Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

A Survey of Computational Steering Environment

J.D. Mulder, J.J van Wijk, R. van Liere

Software Engineering (SEN)

SEN-R9816 September, 1998

Report SEN-R9816
ISSN 1386-369X

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

A Survey of Computational Steering Environments

Jurriaan D. Mulder
CWI,
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Jarke J. van Wijk
Faculty of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Robert van Liere
CWI

ABSTRACT
Computational steering is a powerful concept that allows scientists to interactively control a computational process during its execution. In this paper, a survey of computational steering environments for the on-line steering of ongoing scientific and engineering simulations is presented. These environments can be used to create steerable applications for model exploration, algorithm experimentation, or performance optimization. For each environment the scope is identified, the architecture is summarized, and the concepts of the user interface is described. The environments are compared and conclusions and future research issues are given.

1991 Computing Reviews Classification System: I.6.4 [Simulation and Modeling]: Model Validation and Analysis; I.6.6 [Simulation and Modeling]: Simulation Output Analysis; I.6.7 [[Simulation and Modeling]: Simulation Support Systems - Environments.
Keywords and Phrases: Computational Steering
Note: Published in Future Generation Computer Systems, special issue on Visualization, 1998.
Work carried out under CWI project SEN-1.3, Interactive Visualization Environments

1. INTRODUCTION

Computational steering can be defined as the interactive control over a computational process during execution. In an interactive computational process, a sequence of specification, computation, and analysis is performed. For each adaption that is to be made to the computational model, this process has to repeated. Computational steering closes the loop such that scientists can respond to results as they occur by interactively manipulating the input parameters. Computational steering enhances productivity by greatly reducing the time between changes to parameters and the viewing of the results [11], it enables the user to explore a what-if analysis [4], and as changes in parameters become more instantaneous, the cause-effect relationships become more evident [14].

Computational steering is applicable in many areas of computational science. In a broad perspective, all interactive computational processes can be regarded as a form of steering. Examples in this regard are the on-line debugging of source codes, interactive scientific visualization, and computer games. We confine ourselves to the steering of ongoing scientific and engineering simulations. In this area, three main uses of computational steering can be distinguished: *model exploration*, *algorithm experimentation*, and *performance optimization*. With model exploration, computational steering is used to explore parameter spaces and simulation behavior to gain additional insight in the simulation. In algorithm experimentation, computational steering allows the user to adapt program algorithms in run time, for instance to experiment with different numerical solving methods. In performance optimization, steering is used to manually improve an application's performance, e.g. to perform load balancing interactively in parallel applications.

Computational steering has been recognized as an important issue for over a decade. The influential report of the US National Science Foundation on scientific visualization published 1987 [12], stated computational steering to be a valuable tool for scientific discovery. Brooks expressed the need for generalized tools for interactive steering for large computations in 1988 [3]. Over the years, many computational steering applications and systems have been developed. The results can be classified into application specific computational steering systems, domain specific computational steering systems, and more generally applicable computational steering environments. An example of an early computational steering application is described by Marshall et al. [11]. They present a system for the visualization and steering of a 3D turbulence model of Lake Erie. More general but application domain specific computational steering systems have for instance been developed for molecular dynamics simulation. Bergman et al. [1] describe the VIEW system, an exploratory molecular visualization system with user-definable interaction sequences. Leech et al. [9] presented SMD, a system for the interactive steering of molecular dynamics calculations of protein molecules. Vetter composed an annotated bibliography focussed on computational steering research issues, systems, and applications [19].

In this paper, we present an overview of several general computational steering environments that have been developed and presented over the years. The environments we discuss are: VASE, SCIRun, Progress, Magellan, CUMULVS, VIPER, and CSE. This list of steering environments is not exhaustive, but it is typical for the systems encountered in computational steering research. The environments will be compared on characteristics involving the scope of the environments, their architectures, and their user interfaces. We will first describe the principle concepts of general computational steering environments, and define the characteristics on which the environments will be discussed. In section 3, a short description is given of each of the environments. Section 4 summarizes and discusses the main similarities and differences between the environments. Conclusions and issues for future research are given in section 5.

