Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

CSE: a modular architecture for computational steering

R. van Liere, and J.J. van Wijk

Computer Science/Department of Interactive Systems

CS-R9615 1996

Report CS-R9615
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

CSE
A Modular Architecture for Computational Steering

Robert van Liere

Centrum voor Wiskunde en Informatica

P.O. Box 94079, 1090 GB Amsterdam

The Netherlands

E-mail: robertl@cwi.nl

Jarke J. van Wijk

Netherlands Energy Research Foundation ECN — *Centrum voor Wiskunde en Informatica*

P.O. Box 1, 1755 ZG Petten — *P.O. Box 94079, 1090 GB Amsterdam*

The Netherlands — *The Netherlands*

E-mail: vanwijk@ecn.nl

Abstract

Computational steering is the ultimate goal of interactive simulation. Steering enables users to supervise and dynamically control the computation of an ongoing simulation. We describe CSE : a modular architecture for a computational steering environment. The kernel of the architecture is designed to be very simple, flexible and minimalistic. All higher level system functionality is pushed into modular components outside of the kernel, resulting in a rich and powerful environment. For these modular components (called satellites) a uniform user interface metaphor for users, based on a tray of cards, has been used. The card tray metaphor is very simple to understand and provides users with a simple mechanism to organize and retrieve the tools. Several applications of the environment are shown.

CR Subject Classification (1991): I.3.4, I.3.6

Keywords & Phrases: computational steering, scientific visualization, three-dimensional graphics and interaction

Note: This paper was presented at the 7th Eurographics Workshop on Visualization in Scientific Computing, April 23-25, Prague, 1996

1. Introduction.

Computational steering is the ultimate goal of interactive simulation in which users have direct control over the parameters of a simulation and are able to supervise and dynamically control the computational process. The benefits of computational steering are well known. For example, according to Marshell et al. [1] : “Interaction with the computational model and the resulting graphics display is fundamental in scientific visualization. Steering enhances productivity by greatly reducing the time between changes to model parameters and the viewing of the results”.

There are three reasons why software tools for computational steering are more demanding than those found in traditional scientific visualization environments. First, in traditional visualization systems, a visualization expert can first prepare a visualization, which then is analyzed by the scientific user. Inherent to computational steering is that the user will be an active participant in the visualization loop. Furthermore, due to the exploratory nature of steering, these tools will be used iteratively. Hence, tools must be programmable and modifiable during the analysis cycle, preferably by the end users. Second, end users are usually non-professional programmers who have neither the time nor the training to create a new interface. Therefore, the specification and usage of tools must be very simple and hide the underlying complexity of the system. Third, because of the inherent complexity of large scale simulations, effective usage of distributed computing resources must be guaranteed.

In [2] we introduced CSE , an Computational Steering Environment that encourages exploratory investigation by the researcher of an ongoing simulation. The kernel of CSE is designed to be very simple, flexible and minimalistic. Although we were able to demonstrate a number of applications, the CSE required substantial knowledge and expertise to use. In particular, it was tedious to develop individual tools and use these tools in concert. In this paper we focus on how original design principles of the CSE are used to overcome these difficulties. The governing concept is the modularity of tools. Instead of extending the CSE kernel, we extend the environment by defining new tools that build upon the basic CSE primitives. These tools provide functionality that is usually hard wired in the kernel of other environments.

In section 1.1 we summarize the kernel and the underlying concepts of the CSE . In section 2 we present the underlying principles of the visualization tools – called satellites – and discuss the life cycle of a satellite. In section 3, a standard user interface metaphor based on a card tray is introduced. Sections 4 and 5 give two examples of how CSE ’s system functionality is extended. In section 4 the trigger manager satellite is introduced. Trigger management allows users to define the control of satellites. In section 5 the transport tuple satellite is introduced. This satellite optimizes data transport between two distributed databases. In section 6 we give an example of how all pieces of the CSE fit together. Finally, in section 7 we compare the CSE with other extensible visualization environments.

1.1 Background.

An overview of the architecture of the environment is shown in figure 1.

