CONCURRENCY—PRACTICE AND EXPERIENCE

Concurrency: Pract. Exper. 2001; **0**:1–40 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*

The Polder Computing Environment, a system for interactive distributed simulation

K. A. Iskra[1,*], R. G. Belleman[1], G. D. van Albada[1], J. Santoso[1], P. M. A. Sloot[1], H. E. Bal[2], H. J. W. Spoelder[3] and M. Bubak[4,5]

[1] *Section Computational Science, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands*

[2] *Division of Mathematics and Computer Science, Faculty of Sciences, Vrije Universiteit, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands*

[3] *Division of Physics and Astronomy, Faculty of Sciences, Vrije Universiteit, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands*

[4] *Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland*

[5] *Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland*

(email: {kamil,robbel,dick,judhi,sloot}@science.uva.nl, bal@cs.vu.nl, hs@nat.vu.nl, bubak@uci.agh.edu.pl)

SUMMARY

*Correspondence to: K. A. Iskra, Section Computational Science, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Received July 23, 2001

The paper provides an overview of an experimental, GRID-like computing environment Polder and its components. Polder offers high-performance computing and interactive simulation facilities to computational science. The characteristics of the underlying wide-area cluster system DAS are described. The issues of efficient management of resources, in particular multi-level scheduling and migration of tasks that use PVM or sockets are discussed. The system can be used for interactive simulation, where a cluster is used for high-performance computations, while a dedicated immersive interactive environment (CAVE) offers visualization and user interaction. Design considerations for the construction of dynamic exploration environments using such a system are discussed, in particular the use of Intelligent Agents for coordination. A case study of simulated abdominal vascular reconstruction is subsequently presented: the results of computed tomography or magnetic resonance imaging of a patient can be displayed in CAVE, and a surgeon can evaluate the possible treatments by performing the surgeries virtually and analysing the resulting blood flow, simulated using the lattice-Boltzmann method.

KEY WORDS: problem solving environments, interactive distributed simulation, immersive environment, grid computing, resource management, dynamic load balancing

INTRODUCTION

The Polder Computing Environment (PCE) [1] is an experimental environment comprising hardware and software for high-performance computing and interactive simulation applied to computational science.

Computational science strives to assist researchers in studying complex problems involving phenomena that can be described by computable models. These may involve physical or chemical processes, but are certainly not limited to these domains: biology, economy and traffic management

Prepared using cpeauth.cls

are some of the other important application areas. Such a study typically involves the evaluation of the computational model for a range of model parameters. The goal of the study influences how the parameters must be chosen, and how the results of each model calculation are processed. Generally, the parameter space is very large, making a dense sampling of this space prohibitively expensive. When the goal is to study the behaviour of a system over a wide range of parameters, it is important to quickly locate the regions where interesting phenomena occur, such as phase transitions. When the goal of the study is to optimise the behaviour of the system according to certain criteria, an efficient optimisation method is needed.

Problem solving environments (PSE) aim to provide tools to perform the actual model evaluation, combined with data analysis and presentation tools to analyse the results from a collection of simulations for a range of parameters, and ideally, tools to efficiently select new parameter values on basis of earlier results. Specifically, the selection or adjustment of model parameters is a difficult problem, and in many situations is best performed in an interaction between the human expert and the PSE. This leads to the creation of interactive simulation environments. The details of such an environment are strongly influenced by the specific application; the underlying mechanisms should be made as general as possible to allow re-use of code.

The planning of surgical procedures is an interesting and important area for the application of computational science techniques [2]. By providing appropriate planning support to the surgeon, the cost of the planning procedure can be reduced and, more importantly, the success rate of the procedure can be improved. E.g. in vascular surgery, when planning a bypass procedure, surgeons aim to construct the bypass in such a way that the blood flow is restored as well as possible, while also ensuring that new blockages (stenosis) are unlikely to form. The deposition rate of clots is found to depend on the shear-stress in the flow. Thus, a surgeon can be greatly helped if the blood flow resulting from an intervention can be predicted through interactive simulation of the flow before and after the construction of the bypass. We are currently building such a system, using the geometry of the patient's arteries as derived using magnetic resonance (MR) techniques as input, an immersive interactive environment (a CAVE) for presentation and interaction, and a lattice-Boltzmann code as a flow simulator. This set-up requires a reliable and high-performance computing and networking environment that can give near real-time response.

This takes us from the computational science and PSE side of the PCE to the computer science aspects. The PCE was designed to provide an accessible, versatile experimental environment to an extended research community within a part of the Netherlands. Where current Grid technology aims to create a global computing environment [3], the aims for PCE are more confined. Speaking in networking terms, the PCE adheres to a Municipal Area Network (MAN) and intranet paradigm, rather than a WAN and internet paradigm. The reasons for this choice lie in a better manageability and security that can be realised in this way, a more predictable performance and a higher reliability.

In this metacomputing environment, the Distributed ASCI Supercomputer (DAS) [4], operated jointly by five Dutch Universities and the Advanced School for Computing and Imaging (ASCI†) plays a central role. The DAS consists of four clusters of PCs, each much like a Beowulf cluster [5]. The clusters are internally connected through Myrinet [6] and 100Mb Ethernet, and are linked through a private ATM network and a public network. The operating system is Linux and various communication libraries are available that can use the available local and remote networking facilities efficiently. The

†The ASCI research school is unrelated to, and came into existence before, the Accelerated Strategic Computing Initiative.

DAS is connected to various high-performance systems, including the CAVE and supercomputers at the national computing centre, SARA.