2. Environment Characteristics

Figure 1 depicts a diagram of a computational steering process. The computational steering environment comprises three major components: the user interface, the application, and the communication and data transfer between the user interface and the application. These components may be handled by separate processes that might run in a distributed environment, or they can be united into a single process running on a single machine. The application to be steered might be a distributed application itself. Furthermore, the steering process does not have to be limited to single applications or single users. Several different users might perform collaborative steering of a single or multiple applications simultaneously.

The environment must monitor the application and extract the desired information to be presented to the user. The type of information can vary depending on the scope of the steering application. For model exploration, the user will primarily be interested in the application's output data and input parameters. For algorithm experimentation, the user will have to be informed of the application's program structure. For performance optimization, the user will have to be informed about the application's configuration and progress. In parallel or distributed applications for instance, the distribution of the application over different processors or platforms has to be known, along with processor and network loads. To actuate steering actions, the environment must have access to the application's input parameters, the application's execution code, or the application's configuration. The access to the monitoring information and the steering items could be synchronous or asynchronous. For instance, application output data to be extracted might not always be valid, and input parameters might not be adjustable during certain computations.

As computational steering is a highly interactive process, a crucial aspect of a computational steering environment is the user interface. The user interface has two tasks. First, the information extracted from the application has to be presented to the user. Ideally, this will be accomplished through the use of visualization, although it might also be achieved through simple textual display. The second task of the user interface is to allow the user to manipulate the steerable items of the application. This varies from application input parameter manipulation for model exploration, to code generation for algorithm experimentation, and reconfiguration for performance optimization. These user actions can be performed through textual input, through the use of simple graphical input widgets such as sliders and buttons, or by direct manipulation on the visualization or complex (3D) input widgets.

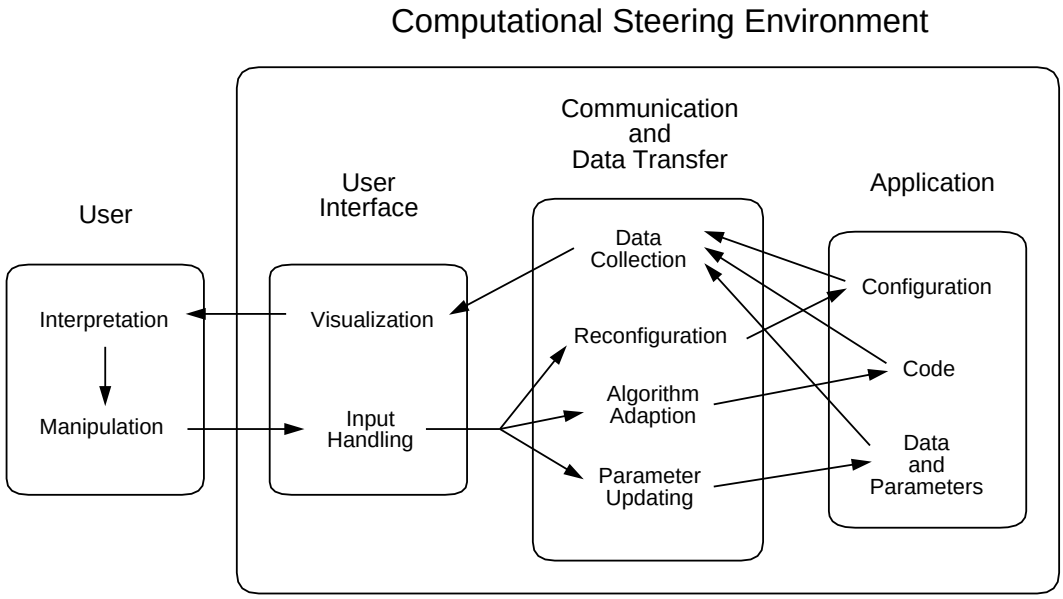

Figure 1: Diagram of a computational steering process

Computational steering environments can differ in many of the aspects described above. We have grouped the aspects in three categories: the scope, the architecture, and the user interface.

Scope

First, the scope of the environment is identified. In particular, the following aspects are addressed:

- The type of steering to be performed: Is the system designed for model exploration, algorithm experimentation, performance optimization, or a combination of these?
- Existing or new applications: Can existing applications be incorporated into the environment, or is the environment intended for the development of new applications?
- Distribution: Is the system designed to steer distributed applications? Can the system itself be distributed, e.g. can the user interface and the target application run on different hosts?
- Multiple applications: Can the environment be used to steer multiple applications simultaneously?
- Multiple users: Does the environment allow multiple users to steer the same application?
- User and developer distinction: Does the environment make a distinction between developers that create the steering application and the end-users?