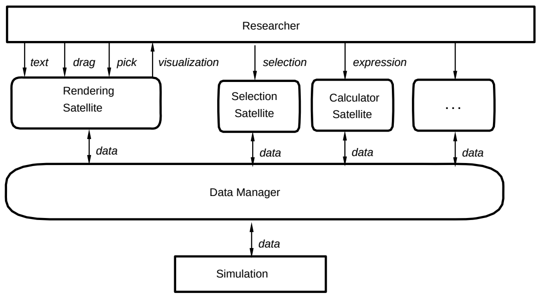


Figure 1: The CSE architecture

The architecture of the environment is centered around a data manager that acts as a blackboard for communicating values, and satellites that produce and visualize data. The purpose of the data manager is twofold. First, it manages a database of variables. Satellites can create, open, close, read, and write variables. For each variable the data manager stores a name, type, and value. Variables can be scalars or arrays, in which case the number of dimensions and size of the dimensions is also stored. Array sizes can change dynamically during the lifetime of the variable. Second, the data manager acts as an event notification manager. Satellites can subscribe to events that represent state changes in the data manager. Whenever such an event occurs the satellite will receive an event from the data manager. For example, if a satellite subscribes to mutation events on a particular variable, the data manager will send a notification to that satellite whenever the value of the variable is mutated.

The foremost satellite is the PGO editor, an interactive graphics editing tool, [3]. The central concept for the graphics editor is the Parametrized Graphics Object (PGO) : an interface is built up from graphics objects whose properties are functions of data in the data manager. Users sketch an interface and bind the graphics objects to variables by parameterizing geometry and attributes with data in the data manager. Simulations may drive the interface by mutating the data bound to the graphics objects. Similarly, users may drive the simulation by interacting with graphics objects. Hence, a two-way communication between graphics and data in the simulation is supported. Since the PGO satellite is an interactive graphics editor, users may incrementally define the interface or change bindings, thus encouraging exploration of the simulation.

The design of the CSE kernel was driven by the following underlying concepts :

- The exclusive use of *low-level primitives*. The CSE kernel uses a simple data model and graphics objects. The interfaces to these are familiar to the satellite developer and user : a UNIX-like I/O library is the API to the data manager and a MacDraw-like editor for the graphics.
- *No higher level semantics* are defined for data and graphics. For example, the data manager

provides no support for defining and maintaining data dependencies between variables. As a result, the environment is general and flexible.

- All operations in both the data manager and the graphics editor are based entirely on *data*. Dragging, picking and text input are translated into mutations of data.
- Satellites rely on *late binding* of variable names. Name matching is used to bind names in a satellite specification to named variables in the data manager. As a result of late binding, it is possible to incrementally define new visualizations of the data output by the simulation, while the simulation continues to run.

We were able to demonstrate a number of applications using the CSE kernel. However, developing satellites was a tedious task. Developers needed to implement all aspects of the interface to the satellite, including the interoperability and user interface, from the low level primitives. Moreover, satellite usage was not straightforward. For example, there was no support for combining satellites into a network of cooperating tools, and each satellite had a different style of interface. In the next section we describe a standard satellite framework that was defined to overcome these problems. The framework defines the behavior of an individual satellite and how it interfaces with its environment. In section 3 we describe the standard user interface for the satellites.

2. Satellite Framework.

An abstract satellite is shown in figure 2. Basically, it consists of an operator that transforms input data into output data. Control determines when this operation has to be carried out, or, in other words, when a satellite is triggered. By defining control externally, instead of using a fixed, built-in control-strategy, a wide variety of cooperation styles between satellites can be realized. The actual definition of the operation is defined by an additional set of parameters. Parameters are manipulated through the satellites user interface: via predefined widgets or by interactions with geometry within the PGO editor.

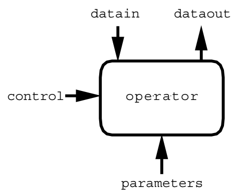


Figure 2: Interfaces to an abstract satellite.

The following phases in the life cycle of a satellite can be distinguished:

- *satellite development.* A satellite developer will design and implement an operator which may or may not be parameterized. The operator is packaged into a satellite.

  Examples of operators are a slicer (selection of data), a calculator (calculation of derived data), and the PGO editor (the visualization and user input of data).