The nodes of the DAS are currently mostly scheduled using “space-sharing”. In order to improve the utilisation in general while retaining the responsiveness of the system for high-priority tasks, we are developing a dynamic, time sharing scheduling system for parallel programs, Dynamite [7]. Dynamite allows the migration of tasks in a parallel program. This dynamic resource management makes it possible to allocate CPU cycles in a dynamic, priority-based manner, honouring the requirements of high priority programs, without unnecessarily starving lower priority jobs.

The realisation of an interactive simulation environment imposes a variety of requirements on the system, besides the performance and reliability provided by a system like the DAS. The need for a combination of high performance computing and an immersive presentation and interaction environment imposes requirements on the interoperability of these systems. The need for interaction with a time-dependent simulation imposes requirements on the management and co-ordination of the virtual (simulation) time within the various components of the system. Portability to other systems, and adaptability to other applications implies a need for a precise separation of functionalities into modular components. These requirements have led us to adopt the High Level Architecture (HLA) [8] as a middleware layer and to implement various integrative and support functions in Intelligent agents (IA).

Where this introduction is largely top-down, the remainder of the paper will present the components of the PCE in a bottom-up fashion, from the DAS, through the operating system and communication layers, the dynamic resource management to the tools for constructing interactive simulation

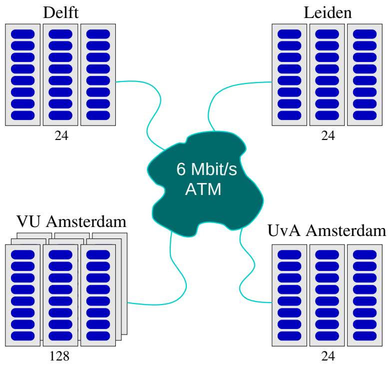

Figure 1. The wide-area DAS system. The system consists of four subclusters interconnected using private ATM network and public Internet network (not shown).

environments to the actual case studies. After that, we will present our conclusions and our views on future work in the PCE.

THE DAS AND ITS ENVIRONMENT

Basic components

Computational power and an environment for immersive presentation of data are both basic and vital components for our PCE. In this section we will discuss in some detail their hardware and software characteristics.

The computing power required by a simulation program varies dramatically but almost always surpasses by far the computational power of a single processor system. Furthermore an interactive environment requires high communication speed and low latency.

The DAS [4] infrastructure meets both requirements of scalable computational power and high speed communications. DAS was designed as a physically distributed homogeneous cluster. We used the cluster technology because of the excellent price/performance ratio of commodity (off-the-shelf) hardware. The system is distributed to give participating universities fast access to local resources. DAS comprises four clusters (see Fig. 1). Each cluster consists of 200 MHz Pentium Pros, the fastest Intel CPU available at the time of purchase. Three clusters have 24 nodes; the cluster at the Free University (VU) initially contained 64 nodes, but was expanded to 128 nodes in 1998.

We decided to keep the DAS system homogeneous to allow easy exchange of software. Both hardware and software are homogeneous: each node has the same hardware and runs the same operating system. We initially chose BSD/OS (from BSDI) as OS, because it is a stable system with commercial support. The increasing popularity of Linux both world-wide and within the ASCI school made us switch to Linux (RedHat 5.2) in early 1999.

Within a cluster the nodes are connected by Myrinet [6]. This is a switch-based network using wormhole routing. The Myrinet switches are connected in a ring topology for the 24-node clusters and in a 4 by 8 torus for the 128-nodes cluster. Myrinet is used as fast user-level interconnect.

The clusters are connected by wide-area networks in two different ways:

- using the National Research Network infrastructure (best effort network) and the LANs of the universities,
- using an ATM-based Virtual Private Network.

On the best effort network the traffic always has to pass about 4 routers, which cause a millisecond delay each. Measurements show that the round trip times vary by about an order of magnitude due to the other internet traffic. For interactive immersive environments this is a highly undesirable characteristic.

The ATM networks provide guaranteed wide-area bandwidth to the DAS system, so the communication latencies vary much less than with shared networks like internet. The ATM connections are all 6 Mbit/s constant links. The round trip times on the ATM connections have hardly any variation and typically are around 4 msec.

For immersive presentation of data we have used the Cave Automatic Virtual Environment (CAVE) [9] facility of the computing centre SARA. The cluster at the University of Amsterdam (UvA) is located in the direct vicinity of the CAVE and thus allows a direct coupling to the SARA network. Typically simulations will run on the DAS cluster whereas visualization will be done in the CAVE, a similar division applies to the workload. This divide-and-conquer approach is necessary due to the computational requirements of both simulation and visualization.

Special hardware

Two of the nodes of the DAS at the UvA host GRAPE-4 [10] boards. These are very high performance special purpose processors designed specifically for the computation of the gravitational or electrostatic interactions in N-body simulations. The combination of a parallel computer like the DAS and these GRAPE boards are used in a study of advanced N-body codes [11].

Communication software

High-speed networks like Myrinet can obtain communication speeds close to those of supercomputers, but realising this potential is a challenging problem. There are many intricate design issues for low-level network interface (NI) protocols [12]. We have designed and implemented a network interface protocol for Myrinet, called LFC [13, 14]. LFC is both efficient and provides the right functionality for higher-level programming systems. The LFC software runs partly on the host and partly on the embedded LANai processor of the Myrinet network interface card. An interesting feature of LFC is its spanning tree broadcast protocol, which is implemented on the NIs. By forwarding broadcast messages on the NI rather than on the host, fewer interactions are needed between the NI and the host, thus speeding up broadcasts substantially. We have also developed a higher-level communication library, called Panda [15], which supports asynchronous point-to-point communication, remote procedure calls, totally-ordered broadcast, and multithreading. On Myrinet, Panda is implemented on top of LFC. The LFC and Panda libraries have been used for a variety of programming systems, including MPI, PVM, Java, Orca, Jackal, and CRL.