Architecture

Besides a description of the general layout of the environment, special attention is given to the aspects of:

- Application annotation: To create a steering application it has to be indicated which information is to be provided to the user and which steerable items the user can manipulate.
- Data acquisition: How does the environment extract the monitoring information from the application?
- Steering access: How does the environment access the steerable items of an application?
- Synchronization: What concepts are available for synchronization?

User Interface

For each environment, the user interface concepts are identified on the aspects of:

- Data presentation: How is the monitoring information presented to the user?
- User input: What interface concepts are available to the user for the manipulation of the steerable items?

3. Computational Steering Environments

3.1 VASE

One of the first computational steering environments that has been developed was VASE [5, 7]. VASE stands for *V*isualization and *A*pplication *S*teering *E*nvironment, and was developed at the University of Illinois. The VASE system evolved from earlier experience with projects involving high-performance distributed visualization systems such as Vista [18] and RIVERS [6].

Scope

VASE was designed for model exploration and algorithm experimentation. The steering capabilities provided by the VASE system include the altering of the values of key parameters and the addition of new code at key places in the application. Existing applications are integrated in the environment by annotation of the application's source code. It is emphasized that this is performed by the application developer. An abstract program structure that describes the essential algorithms and data on a high level is created for the end-user to work with. Multiple applications can run in a distributed heterogeneous environment and be steered simultaneously. VASE does not provide any specific support for collaborative steering by multiple users.

Architecture

The VASE system consists of a collection of programming tools and system software. With these tools, the developer creates a steerable program structure of the application program that describes the essential algorithms and data on a high level. The structure consists of logical blocks representing contiguous source code lines and arcs that represent the control flow between these blocks. At the arcs, breakpoints can be inserted and it can be indicated which data is accessible at that point for visualization or steering.

Monitoring and steering are performed by breakpoint scripts. Scripts can read and write variables in the application, control the flow of data between processes (e.g. send data to a visualization process), and call subroutines defined in the application program. The scripts are executed by the VASE interpreter which is linked directly into the executable application and therefore has direct access to the application address space. A script is executed when the running application encounters the breakpoint, so steering is performed synchronously.

Data communication is performed over data channels between VASE data ports. The ports are the endpoints for data communication between VASE application processes. They can be added or removed at any time, breakpoint scripts can reference them by name, and users can specify connections between them to create data channels.

User Interface

During a VASE session, the VASE Configuration and Execution Tool is used to configure and execute VASE executables through a graphical user interface. Configuration includes adding or deleting applications, specifying application hosts, adding, removing, and modifying data ports and channels, and altering breakpoint scripts and breakpoint modes.

The actual steering of an application is achieved in a textual manner with the use of breakpoint scripts. These scripts are written in an extended subset of the C programming language, and are interpreted at run time. VASE does not offer any tools or utilities for the visualization of application output data. This must be accomplished with the use of existing visualization packages such as IRIS Explorer. Breakpoint scripts, data ports, and data channels have to be configured to transfer the data from the application to the visualization package.

3.2 SCIRun

The second computational steering environment we consider is SCIRun [14, 15]. The SCIRun environment was developed at the University of Utah. The name SCIRun was derived from the *S*cientific *C*omputing and *I*maging research group.

Scope

SCIRun primarily provides steering for model exploration and algorithm experimentation. In addition, some minor functionality for performance optimization is available. SCIRun was designed for the development of new applications, although it is possible to incorporate existing applications into the system. A SCIRun application is multithreaded and can run on a single multiprocessor system. SCIRun was not designed to steer multiple applications simultaneously, and no support for collaborative steering by multiple users is provided. Furthermore, no separation between application development and application use is applied. SCIRun is intended as a computational workbench that allows scientists to interactively construct, debug, and steer large-scale scientific computations.

Architecture

SCIRun is based on a data flow programming model common to many scientific visualization packages, such as AVS and IRIS Explorer. The main components in the SCIRun data flow model are: the module, the port, the data type, and the connection. A module represents an algorithm or operation. A port provides a connecting point for routing data to different stages of the computation process. A data type is a quantity such as a scalar field or matrix. A connection is used to connect two modules: the output port of a module can be connected to the input port of one or more other modules.