- *edit mode.* The user specifies a parameterization of the operator. This is done by entering names for each parameter of the operator.

  Examples of parameterizations are for the slicer the name of the input variable, the name of the output variable, and the names or values of the slice bounds; for the calculator a mathematical expression; and for the PGO editor a set of graphics objects, parametrized to names of variables.

- *run mode.* In run mode the satellite will bind parameter names to values in the data manager. Name matching is used to bind parameter names to named variables in the data manager. Each triggering of the satellite will result in the re-evaluation of the operator with new input data and the effected output values will be written to the data manager.

  In run mode, the slice operator will be re-evaluated and the output wil be written whenever the input variable or a slice bound is mutated. For the calculator, if a name in the right hand side of the expression is mutated the left hand side is re-evaluated and written. Finally, when a name in the drawing of the PGO editor is mutated, either by changes in the data manager or by interaction on a graphics object, the drawing will be re-rendered.

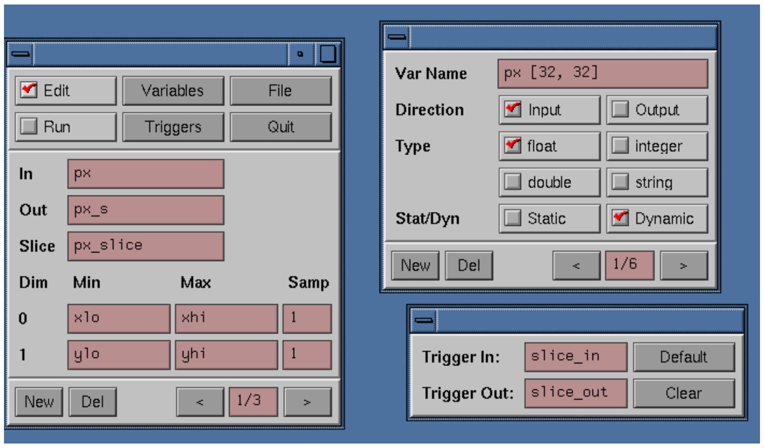
Users will typically iterate a number of times between edit and run mode.

Development and usage of the satellite is simplified through standardization. A satellite development environment is offered, that includes high level libraries and tools that hide the underlying complexities of the satellite's interface. In addition to the development environment, a standard user interface metaphor to a satellite is provided. Standardization on the user interface of the satellite reduces the learning time to operate a satellite.

3. The Card Tray as a User Interface Metaphor.

All satellites in the CSE adhere to a simple user interface metaphor. The metaphor is a tray of cards, with a browsing mechanism to iterate through the cards. Each tray implements a class of operations and each card represents a particular parameterization of the operation. On the left side of figure 3 the user interface for the slicing satellite is shown. It consists of three panels, of which the top and bottom panel are the same for all satellites. The top panel is responsible for the connection administration with the data manager. Every satellite contains a variable browser and trigger editor. The right side of figure 3 shows the popup panels for the variable browser and trigger editor. The variable browser can be used to define or inspect properties of variables that belong to the satellite. The trigger editor is used to specify variable names that will control the satellite. By default a unique variable name will be generated, but users can change this to any name. The details of triggering and trigger names will be explained in sections 4. The bottom panel of the card tray is responsible for the card administration. In edit mode, users can add or delete operators by creating or destroying cards. In run mode, users can browse through the tray and pull cards out of the tray.

The middle panel contains the operator specific user interface. In figure 3 the user interface consists of specifying an input and output variable and a slice name. The slice name itself is parameterized with four variable names and two constants.



[✓ Edit] Run

In: px
Out: px_s
Slice: px_slice

Dim	Min	Max	Samp
0	xlo	xhi	1
1	ylo	yhi	1

1/3

Var Name: px [32, 32]
Direction: [✓ Input] Output
Type: [✓ float] integer double string
Stat/Dyn: Static [✓ Dynamic]

1/6

Trigger In: slice_in
Trigger Out: slice_out

Figure 3: The slicer card tray with variable and trigger browser.