We did some measurements on a distributed application (RoboCup) using the DAS cluster and communicating via the UDP protocol. Within a cluster, we used an efficient user-space implementation of UDP on top of Myrinet. This UDP protocol was developed using the fast Panda/LFC communication layer. This protocol obtains a round trip latency of 41 microseconds (1 byte packet) and a throughput of 57 MByte/s (8 KByte packet). Communication over the wide-are ATM network uses a kernel-space UDP protocol. We obtain a round-trip latency of about 1.9 milliseconds and a throughput of 567 KByte/s (between the clusters at the VU and the UvA).

RESOURCE MANAGEMENT

Though the DAS plays a central role in the PCE, the PCE is not confined to this system. As already mentioned, the PCE can make use of the CAVE at SARA. Other important computing resources for the PCE include the pools of workstations at the VU and the UvA. In a heterogeneous and subdivided environment like this, a powerful and flexible resource management (RM) is essential. As argued in

[1], a hierarchical resource management system is appropriate for such systems. Each (sub-)cluster will retain its own local cluster management with its own policies. The distribution of work over the (sub-)clusters is accomplished by a global scheduling layer. The local cluster manager, in turn, will make use of the RM and scheduling services of the node operating system. Together, the three scheduling layers strive to accomplish the scheduling goals for the system.

The efficiency of a metacomputing system can be viewed in two different ways. For High Performance Computing (HPC) a parallel job perspective is taken. The system performance is defined in terms of the turnaround time and possibly near real-time response of highly demanding parallel jobs.
In the view taken by High Throughput Computing (HTC), the performance of the system is mainly defined in terms of the number of jobs that are processed within a certain period of time.
HTC and HPC need not be mutually exclusive. In the view currently being developed in PCE, HTC processing is the normal operating mode of the system. When high priority HPC jobs are submitted to the system, the scheduler obtains the necessary computing resources by checkpointing or migrating HTC jobs.

At each level in the scheduling hierarchy, the situation can change dynamically. This can be caused by changes in the resource requirements of the individual jobs, submission of high-priority jobs, changes in resource availability. At the level of the individual nodes, this dynamism is highest and resource use can change many times per second. The OS scheduler is well-geared to respond on these time scales. At the local cluster level, the scheduler can make load-adjustments on a time-scale of several seconds, to several minutes. At the global level, adjustments are even more costly. Global

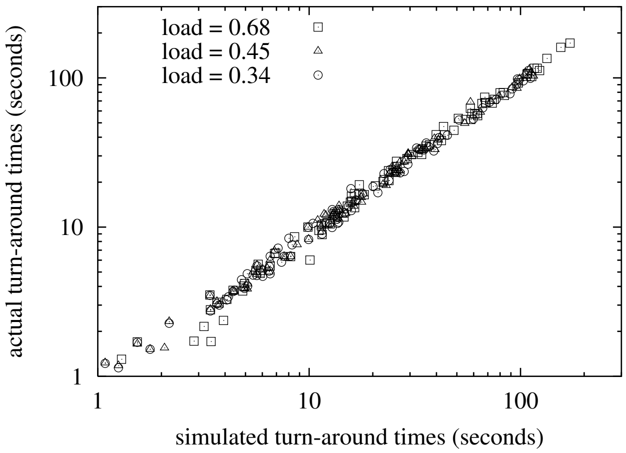

Figure 2. Simulated and actual turn-around times for a collection of PVM jobs on a four node system. In this case, the tasks in a PVM job synchronise only at the beginning and the end of the execution. Measurements were performed for system utilisation "loads" of 34%, 45% and 68%.

scheduling should, therefore, preferably be applied only at the initial start-up of a job, and for long-running jobs [16].

Models for static two-level scheduling

Because of the importance of scheduling for the performance of a metacomputing system such as the PCE, we have built a simulator for the scheduling in these environments [17]. This simulator allows us to study the interaction between the three levels of scheduling, and the effects of the scheduling policies applied. It allows simulation of parallel jobs in a time-sharing environment. The aim of the simulation is not to provide a quantitatively faithful representation of a particular environment (that would require an enormous amount of detail in the simulation), but to at least provide sufficient qualitative and

Prepared using cpeauth.cls

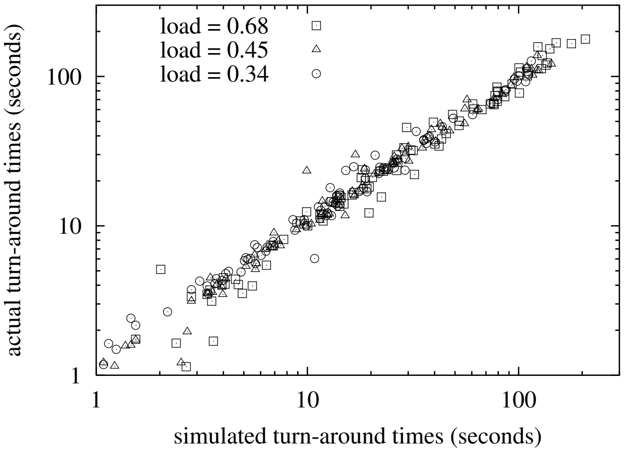

Figure 3. Simulated and actual turn-around times for a collection of PVM jobs on a four node system. In this case, the tasks in a PVM job synchronise every second. Measurements were again performed for system utilisation "loads" of 34%, 45% and 68%. At the higher loads much larger discrepancies between predicted and measured turn-around times are found than in Fig. 2. This is in large part due to variations in the communication delays in the synchronisation phase.

quantitative accuracy to allow a relative comparison of scheduling methods. The simulator is currently being validated. One of the validation problems used was the prediction of the turn-around times for a large number of small PVM jobs in a time-sharing environment (see Fig. 2, 3).