An application in SCIRun resides in one or more modules. SCIRun is implemented in C++. Writing a new module involves writing a new C++ class. The constructor function of this class creates the input and output ports for the module and defines which parameters the user interface may control. To create a steering application, the user constructs a network out of the modules using a visual programming paradigm. The user configures each module and establishes the desired connections between the modules for the data flow.

Steering actions can be both synchronous and asynchronous. An example of an asynchronous steering action is the adjustment of some parameter of a single module that can be adjusted regardless of the current state of the module computation. An example of a synchronous steering action is the adjustment of a boundary condition that affects multiple modules that form a loop of a time varying problem. Such a steering action will be integrated in the next iteration of the loop.

User Interface

The user interface of SCIRun includes several predefined modules for data visualization and program monitoring (progress meters, thread display, memory usage statistics). For user input, modules are equipped with a Tcl/Tk user interface with which the steerable items of that module are controlled. In addition, some modules have predefined 3D widgets for 3D interaction. A key module in this regard is the Salmon module. Salmon is the graphical viewer that collects the geometric primitives from any number of modules and presents them in a single 3D view. Salmon has the ability to send messages upstream the data flow network. This allows steering of upstream module parameters by direct manipulation on the objects displayed by the Salmon module.

3.3 Progress and Magellan

Progress [20] and Magellan [21] have been developed at the Georgia Institute of Technology. Progress stands for *Prog*ram and *Re*source *S*teering *S*ystem. Magellan is a prototype system designed as the successor of Progress.

Scope

The types of steering that can be performed with Progress are model exploration and performance optimization. Existing high-performance multi-processor applications can be extended with interactivity by annotation of the application's source code. The annotation is performed by the application developer, and is intended to provide the end-user with an abstraction of the steerable application that only reveils the important steering parameters and output data. The steerable application can be run on a different machine than the user interface. Progress does not provide any particular support for simultaneous steering of multiple applications or collaborative steering by multiple users. In Magellan, special attention is devoted to high performance of the steering process itself. Furthermore, Magellan allows for multiple applications to be steered simultaneously.

Architecture

To integrate an application into the steering system, the application's code is annotated with Progress library calls to create a steering object model. Several operations can be performed on the steering objects defined in

the object model. The operations *probe* (an asynchronous read or write of an object), *sense* (a synchronous capture of an object's state), and *actuate* (a synchronous modification of a steering object) can be performed on all objects. *Synch points*, *functions*, and *scripts* are operations that can be performed on specialized steering objects. Synch points are used to halt an application. Functions are predefined in the application and can be used to alter application specific data structures within the executing application (other than the known steerable objects). Scripts provide users with the functionality of combining other steering operations in a language form for repeated execution.

At run-time, the system consists of a server and a client. The server executes as a separate thread in the same memory space as the application. The server has three basic tasks: interact with the steering client, gather monitoring output from the application, and steer the application via the steering objects it has registered. The client is a single threaded Motif application which can run on a remote machine. The client has three basic tasks: interact with the user, communicate with the steering server, and keep relatively consistent state information about the steering objects. The client is transitory and can connect to the server many times throughout the server's (and application's) existence.

In Magellan, there is a server for each application. A master server is used for the communication among the different servers and between the servers and clients. This communication is based on the language ACSL, an *A*dvanced *C*omputational *S*teering *L*anguage.

User Interface

The client process provides a command and graphical user interface to the end-user to control the steering server. Simple monitoring and steering commands can be applied to registered steering objects through interface buttons, and a command line is available for expressions not easily entered in the graphical interface. The client has only limited graphical display functionality. For more advanced visualizations, the client must interface to existing program animation or data visualization systems such as Glyphmaker [17]. The user interface of Magellan is more text oriented than that of Progress, as the steering commands are specified in ACSL. However, ACSL statements may also be bound to graphical user interfaces.

3.4 CUMULVS

CUMULVS [4] is developed at the Oak Ridge National Laboratory. The current CUMULVS system evolved from an earlier prototype system that linked a PVM application to AVS for floating point data visualization and some simple steering operations [8].