There are a number of advantages in choosing a simple user interface metaphor :

- Since the interface of the satellite is standardized, the user interface to this functionality is the same for all satellites. Uniform variable and trigger editors are generated giving powerful browsing facilities for names local to each satellite. These editors provide the functionality needed to interface the satellite to the rest of environment. Satellite developers need only to supply the functionality of the operator itself.
- The card tray administration is also standardized. The user interface to this functionality is the same for all satellites and is generated automatically.
- Users have a standardized way of interacting with the functionality provided by the card tray. Only the user interface to the operator must be learned. This facilitates the user's task of learning to use new satellites.

- The use of card trays reduces clutter of the screen. Push and pop functions allow for selective control on the number of simultaneously visible cards. The card tray metaphor is scalable in the number of cards in the tray. Efficient card browsing facilities can help locate individual cards within the tray.

The CSE contains a large collection of general purpose satellites. For example, *dmpgo2D* and *dmpgo3D* are the general purpose graphics editors, *dmslice* allows data selections, *dmannot* is a generalized annotation satellite, *dmcalc* is a calculator for scalar and array values, *dmtrans* is a fourier transformation satellite, *dmtimer* provides a general purpose clock, *dmscheme* is a satellite that interprets Scheme scripts, and *dmlog* logs a history of values. The development time of these satellites was greatly reduced by the standard framework provided.

4. Trigger Manager Satellite.

Satellites cooperate via the basic input/output mechanisms that are provided by the data manager for variables. Writing to a variable will cause an event to be sent to all satellites subscribed to that variable. This mechanism is used in two ways. First, the user can specify that only if one particular variable, the *input trigger variable* is changed, the operator has to be re-evaluated. The action of operator re-evaluation is called *triggering*. Second, if no such trigger variable is specified, then upon each mutation of any input variable the output variables are re-evaluated. The satellite will subscribe to all its input variables, and every mutation will cause the satellite to re-evaluate the operator.

Further, the user can also specify an *output trigger variable*. This variable is written to each time the operator has been re-evaluated, and can be used to link the control flow for satellites.

Using mutations on data to trigger satellites provides tremendous flexibility. However, this flexibility also introduces additional complexity. Users must provide distinct output trigger names of the producing satellite which, in turn, must match the input name of the consuming satellite. This is not a problem when using a few satellites, but becomes unmanageable when many satellites are involved.

A trigger manager satellite, *dmTM,* has been developed to simplify the definition of trigger variables. The *dmTM* satellite allows users to define triggers by linking two named variables from independent satellites together. When one variable is written, the trigger manager will copy its value to the second variable. The effect is that the satellites owning the second variable will get a mutation event from the data manager. Notice that copying can be potentially inefficient when applied to large data values. In practice, however, only scalar variables will be used as triggers.

Linking two variables defines a data dependency between these two variables. Linking a number of variables results in an undirected graph, which we call the *trigger graph*. Users may build and edit the trigger graph whenever satellites are connected to the data manager. The task of *dmTM* is to manage the trigger graph.

In addition to managing triggers, *dmTM* is used to monitor the flow of data between satellites. This is done by recording the satellite that has written to a variable, resulting in a directed flow dependency graph of write operations on a variable.

The user interface of the trigger manager satellite can be implemented in many ways. However, since the CSE already provides a general purpose graphics editor, the user interface of *dmTM* is implemented as a card in the PGO satellite. Figure 4 is a snapshot of the satellite configuration of the smog prediction model discussed in section 6. The nodes of the graph represent the satellites connected to the data manager. The blue edges indicate data flow dependency. A blue arrow indicates a directed data dependency. Green edges indicate the trigger graph. The trigger graph can be edited at any time. The panel on the top right provides additional variable information of a selected satellite. The panel on the bottom is the interface to the trigger graph editor.