Dynamic scheduling and task migration for communicating programs

As stated in the introduction of this section, the workload on each node of the metacomputer can change dynamically and on fairly short time-scales. For this reason, and also because resources may be reclaimed (or additional resources may become available), it may be desirable to move load from one node to another. This can be accomplished by incorporating a load-balancing mechanism into every

Prepared using cpeauth.cls

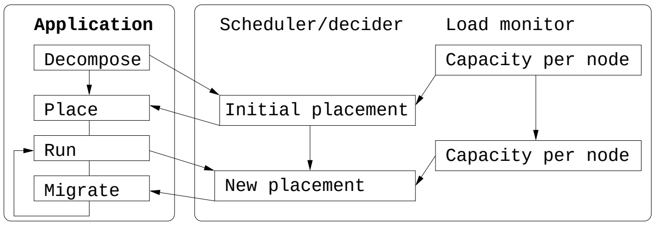

Figure 4. Running an application with Dynamite. An application has to be decomposed into several subtasks already, and it must be linked with the Dynamite libraries. The run-time system places these on computational nodes, starts execution and monitors the system. If it decides that the load is unbalanced (above a certain threshold), one or more task migrations may be performed to establish a new and more optimal load distribution.

program. This allows a very fine-grained load balancing, but puts the responsibility for load-balancing on the shoulders of every single user. Furthermore, in a time-sharing environment, different jobs may now interact in an unpredictable manner. For these reasons, we prefer to place the responsibility for load-balancing with the cluster management software. The migration and scheduling mechanisms in Condor [18] provided a first step towards this goal. However, the migration of tasks in parallel programs adds additional requirements to the checkpointing and migration mechanisms.

We have met this challenge with the Dynamite system [7, 19]. Dynamite provides dynamic load balancing for PVM applications. It supports migration of individual tasks between nodes in a manner transparent both to the application programmer and to the user. Dynamically linked executables are supported, as are tasks with open files and both direct and indirect PVM communication. Applications running under Linux and Solaris are supported, and the migration mechanism is implemented entirely in user space. Neither the operating system nor the application source code need to be modified, only relinking with the modified libraries is required. Dynamite is modular and consists of three largely independent parts: the checkpointer/restarter that is implemented in the dynamic loader and is responsible for preserving the memory image of the running process, the PVM environment that has been extended to preserve the PVM communication across the migration and the scheduling subsystem that performs the migration decisions based on the feedback from the monitors running on computational nodes (see Fig. 4). The Dynamite software is available for research purposes [20].

We are currently working on extending the range of communication libraries supported. Versions of Dynamite for TCP/IP connections (described in the next section) and for MPI are now in the testing phase. The latter incorporates the Hector [21] communication library, and was realised in collaboration with I. Banicescu of Mississippi State University. This library will also allow parallel programs to be checkpointed for fault-tolerance purposes. Furthermore, we are working on a modification of the checkpointing mechanism that should allow tasks to be migrated across clusters. Currently, the migration data must be stored on a network filesystem, we are extending the environment to support direct data transfer between source and destination nodes across a TCP/IP socket connection, which will remove this limitation and should also decrease the migration time.

The interdependence of the tasks in a parallel program complicates scheduling and monitoring. An essential part of our research is concerned with developing suitable models for the scheduling behaviour of parallel programs.

Migratable sockets

Many parallel computing environments use TCP or UDP through the socket interface as a communication mechanism. For this reason we have elaborated the concept and implemented a library that, besides offering the same functionality as system socket library for the TCP/IP protocols family, allows migration of the process without interrupting communication with other processes. The new library, called msocket [22], can be a substitution of the standard socket library so that no changes in the program will be required. All necessary modifications are handled at the library level so that no changes in the kernel are needed, either.

The essential requirement for the socket migration is that all modifications of communication libraries have to be transparent for the TCP and UDP protocols. To easily migrate a process with the socket interface in use, the modifications of the communication library have to warrant that establishing new connections must be allowed regardless of the number of process migrations, and all connections that had been established before the migration took place have to be kept, and data must not be lost.

This, in turn, requires to consider the way in which:

- migrating processes will be able to connect and keep the connection between themselves,
- migrating processes will establish and keep connections with processes which use the original socket library,
- a client using the original socket library will connect to a migrating process; the location of the migrating process should have no influence on communication establishment.

An essential requirement is that a parallel application with processes intended to migrate during runtime should not need to be developed in any special manner, and a programmer should not need to know in advance that a particular application will migrate at runtime.

The requirement not to change the UNIX kernel forces us to modify the system calls and library calls. In the user code the communication through the TCP and UDP protocols uses sockets which are treated by the process as file descriptors. For this reason wrappers are used for each call which has file descriptors as arguments. In some cases this is a simple change which only translates the file descriptor number.

To allow uninterrupted redirection of packets or stream flow to a migrating process, it is possible to use one of the following approaches:

- all communication between processes goes through a daemon which forwards packets to the current process location,
- after migration, the process leaves a piece of code on the *old* machine and this code takes care of forwarding data to the new process location. In this paper, the term *mirror* is used for a process which is used to redirect network packets.
- the migrating process flushes all connections with other processes before migration and afterwards re-establishes them. This requires the immediate cooperation of all other processes involved.

None of the solutions is perfect. The global (centralized) data distribution is not fault tolerant and could be too slow. In the second case, after process migration the machine is still in use, but in a different way. It means that the machine can not go down. This solution is not fault tolerant, either. The third solution requires special signal handling routines in all participating processes to ensure a timely response.

The msocket library is based on all three concepts in a way which eliminates drawbacks. After process migration, all connections have to be redirected to the new process location. The migrating process has to leave a mirror because some data could be on-the-fly (inside network stack buffers or inside active network equipment like a router or switch). The mirror should be removed as soon as possible. In this case, the mirror captures and redirects the packets which were on-the-fly during the process migration.