Scope

The steering capabilities of CUMULVS include model exploration and performance optimization. CUMULVS aims at existing single parallel PVM programs. Multiple users can connect user interface programs to the application for collaborative steering. No explicit distinction is made between the application developer and the end-user, but application construction is separated from application usage. CUMULVS distinguishes itself from other environments by providing a mechanism for fault-tolerance. In case of a failure, applications can then roll back to a saved state, in stead of having to be restarted.

Architecture

The CUMULVS library is divided in two pieces: one for the application program and one for the visualization and steering front end. These two libraries encompass all the connection and data protocols needed to dynamically attach multiple, independent visualization and steering front ends to a running application. The basic principle of CUMULVS is to have the user declare in the application how an array or field of variables has been decomposed over a collection of parallel processors and specify which parameters are allowed to be modified or steered during the computation. This description of the data and their decomposition is also used for user-directed checkpointing, i.e. efficiently collect checkpoint information and restore the application in the event of a failure.

During its initialization, a user interface program gathers information about the data fields and parameters made available in the desired application and selects the portion of the data to be collected. To allow steering, the user interface process creates a loosely synchronized connection with the application which guarantees that all tasks in the application will apply the steering updates at the same time or point in the application. To prevent multiple viewers from steering the same parameter simultaneously, a viewer can lock a steerable

parameter by obtaining the steering token of that parameter.

A separate process (one per machine) is used to handle fault-tolerance: the checkpointing daemon. The programmer must specify what variables need to be saved at the checkpoints and provide the logic to determine if the application is starting normally or from a checkpoint. CUMULVS manages the details of retrieving the most current (coherent) checkpoint and loading it into the user's variables.

User Interface

The visualization and steering front-ends currently provided with the CUMULVS system are a text-only viewer, a sample custom Tcl/Tk viewer, and a standard AVS-compatible viewer. The particular user interface methods available in these viewers for steering are not described, but it can be expected that steering primarily has to be performed with the text-only viewer, or with a custom built (Tcl/Tk) user-interface, as AVS is primarily intended for data visualizations. CUMULVS provides a C language interface to communicate with other viewing and steering interfaces.

3.5 VIPER

VIPER [16] stands for *VI*sualization of massively *P*arallel simulation algorithms for *E*xtended *R*esearch. VIPER is developed at Technical University of Munich, Germany.

Scope

VIPER is primarily aimed at model exploration of existing massively parallel high performance (CFD) applications. In addition, VIPER allows for the distribution of the processes in the computational steering process. No support is reported for the simultaneous steering of multiple applications or by multiple users, and no explicit distinction is made between the application developer and end-user.

Architecture

VIPER is based on a client/server/client architecture, which they refer to as a Dual Server model. One client is the computational unit which is the (remote) massively parallel application. The other client is the visualization unit formed by the processes of the user interface. The server is called the connectivity unit and is formed by the processes of the Dual Server which offers an infrastructure to extract data out of the application, transfer the data, and hand it out to the visualization unit. The Dual server consists of multiple multi-threaded processes which can run in parallel on distributed machines.

The application program is annotated to identify the data and input parameters as objects. Each object is associated with a synchronization point. When the application encounters such a synchronization point, the Dual Server is notified which in turn extracts the data out of the application or restores input parameters. Furthermore, the synchronization between the three different units can be specified, ranging from fully synchronized to completely asynchronous.

User Interface

For user interaction a graphical interface is supplied. It is however not further specified which capacities this user interface has for the visualization and parameter manipulation.

3.6 CSE

CSE [10, 22] stands for *C*omputational *S*teering *E*nvironment. The CSE is developed at the Center for Mathematics and Computer Science in Amsterdam, the Netherlands.

Scope

The steering capability of the CSE is primarily focussed at model exploration. Multiple existing applications can be integrated into the environment by annotation of the application source codes. The applications and multiple user interface processes can be distributed over different platforms. No strict distinction is made between application developers and end-users. Although the integration of an application requires source code annotation, the construction of the final steering application can be performed by the end-users.

Architecture

The architecture of the CSE is centered around a data manager that acts as a blackboard for communicating data values. Separate processes (called satellites) can connect to the data manager and exchange data with it. An application is packaged as a satellite. The application code is annotated to make a connection to the data manager, to publish its output variables for visualization and its input parameters for steering, and to indicate

when these variables are to be read and/or written.

