this enables the communication of diverse data-types (e.g. integers and audio streams) to be treated in a consistent way and with a uniform approach to QoS management, and iii) the need for openness in platforms to facilitate reconfiguration—this is necessary to support the wide range of adaptive behaviour found necessary in scientific visualisation (as well as other demanding application areas).

In addition to the software and papers already produced, we are currently involved in the production of two definitive project outcome papers for publication in quality archival journals: a paper from RAL/OBU with a visualisation focus; and one from Lancaster with a middleware support focus.

FURTHER RESEARCH AND DISSEMINATION ACTIVITIES

Cooper and Duce teach on the High Speed Networking and Distributed Systems MSc course at Brookes. The Visual Beans project has been influential in their teaching and has provided illustrations and demonstrations of the concepts taught and motivation for a new module in Multi-service Networks.

Results of the project are also disseminated in the Advanced Distributed Systems module of Lancaster's MSc in Distributed Interactive Systems.

A follow on project is being planned to further develop the very fruitful collaboration between RAL and Lancaster (and now Oxford Brookes University). In particular, we hope to develop our approach in certain strategic directions and to apply/ evaluate it on a wider scale in the area of e-Science and Grid computing. In addition, the results of the project are being disseminated to the Grid community through Coulson's participation in the UK e-Science Architecture Task Force.