During normal work, processes use the direct connections. The daemons are not used for passing user data between hosts. Daemons should exist on each machine, and their role is limited only to

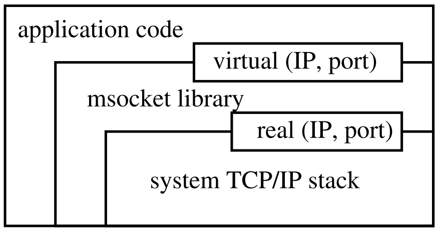

Figure 5. Layers of the library.

redirecting connections. When a process migrates, it has to inform peers about its new location. While restoring the process, daemons on all hosts which were involved in communication with the migrating process should be informed about its new location. Subsequently, the daemons force peer processes to redirect connections.

The data sent between processes always goes through an additional layer built of wrappers of the system calls. Inside the wrappers, some control information is changed but the communication is performed by the original TCP/IP stack (see Fig. 5).

In order to check the library, three groups of tests have been worked out. The first group was designed to check if the planned functionality of the msocket is realized whereas the second group focuses on measuring the level of overhead induced by the msocket. Finally, the last one was developed to test the stability of the socket connection during the migration.

To verify the functionality, the following tests have been performed: creation of the virtual socket, establishing connection before the migration, establishing connection after migration(s), establishing connection during migration, migration with connection in use. Overhead tests measure the time spent

Prepared using cpeauth.cls

on executing the code of this library. The generic applications used in tests are a typical producer–consumer and ping-pong programs.

The library is written in C and was tested on Linux systems.

We are working on a prototype implementation of MPI (*mpich*) on top of the msocket library. This research is aimed at the final test that the design presented here has a broader application area.

INTERACTIVE DISTRIBUTED SIMULATION

Interactive simulation environments are systems that combine simulation, data presentation and interaction capabilities that together allow users to explore the results of computer simulation processes and influence the course of these simulations at run-time. The goal of these interactive environments is to shorten experimental cycles, decrease the cost of system resources and enhance the researcher's abilities for the exploration of data sets or parameter spaces.

The conceptual idea of allowing users to interactively steer a computation while it is running is not new and examples of applications that benefit from it are abundant [23, 24, 25]. However, very often these applications provide *ad hoc* solutions that are very specific to the problem at hand. Here, we investigate the issues that are involved with building an interactive computational steering environment in an attempt to improve the synergy between computational simulation and interactive exploration through a generalised architecture. This architecture is currently being validated by analysis of a prototypical case study of *simulated abdominal vascular reconstruction*.

Exploration environments

In many scientific computing problems, the complexity of both the simulation and the generated data is too vast to analyse analytically or numerically. For these situations, exploration environments provide essential methods to present and explore the data in a way that allows a researcher to comprehend the information it contains. Exploration environments combine presentation and interaction functions into one system to allow exploration of large data spaces. These data spaces may originate from data acquisition devices or represent results from computer simulations. In our research we discriminate between static and dynamic exploration environments.

In static exploration environments (SEE), the data presented to the user is time invariant; once the data is loaded into the environment, the user is presented with a visual representation of this data. Interaction methods are provided to change the visualization parameters interactively in order to get the best view to gain understanding. The data itself, however, does not change.

An important step towards a successful exploration environment is to involve the researcher into the presentation as much as possible, thereby increasing the researcher's level of awareness [26]. To achieve this, an exploration system needs the following, often conflicting capabilities:

- High quality presentation – The most common method to provide insight in large multi-dimensional data sets is to represent data as visual constructs that present quantitative and relational aspects to the observer in an apprehensive manner. Many scientific visualization environments are now available that provide means of efficiently achieving this.
- High frame rate – While the capabilities of modern graphical hardware allow increasingly complex images to be rendered with relative ease, the level of detail in the presentation should

Prepared using cpeauth.cls

be minimized to avoid information clutter and achieve high frame rates (a compromise is often necessary).

- Intuitive interaction – A prerequisite of a successful SEE is that a sufficiently rich set of interaction methods is provided that allows a user to extract both qualitative and quantitative knowledge from the data sets. An unfortunate side effect of increasingly richer sets of interactive methods is that user-friendliness is compromised, so careful consideration is required during user-interface design.
- Real-time response – Some delay will always occur between the moment a user interacts with a presentation and the moment that the results are visible. This is caused by low tracking rates of input devices, (re-)computations, communication delays or temporary reduced availability of computational or network resources. To attain accurate control over the environment and to avoid confusion with the user, the amount of lag in an exploration system should be minimized.

Provided these capabilities are carefully considered, such environments are well suited for the exploration of *static* multi-dimensional data sets [27].

Static exploration environments can be customized to observe iteratively updated data sets produced by "living" simulations. When interaction with the simulation is also allowed, however, we speak of *dynamic* exploration environments and radically different considerations come into play, as we describe in the next section.

Design considerations for the construction of dynamic exploration environments

Dynamic exploration environments (DEE) extend the previously described static model in that the information provided to the user is regenerated periodically by an external process, in our case

Prepared using cpeauth.cls

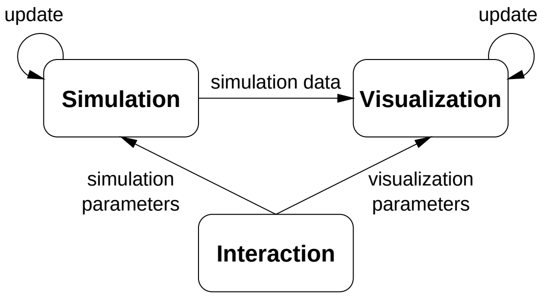

Figure 6. Schematic representation of a dynamic exploration environment (DEE).

a computer simulation. Here, the environment is expected to provide (1) a reliable and consistent representation of the results of the simulation *at that moment* and (2) mechanisms enabling the user to change parameters of the external process (i.e. simulation) (see Fig. 6).