All data transport and communication is performed through the data manager. The purpose of the data manager is twofold. First, it manages a data base of variables. Satellites can create, open, close, read, and write variables. Second, the data manager acts as an event notification manager. Satellites can subscribe to state changes in the data manager. These functions of the data manager allow the satellites to communicate and exchange data with other satellites. Special synchronization variables can be used to synchronize the different satellite processes. The satellite processes can run on different hosts in a distributed environment. To reduce network loads, the actual implementation of the data manager incorporates local data managers that can run on the same machine as a satellite process. Careful design of the communication protocols between the local data managers ensures that data is only transported when needed.

User Interface

The user interface to visualize and manipulate data is provided by additional satellite processes. The most important user interface satellite is the PGO editor [13]. This satellite is an interactive MacDraw-like graphics editing tool that allows a researcher to create custom 2D, 3D and immersive 3D user interfaces to visualize and manipulate the data in the data manager. The user draws an initial layout of the interface with graphics objects. He then specifies how the geometry and attributes of these objects should be coupled to variables present in the data manager. The coupling of the objects to the data can be bidirectional. As a result, the user can manipulate variables (i.e. the simulation steering parameters) by direct manipulation on the object. At the same time, changes in the data in the data manager cause the object geometries to change, i.e. the objects serve as the visualization of the simulation output data.

4. Summary and Discussion

Table 1 summarizes the computational steering environments on their major characteristics. The table shows that none of the environments provides all three possible applications of computational steering (model exploration, algorithm experimentation, and performance optimization). Although some of the systems provide some functionality for the application of steering in one of the areas not indicated in the table, they do not provide full functionality in that respect. SCIRun for instance, does allow some monitoring of a running application and some tweaking of the data flow network that constitutes the running program. This functionality however is not sufficient for full performance optimization.

To construct a steering application, all systems except SCIRun require the application source code to be manually annotated with program statements by the application developer. SCIRun uses visual programming to create a steering application. However, this only applies to applications that can be constructed out of existing modules. To integrate an existing application into the system, or when the desired functionality is not supplied by the standard modules, the application developer has to program new modules as C++ classes.

For data extraction and parameter access, most environments are based on some sort of client-server model. In Progress, Magellan, and CUMULVS, the user interface process is regarded as the client which can request and manipulate data and steerable objects via a server. The server takes care of the actual extraction and access to the application data and parameters. In VIPER and the CSE, the client server model is carried even further. Here, the application is regarded as a kind of client process as well. The server's main task is to manage the communication and data transport between the different client processes. A difference between VIPER and the CSE is that in VIPER the server is primarily focussed at data transportation in between the application and user interface process, whereas in the CSE the server is also used for data storage and data manipulations by other clients (i.e. satellites) than the user interface process.

Exceptions to the strict client server model are VASE and SCIRun. SCIRun uses a data flow model known from many existing visualization packages such as AVS and IRIS Explorer. In SCIRun however, the application resides in multiple (steerable) modules whereas these visualization packages primarily provide modules needed for the visualization process. Furthermore, in SCIRun feed back loops have been introduced to allow upstream communication for steering. VASE provides the user with a hybrid model of control flow and data flow. The control flow model is used to provide the user an abstract model of the program structure. The data flow model is used to direct data streams in between applications and towards visualization processes.

All environments provide synchronous access to the application data and parameters. Except for SCIRun,

	Scope			Architecture			User Interface
	model exploration	algorithm experiments	performance optimization	Application Integration	Data and Parameter Access Model	Synchronization	
VASE	x	x		manual program annotation to create abstract steerable program structure	hybrid control flow and data flow	synchronous breakpoints in application source code	visualization through existing packages; steering through textual input..
SCIRun	x	x		visual programming for new applications.	dataflow with feed back loops	module firing and centralized module scheduler	visualization through visualization modules; steering through Tcl/Tk interfaces and widgets.
Progress & Magellan	x		x	manual program annotation to create steering object model	client server	synchronous sense and actuate operations	visualization through existing packages; steering through command line and GUI.
CUMULVS	x		x	manual program annotation to indicate data decomposition and steerable parameters.	client server	checkpoints in application code	visualization through AVS; steering through textual and custom (Tcl/Tk) GUI.
VIPER	x			manual program annotation to report data and parameters to Dual Server	client server	sync points associated with data and parameters in application code	GUI.
CSE	x			manual program annotation to connect data and parameters to Data Manager	client server	update locations in application code	graphical editor for 2D visualization and (direct manipulation) steering.