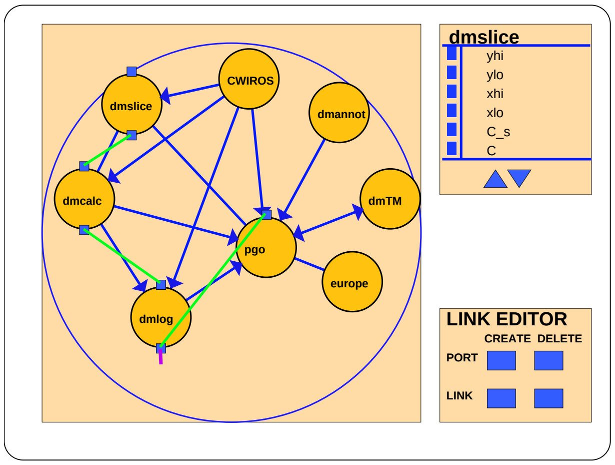


Figure 4: PGO interface to the trigger manager satellite

There are a number of advantages in having a satellite, rather than the kernel, manage the trigger and dependency graphs :

- Since the satellite makes use of the API to read/write and query with the data manager, it implies that synchronization is not intrinsically defined within the kernel of the CSE . Alternative synchronization schemes can be realized by replacing the trigger manager satellite.
- The visual representation and interactions with the underlying data dependency and trigger graph is a card in the PGO editor. The user interface to the graph can be modified at any time.

As an illustration of the functionality provided by *dmTM*, consider the three cases illustrated in figure 5. This typical satellite configuration consists of a simulation satellite, a data mapper satellite and the PGO editor. The time dependent simulation dumps its output to the data manager after every time step. In the configuration on the left, the simulation will run asynchronously with the PGO editor. Data may not be visualized, as the simulation may be dumping data at a higher rate than the mapper or PGO can consume. In the configuration in the middle, the simulation will run synchronously with the PGO editor. The PGO editor will trigger the simulation after it renders one frame. In the configuration on the right, the satellite *dmbutton* will trigger the simulation. *dmbutton* is triggered manually.

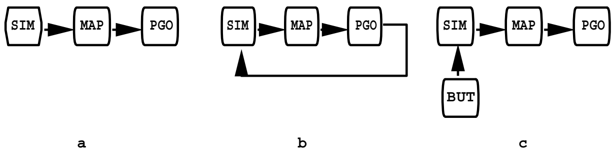


Figure 5: Three different synchronization configurations.

By editing trigger graph, the user can switch between the three configurations while the simulation is running. When the simulation is in a non-interesting state, the user may wish to run the simulation asynchronously. When the simulation is in a semi-interesting state, the user may wish to run synchronously. Finally, when the simulation is in a critical state and each time step requires careful study, the user may wish to trigger the simulation manually.

5. Transport Tuple Satellite.

Due to the quantity of data, efficient data transport is crucial in a distributed data visualization environment. In the case of a centralized data manager, data transport between two satellites can be inefficient if these satellites reside on one computing node and the data manager resides on a different node. Data will be written from one satellite to the data manager and will be read by the other, all via the network. Obviously, for small data sets this is no problem, but for large data sets the network can be a severe bottle neck.

The CSE approach to solve this problem was to develop a satellite that can control data transport between data managers. The satellite that manages this functionality is called the transport tuple satellite, *dmTT*.

The basic idea of *dmTT* is to transport a tuple of variables between two data managers. Each *dmTT* card parameterization contains an input tuple and an output tuple. Each tuple is a description of a set of named variables, denoted as $t_{in} =< v_{1_{in}},\ v_{2_{in}}, ..., v_{n_{in}} >$. The properties of each variable $v_{i_{in}}$ in the input tuple must match the corresponding properties of the output variable $v_{i_{out}}$. When *dmTT* is triggered it will copy the input tuples to the output tuples. Tuples may optionally have a name.

Figure 6 illustrates the transport tuple satellite. To the left is one data manager that manages named variables of all its connecting satellites. To the right is a second data manager. Data managers reside on different hosts. The satellites themselves may or may not execute on these hosts. The transport tuple satellite is responsible to pass data between managers. Data managers do not know of each others existence.

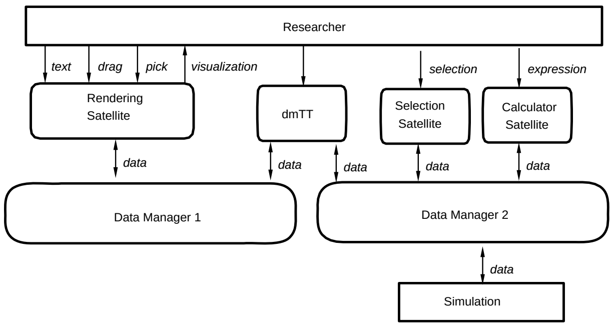


Figure 6: The transport tuple satellite.