Dynamic environments have additional requirements over static environments as any interaction influences both the visualization and the simulation environment. In the sequel we address some of the functionalities required to develop generic dynamic exploration environments [28]. We start with a brief description of the specific time management aspects in DEE and present a top-down description of the associated additional requirements in such systems.

Time management

An important issue in a DEE is time management. Time management deals with the exchange of time stamped information between components. For a DEE, the four most time demanding components are; the simulation environment, the visualization modules, the rendering layer and the explorer (i.e. the user).

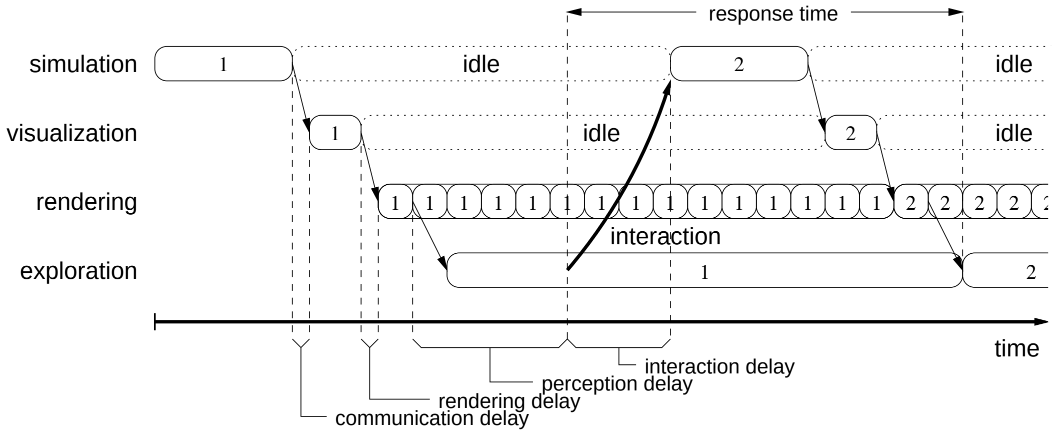

Figure 7. Time frames and delays in a lock-step interactive dynamic exploration environment.

Figures 7 and 8 show time frame diagrams illustrating the advancement of time, under two different time management strategies; *lock-step* and *asynchronous*. Time frames are illustrated by rounded boxes. The gaps between time frames on a same level represent the *idle* time of the component on that level. The gaps between time frames on neighbouring levels represent the *delays* that occur between the time one component is done with a time frame and the next component starts working on it. These delays are delineated at the bottom of the figure.

Fig. 7 shows a time frame diagram in a lock-step interactive dynamic exploration environment. In this strategy, the simulation is allowed to advance only if the user explicitly tells the environment it is alright to do so. While the user is exploring the results rendered by the graphics system, the simulation and visualization modules sit idle. In situations where a single simulation, visualization and rendering time frame takes a negligible amount of time, this strategy may be perfectly adequate since the user will see the result of his interaction in short notice. However, if the these time frames are long, it may take

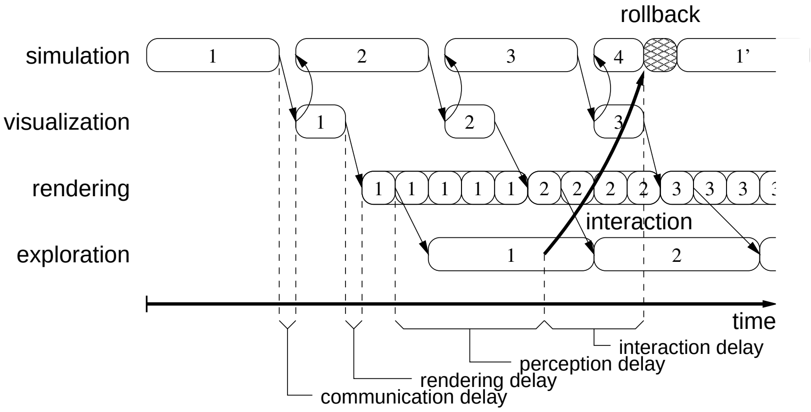

Figure 8. Time frames and delays in an asynchronous interactive dynamic exploration environment.

a long time before the result of an interaction is shown. This often leads to an unusable environment and confusion with the user.

The time required before simulation updates are presented to the user (i.e. the length of a time frame on the exploration level) can be shortened by allowing the simulation to run asynchronously from the rest of the environment. Fig. 8 shows a time frame diagram in an asynchronous environment. In an asynchronous system, different components are allowed to advance when they have finished processing and communicating the current time frame. Different components may therefore execute at different time scales. In addition, these time scales are mostly nondeterministic because of hardware, software and human imposed delays. As a consequence, time delays occur as the output generated by one component can not be accepted immediately by another component for processing. When components depend on the output of an increasing number of other components, the time frame that is processed by "later" components fall further apart. This has a causality consequence for the user who explores the

Prepared using cpeauth.cls

final component in the environment and therefore interacts with components at a much "earlier" time compared to what is being processed by a simulation at the same wall-clock time. Time management is responsible for detecting and resolving this causality violation. Methods for resolving time causality problems have been investigated within our group [29].

Interaction

A DEE provides the opportunity to interact with a living simulation. This interaction can take any form; from typed input for simple types of interaction via graphical user interfaces to fully immersive virtual environments. The main feature of immersive virtual environments over other graphical user interfaces is that user-centered stereoscopic images are presented to a user rather than visualization centered three-dimensional (3D) projections. Interaction with a virtual environment is a key issue, especially in an interactive simulation environment. The following types of interaction are deemed mandatory.