Table 1: Summary of environment characteristics

the access points are indicated in the application source code. In VASE for instance, the locations of the breakpoints indicate the position where the data and parameters are accessible. In SCIRun, the modules provide the output data when valid. Module execution after a steering action is synchronized by a central module scheduler. Progress and Magellan are the only two environments that allow asynchronous access to the application data and parameters. It is left to the application developer to ensure that such asynchronous access is a valid operation. In all the client-server based systems, synchronization between the client and server processes is available but not mandatory. In the CSE for instance, different satellite processes can run asynchronously, while trigger variables can be used to synchronize different satellite processes via the data manager.

On the aspect of the user interface, CSE distinguishes itself from all other environments by providing a graphical editor for the creation of customized 2D and 3D interfaces. The interfaces are used for both the visualization of the application results as for the steering of the application input parameters by direct manipulation on the visualization. The other environments mostly rely on existing packages for the visualization of the monitoring information and use command-line, textual, or basic graphical user interfaces for steering of the input parameters. SCIRun is an exception in this regard as it supplies a limited number of predefined graphical widgets for the visualization and manipulation of steerable parameters, next to per module Tcl/Tk interfaces.

5. Conclusion and Future Research

It is impossible to order the different environments in a ranking from best to worst. They each have their own strong and weak points, and differ in their approaches depending on the envisioned application and wishes and demands of the user. Therefore, we only give global guidelines to which environments seem most suitable for particular types of applications.

In regard of the ease of use of the environments, SCIRun has the advantage as it uses a visual programming paradigm for the construction of the application and the visualization. Furthermore, SCIRun provides some functionality for steering through direct manipulation on (predefined) 3D graphical widgets . Two major disadvantages of SCIRun are that it does not allow the different processes in the steering application to be distributed over multiple machines, and that it is cumbersome to integrate existing applications into the system.

In this regard, CSE might be a better choice. In CSE, existing applications can be integrated with limited annotation of the application's source code, and the application can be run on a remote (super) computer while the user interface resides on a graphical workstation. An additional advantage of CSE is that it allows for the full customization of the user interface through graphical editing, and that it provides steering by direct manipulation on the 2D or 3D (user defined) interface widgets. VIPER resembles the approach of CSE but is targeted more towards parallel (CFD) applications. VIPER however, is not equipped with a fully customizable visualization and direct manipulation steering interface.

SCIRun, CSE, and VIPER provide only limited support for algorithm experimentation. In SCIRun, modules containing different algorithms can be interchanged while in CSE a number of different satellite processes can manipulate the same data in the data manager. The environment that provides the richest capabilities for algorithm experimentation is VASE, which enables the user to write scripts to be inserted at the breakpoints in the control structure of the application.

If performance optimization is a major target of the steering process next to model exploration, Progress and Magellan are very suitable. If the particular application is a massively parallel application built on PVM, CUMULVS is the environment of choice.

As stated in the previous section, integrating an existing application into a steering environment requires manual programming. As a result, to be able to construct a steering application, the user is required to have a high level of knowledge of the simulation application code and the steering environment's libraries. It would be a valuable contribution to each of the environments if the integration of an existing application could be automated, or at least provide the developer with a higher level user interface to assist in the annotation of the application.

CUMULVS distinguishes itself from the other environments by providing a mechanism for fault-tolerance. However, this mechanism aims at recovering from faults that may occur in the running application. Because computational steering aims at giving the user the power to experiment with the application, a fault-tolerance mechanism aimed at user actions could be a valuable addition to a steering environment. Such a mechanism

would allow the user to scroll back in time in the case the user has performed an erroneous steering operation, or if he simply wants to restart the application from a previous state to experiment with different parameter settings from that point on. Some initial research in this regard has already been presented by Brodlie et al. [2]. They studied how backtracking can be used in a framework for the management of a problem solving process.