There are three options for the triggering of *dmTT*. First, *dmTT* may have one global trigger which, when fired, will copy all input tuples to the output tuples. Second, each input tuple can be triggered through its name, resulting in the copying of one single tuple. Finally, each variable can be triggered through its name, resulting in the copying of one single variable. These three triggering cases ensures that the granularity of data transport can be controlled.

There are a number of advantages of having *dmTT* manage data transport between compute resources :

- Data managers can run on the same machine as the satellites which are connected to it. Communication between satellites local to one machine is substantially faster than over a network. Hence, computing resources can be utilized more effectively.
- Data movement between data managers can be controlled by a satellite. The amount of data to be moved and the time the movement is to take place can be parameterized within *dmTT*.
- The functionality of the data manager can stay simple. No additional support for efficient data transport is necessary in the CSE kernel.

As an example of *dmTT*'s functionality, consider a configuration in which the simulation is running on one machine (say *mach A*) and the slicer and the PGO on a second machine (say *mach B*). The slicer takes its input data from the simulation and gets its slice bounds from a user defined region in the PGO. We consider two extreme cases. First, the simulation generates a large quantity of data at a very fast rate, but the user is only interested in a very small portion of this data. Second, the simulation does not generate data that frequently, but the slicer is triggered very rapidly by the changing of the region of interest. Without *dmTT*, data transport can be very inefficient. If the data manager resides on *mach B*, then the first case will cause a bottleneck because much data must be transported over the network that not will be visualized. On the other hand, if the data manager resides on *mach A*, then the second case will cause a bottleneck because the slicer will need to read its input over the network for each evaluation. However, when *dmTT* is used, both cases can be handled efficiently. For the first case we configure *dmTT* to transport only the sliced data, and hence only these data are copied to *mach B*. For the second case we configure *dmTT* to transport all data to *mach B*. A local copy of the data is then available for rapid selections of the data.

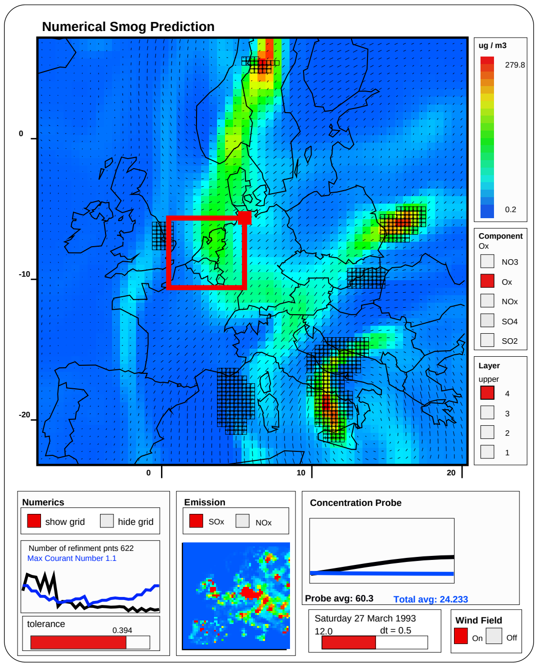
6. Application.

The CSE was applied to the simulation of a model for smog prediction over Europe. The full blown model forecasts the levels of air pollution, which is characterized by approximately 104 reactions between ca. 70 species. For example, the concentrations of ozone (O_3), sulphur dioxide (SO_2) and sulphate aerosol (SO_4) are calculated. The vertical stratification is modeled by four layers; the surface layer, the mixing layer, the reservoir layer, and the upper layer. The physical and chemical model is described by a set of partial differential equations that describe advection, diffusion, emission, wet and dry deposition, fumigation, and chemical reactions.

An important numerical utility to solve these equations is local grid refinement. This technique is used to improve the quality of the model calculations in areas with large spatial gradients (for example in regions with strong emissions). The tradeoff to be made in local grid refinement is calculation accuracy versus computation speed.