- Object interaction – An "object" is defined here as a visual entity that is in the center of interest to an end-user. These objects are representations of data sets or simulation results but also "widgets" (menus, buttons, sliders, etc.). An object has attributes associated with it such as position, scale, level, state, etc. Object interaction is concerned with changing these attributes.
- Navigation and wayfinding – Navigation provides users with methods to move beyond the confinements of the VE's physical dimensions. Objects beyond the VE come into reach by moving the user towards them. Note how this concept places the user of the VE in the center of this type of interaction; the user is transported from one place to another while the objects remain where they are. Wayfinding is a relatively new concept in VE applications and provides the user with a reference on where he is in a virtual environment [30].

- Probing – Although visual presentations allow researchers to qualitatively analyse their data, an instrument for obtaining quantitative information from the visualization is a valuable asset. Previously, we have developed an architecture that allows researchers to probe visual presentations in order to obtain quantitative information [31].

Coordination: Intelligent Agents

Since the various components in a DEE are independent processes, some means of coordination between them is required to allow a complex environment of this kind to be used. Especially in the case of interactive simulations, interaction involves not only the visualization part, but also the simulation part. For software engineering and efficiency reasons, it is reasonable to move the processing for those general interactions into independent modules, and let them be reusable to all components that need these interactions. Our approach to this is through Intelligent Agents (IAs).

IAs are software modules with the capability of performing three functions: (1) perceiving state changes in the environment through the use of monitors, (2) taking actions that affect conditions in the environment and (3) reasoning to interpret perceptions, solve problems, draw inference, and determine actions [32]. Agents execute autonomously, interfering minimally with the rest of the environment, apart from communicating with other agents or the user.

Feedback is generated when the agent has inferred a solution to a problem, or when it has prepared suggestions based on the current running context. This feedback is presented as information to the user, or the information is sent to environment components. Depending on the permission settled beforehand, an agent provides feedback without affecting the working status of the environment, or makes changes to the environment based on its reasoning.

We currently use agents to provide feedback to the user concerning the state of a simulation (e.g. accuracy of the simulation, time to completion, convergence rate), user interaction (including speech recognition and synthesis). In the near future we will develop agents for feature extraction (e.g. determining the geometric skeleton of objects, detecting eddies in flow domains) and providing assistance (context sensitive help).

Distributed execution environment

Although the capabilities of modern computer systems are nearing the requirements for performing both simulation and visualization tasks on the same machine, some performance increase may be attained by running these tasks on dedicated computing platforms. For example, simulation applications may perform better on dedicated hardware such as high performance computing machinery, while state-of-the-art graphical systems are now available that are well suited for the visualization tasks. Moreover, a decomposition of the environment into separate communicating tasks facilitates implementation and allows more control over the performance of the system as a whole.

Especially in the case of distributed environments, some means of job control is required that starts/stops the execution of the different components of the environment on the various computing platforms. In many organizations this system also needs to allocate the required resources prior to execution (for example in the case of batch execution systems). GLOBUS is one such software infrastructure for computations that integrates geographically distributed computational and information resources [33].

In distributed systems, components execute on different, possibly heterogeneous computing platforms. To be able to communicate data with each other, components provide access to their attributes which can then be made available to other components. In heterogeneous computing environments, the attributes often have to be converted into different representation format. Furthermore, in many circumstances not all components in an environment will participate in a communication. For these situations a publication and subscription mechanism needs to be provided that limits communication to members of a restricted group. Message passing systems such as PVM and MPI support this data distribution facility for the most part. In general, however, these systems do not support the construction of asynchronous systems as they restrict the application programmer to the Single/Same Program Multiple Data (SPMD) paradigm, which in DEEs is troublesome.

Attribute ownership

The behaviour of individual components in the environment is defined by one or more attributes (or parameters) which together define the state of that component. In a distributed system, attribute changes (which can be considered events, for example as a result of user interaction) should only be performed by a component that *owns* the attribute to avoid race conditions. In some cases it may be necessary to transfer ownership so that attributes can be changed by other components (for example in a collaborative environment where multiple users manipulate the same components).

Runtime support system

From the considerations described in the previous sections, it becomes clear that a generic framework to support the different modalities is required. In our research we have chosen for the "High Level Architecture" as a suitable architecture for constructing a DEE. The High Level Architecture (HLA) aims to establish a common architecture for simulation to facilitate interoperability among simulations and promote the reuse of simulations and their components [34, 8]. As a successor to the DIS (Distributed Interactive Simulation) protocol, HLA provides a robust architecture with which distributed discrete event and other types of simulations can be designed.

HLA simulations are composed of "federates", these are gathered to form "federations", interaction between federations is controlled by a "Runtime Infrastructure" (RTI). HLA systems comprise *rules* which govern how federates and federations are constructed; an *interface specification* which governs how federates interact with the RTI; and an Object Model Template which provides a method for documenting key information about simulations and federations.

HLA provides solutions to many of the problems and issues described in the previous sections. Specifically, HLA allows data distribution across heterogeneous computing platforms (including message groups), supports a flexible attribute publish/subscribe and ownership mechanism and offers several methods to do time management.

CASE STUDIES

Vascular reconstruction: a full-blown case study

The design considerations described in the previous section cover the issues that are involved with building a DEE. The architecture is validated by analysis of a prototypical case study of *simulated abdominal vascular reconstruction*.

The application we have chosen as a test case combines visualization, simulation, interaction and real-time constraints in an exemplary fashion. By a detailed analysis of the spatial and temporal characteristics of the test case we attempt to recognize generic elements for the design of a computational steering architecture. We begin with a description of the test case.