References

1. L.D. Bergman, J.S. Richardson, D.C Richardson, and F.P. Brooks, Jr. VIEW - an exploratory molecular visualization system with user-definable interaction sequences. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 117–126, 1993.
2. K. Brodlie and L. Brankin. GRASPARC - a problem solving environment integrating computation and visualization. In G.M. Nielson and D. Bergeron, editors, *Visualization '93 (Proceedings of the 1993 Visualization Conference)*, pages 102–109, 1993.
3. F.P. Brooks. Grasping reality through illusion – interactive graphics serving science. In *Proceedings of the CHI '88, ACM Conference on Human Factors in Computer Systems*, pages 1–11. ACM, 1988.
4. G.A. Geist II, J.A. Kohl, and P.M. Papadopoulos. CUMULVS: Providing fault tolerance, visualization, and steering of parallel applications. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):224–235, 1997.
5. R. Haber, B. Bliss, D. Jablonowski, and C. Jog. A distributed environment for run-time visualization and application steering in computational mechanics. *Computing Systems in Engineering*, 3(1–4):501–515, 1992.
6. R.B. Haber. Scientific visualization and the RIVERS project at the national center for supercomputing applications. *IEEE Computer*, 22:84–89, August 1989.
7. D.J. Jablonowski, J.D. Bruner, B. Bliss, and R.B. Haber. VASE: The visualization and application steering environment. In *Proceedings of Supercomputing '93*, pages 560–569, 1993.
8. J.A. Kohl and P.M. Papadopulos. A library for visualization and steering of distributed simulations using PVM and AVS. In V. Van Dongen, editor, *Proceedings of the High Performance Computing Symposium, Montreal, Canada*, pages 243–254, 1995.
9. J. Leech, J.F. Prins, and J. Hermans. SMD: Visual steering of molecular dynamics for protein design. *IEEE Computational Science & Engineering*, 3(4):38–45, 1996.
10. R. van Liere, J.D. Mulder, and J.J. van Wijk. Computational steering. *Future Generation Computer Systems*, 12(5):441–450, April 1997.
11. R. Marshall, J. Kempf, S. Dyer, and C.-C. Yen. Visualization methods and simulation steering for a 3D turbulence model of Lake Erie. *Computer Graphics*, 24(2):89–97, 1990.
12. B.H. McCormick, T.A. DeFanti, and M.D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), November 1987.

13. J.D. Mulder and J.J. van Wijk. 3D computational steering with parametrized geometric objects. In G.M. Nielson and D. Silver, editors, *Visualization '95 (Proceedings of the 1995 Visualization Conference)*, pages 304–311, 1995.

14. S.G. Parker and C.R. Johnson. SCIRun: a scientific programming environment for computational steering. In *Proceedings of Supercomputing '95*, 1995.

15. S.G. Parker, D.M. Weinstein, and C.R. Johnson. The SCIRun computational steering software system. In E. Arge, A.M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools for Scientific Computing*, pages 1–40. Birkhäuser Verlag AG, Switzerland, 1997.

16. S. Rathmayer and M. Lenke. A tool for on-line visualization and interactive steering of parallel hpc applications. In *Proceedings of the 11th International Parallel Processing Symposium, IPPS 97*, pages 181–186, 1997.

17. W. Ribarsky, E. Ayers, J. Eble, and S. Mukherjea. Glyphmaker: Creating customized visualizations of complex data. *IEEE Computer*, 27(4):57–64, July 1994.

18. A. Tuchman, D. Jablonowski, and G. Cybenko. Run-time visualization of program data. In *Visualization '91 (Proceedings of the 1991 Visualization Conference)*, pages 255–261, October 1991.

19. J. Vetter. Computational steering annotated bibliography. *SIGPLAN Notices*, 32(6):40–44, June 1997.

20. J. Vetter and K. Schwan. Progress: a toolkit for interactive program steering. In *Proceedings of the 1995 International Conference on Parallel Processing*, pages 139–142, 1995.

21. J. Vetter and K. Schwan. High performance computational steering of physical simulations. In *Proceedings of the 11th International Parallel Processing Symposium, IPPS 97*, pages 128–132, 1997.

22. J.J. van Wijk and R. van Liere. An environment for computational steering. In G.M. Nielson, H. Müller, and H. Hagen, editors, *Scientific Visualization: Overviews, Methodologies, and Techniques*, pages 89–110. Computer Society Press, 1997.