We have used the CSE to steer various aspects of the smog prediction simulation. We name a few of these aspects:

- control of the tolerance value that determines where refinement is necessary.
- editing of emission data.
- use of a bounding box as a concentration probe. The coordinates of the bounding box steer the slicing satellite, which in turn triggers the calculator and logging satellites. The result of the logging satellite triggers the PGO editor.
- interactive control over simulation time.



Number of refinment pnts 622
Max Courant Number 1.1
tolerance 0.394
Probe avg: 60.3
Total avg: 24.233
Saturday 27 March 1993
12.0 dt = 0.5

Figure 7: Smog prediction

Figure 7 shows a snapshot of a step in the simulation. An example of a question a numerical mathematician would pose would be to gain insight in the relationship between grid refinement tolerance, the maximal Courant number, and the simulation time. Satellites can easily be configured to address this question. The graph on the lower left shows a log of the number of cells that were refined and the maximum Courant number. The *dmlog* satellite records the data for display. Hence the effects of changes on the tolerance or the simulation time will be displayed immediately.

Similarly, average concentration probes of a region of interest can be defined through the combination of the *dmslice*, *dmcalculator* and *dmlog* satellites. The user specifies the region of interest is specified by dragging the red bounding box. *dmslice* slices the region of interest, *dmcalculator* calculates the average of the sliced area, and *dmlog* maintains the log. The log of these variables is plotted in the lower right of figure 7. Notice that values output from *dmslice*, *dmcalculator*, and *dmlog* are derived variables and are not variables in the simulation. A different operation on the area of interest, for example the maximum concentration, can be plotted by simply changing the expression in *dmcalculator*. Figure 4 shows a view of the trigger manager of this configuration.

This particular configuration runs at approximately two frames a second on a modern workstation. The amount of data involved is substantial. Depending on tolerance level, the amount of data may vary between one and four megabytes per time step. The simulation has 447 time steps. Approximately

90 percent of the CPU time was taken by the simulation satellite. The remaining 10 percent was used by the other satellites.

7. Comparison with Other Systems.

Many research and development teams have designed and implemented interactive visualization environments. Williams, Rashure and Hanson [4] provide a framework to understand design tradeoffs when developing data flow based visualization systems. Giving an in depth analysis of other visualization environments is outside the scope of this paper. Instead, we discuss only some issues that resemble those in the CSE . Many of the concepts in this paper have their counterparts in other systems. However, their combination and application to steering is novel.

- In data flow environments operators are combined by linking output and input ports. Operators are executed upon availability of data on the input port. Operators are packaged as modules, and most environments provide high level tools for building modules.

  IRIS Explorer [5] is an example of an advanced data flow visualization environment. However, there are many fundamental differences between IRIS Explorer and CSE . First, direct manipulation is very difficult to achieve in data flow environments. In IRIS Explorer there is no one-to-one relation between geometry and the corresponding Lattice object in an upstream module. This makes direct manipulation of objects in the simulation very tedious. In contrast, with CSE 's binding mechanism direct manipulation is ensured. Second, IRIS Explorer's mechanism to manage data transport differs from CSE 's. Global and local controllers are configured as a result of the topology of the data flow map. In contrast, data transport in CSE is managed by a centralized data manager. When efficient data transport is necessary, the tuple transport satellite can be used. Finally, in contrast to IRIS Explorer's rigid and hard-wired firing algorithm, CSE 's control rules are very flexible and are managed by a satellite. Many other IRIS Explorer functions are built in the run-time system.

- Glyphmaker [6] is a system that allows users to customize graphical representations using a glyph editor and a simple point-to-click binding mechanism. Glyphmaker is implemented as a collection of modules in IRIS Explorer.

  Allowing users to specify customized graphical representations resembles the main idea of the PGO editor, although the implementations are very different. Glyphmaker's graphics primitives are targeted to glyph specifications for the visualization of data, whereas the PGO's graphics primitive set is larger, and aims at both visualization and user input. Also, Glyphmaker's variable names are distilled from an input file by the Read Module. These names can then be used by the binder to bind to active elements. In CSE , names in the data manager are explicit and can be used in the satellites' parameterization.

- VIEW [7] is a system that is based on a tight coupling of on-screen geometry with a database. A data drawing tool allows users to define composite geometric objects by selecting primitive graphical components from the database. In addition, an event-definition mechanism allows the user to customize interaction sequences. A tool scripting language is used to specify these interaction sequences, and simple selection functions are offered to bind names in the scripts to geometry in the database. Event monitors are used to execute scripts.

  A principle difference between VIEW and CSE is event handling. VIEW provides event monitors to customize interaction sequences. Events in VIEW include changes in input device state and picking of geometry. CSE notion of events is based exclusively on state changes within the data manager. Satellites may receive events by subscribing on the state change.

- Spreadsheet Images [8] is a data visualization system based on spreadsheets. Cells may contain graphical objects, widgets, or formulas written in a scripting language. The output of a cell can be referenced by other cells, resulting in a number of dependency relationships between cells. These dependency relationships are represented by a directed acyclic graph which, when a cell is modified, is updated through a predefined firing algorithm.

  A similar aspect with Spreadsheet images is the strong emphasis of a common user interface metaphor. Spreadsheets are conceptually easy to learn, and the screen space is used very effectively. In contrast, however, CSE groups operators with similar functional behavior in one card tray instead of scattering them throughout the spreadsheet.

8. CONCLUSION

CSE is a modular computational steering environment that provides an interface between a researcher and an ongoing simulation. The interface consists a wide range of cooperating satellites that implement various visualization functions. The kernel of the CSE architecture is designed to be very simple, flexible and minimalistic. All higher level functionality is pushed into the satellites, thus ensuring that a rich environment can be developed yet maintaining the simplicity and flexibility of the underlying architecture.

The notion of modular satellites is certainly not new, and has been applied to many visualization environments. However, CSE takes this modularity one step further by defining systems functions in satellites. These system functions, which are usually hard-wired in the runtime systems of other visualization environments, can be tailored to meet the specific needs of the simulation environment. We presented *dmTM* and *dmTT* as two examples of systems satellites.

We have presented a standard user interface metaphor for all satellites. The card tray can be viewed as a generic operator and individual cards are viewed as parameterizations of the operator. The card tray is easy to understand since it provides an intuitive metaphor for organizing and retrieving cards.

CSE can be used in many configurations. The minimal would include one data manager, the PGO editor and the simulation, which ensures lean but efficient support for many tasks. A full blown application can include many visualization satellites in combination with system satellites.

REFERENCES

1. R.E. Marshall, J.L. Kempf, D. Scott Dyer, and C-C Yen. Visualization Methods and Simulation Steering a 3D Turbulence Model of Lake Erie. *1990 Symp. on Interactive 3D Graphics, Computer Graphics*, 24(2):89–97, 1990.

2. J.J. van Wijk and R. van Liere. An Environment for Computational Steering. Technical Report CS-R9448, Centre for Mathematics and Computer Science (CWI), 1994. Presented at the Dagstuhl Seminar on Scientific Visualization, 23-27 May 1994, Germany, proceedings to be published.

3. J. Mulder and J.J. van Wijk. 3D Computational Steering with Parameterized Graphics Objects. In *Proceedings Visualization '95*. IEEE Computer Society Press, Los Alamitos, CA, 1995.

4. C. Williams, J. Rasure, and C. Hansen. The State of the Art of Visual Languages for Visualization. In *Proceedings Visualization '92*, pages 202–209, 1992.

5. Explorer Development Team. Iris Explorer 2.0 Module Writer's Guide. Technical Report 007-1369-020, Silicon Graphics Inc, 1993.

6. W. Ribarsky, E. Ayers, J. Eble, and S. Mukherjea. Glyphmaker: Creating Customized Visualization of Complex Data. *IEEE Computer Graphics and Applications*, 14(4):57–64, 1994.

7. L. Bergman, J. Richardson, D. Richardson, and F. Brooks Jr. VIEW – An Exploratory Molecular Visualization System with User-Definable Interaction Sequences. *Computer Graphics*, 27(6 (SIGGRAPH '93)):117–126, 1993.

8. M. Levoy. Spreadsheets for Images. *Computer Graphics*, 28(6 (SIGGRAPH '94)):139–146, 1994.