Simulated abdominal vascular reconstruction

Vascular disorders in general fall into two categories: *stenosis*, a constriction or narrowing of the artery by the buildup over time of fat, cholesterol and other substances in the vascular wall, and *aneurysms*, a ballooning-out of the wall of an artery, vein or the heart due to weakening of the wall. Aneurysms are often caused or aggravated by high blood pressure. They are not always life-threatening, but serious consequences can result if one bursts.

A vascular disorder can be detected by several imaging techniques such as X-ray angiography, MRI (magnetic resonance imaging) or computed tomography (CT). Magnetic resonance angiography (MRA) has excited the interest of many physicians working in cardiovascular disease because of its ability to non-invasively visualize vascular disease. Its potential to replace conventional X-ray angiography that uses iodinated contrast has been recognized for many years, and this interest has been stimulated by the current emphasis on cost containment, outpatient evaluation, and minimally invasive diagnosis and therapy [35].

A surgeon may decide on different treatments in different circumstances and on different occasions but all these treatments aim to improve the blood flow of the affected area [36]. Common options include *thrombolysis* where a blood clot dissolving drug is injected into, or adjacent to, the affected area using a catheter; *balloon angioplasty* and *stent placement* which is used to widen a narrowed vessel by means of a inflatable balloon or supporting framework; or *vascular surgery*. A surgeon resorts to vascular surgery when less invasive treatments are unavailable. In *endarterectomy* the surgeon opens the artery to remove plaque buildup in the affected areas. In vascular bypass operations, the diseased artery is shunted using a graft or a healthy vein harvested from the arm or leg.

The purpose of vascular reconstruction is to redirect and augment blood flow or perhaps repair a weakened or aneurysmal vessel through a surgical procedure. The optimal procedure is often obvious but this is not always the case, for example, in a patient with complicated or multi-level disease. Pre-operative surgical planning will allow evaluation of different procedures *a priori*, under various physiologic states such as rest and exercise, thereby increasing the chances of a positive outcome for the patient [37].

What is needed?

The described test case contains all aspects of an interactive dynamic exploration environment that are of consequence in the construction of a generic dynamical computational steering architecture. Our aim is to provide a surgeon with an environment in which he/she can try out a number of different bypass operations and see the influence of these bypasses. The environment needs the following:

- An environment that shows the patient under investigation with his infliction. A patient's medical scan is 3D, so to obtain best understanding on the nature of the problem, the surgeon should be able to look at his specific patient data in 3D, using unambiguous visualization methods.
- An environment that allows the surgeon to *plan* a surgical procedure. Again, this environment should allow interaction in a 3D world, with 6 degrees of freedom (DOF). The CAVE environment allows us to interact with 3D computer generated images using 6 DOF interaction devices [9]. Note that *visual* realism is not the primary goal here; what is more important here is *physical* realism, and then only of particular issues in fluid flow, as discussed later.

Prepared using cpeauth.cls

- An environment that shows the surgeon the effect of his planned surgical procedure. As the aim of the procedure is to improve the blood flow to the affected area, the surgeon must have some means to compare the flow of blood before and after the planned procedure. This requires the following:

  - a *simulation* environment that calculates *pressure*, *velocity* and *shear stress* of blood flowing through the artery,
  - a *visualization* environment that presents the results of the simulation in a clear unambiguous manner,
  - an *exploration* environment that allows the researcher to inspect and probe (qualitatively and quantitatively) the results of the simulation (e.g. means for performing measurements, annotate observations, releasing tracer particles in the blood stream, etc.).

All this should be *interactive*, or in other words; it should be fast enough such that a surgeon does not have to wait for the simulation results.

Implementation of a dynamic exploration environment

Parts of the components mentioned in the previous section have already been implemented in the course of previous projects. Others require minor adaptations to fit into our dynamic exploration architecture. In the following subsections we will briefly discuss the current status of the visualization and exploration environment, the interaction environment, the simulation environment and the middleware that combines these together.

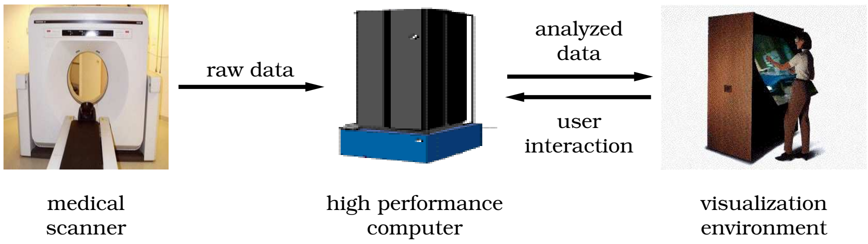

Figure 9. The *Virtual Radiology Explorer* (VRE) environment allows medical data from hospitals to be preprocessed on HPC systems for 3D visualization. High speed networking initiatives such as the *GigaPort* project [40] allow hospitals to make interactive use of HPC visualization techniques for patient diagnostics.

VRE: immersive static exploration

We have previously built a SEE called the *Virtual Radiology Explorer* (VRE [38, 39]) which is capable of visualizing medical CT and/or MRI data in 3D (see also Fig. 9). 3D data sets acquired with computed tomography (CT) or magnetic resonance imaging (MRI) are often displayed and evaluated from various perspectives or at different levels, including sets of single slices, stack mode (cine loop) interactive representation of sets of slices, or multi-planar reformation (MPR) represented as single slices or interactive cine loops. Despite the increased possibilities of acquiring data, clinical use of 3D rendering has been hampered by insufficient computing capacity in the clinical environment.

An example of the clinical use of 3D rendering is simulated endoscopy‡. Simulated endoscopy has several advantages over mechanical endoscopy (shorter acquisition times, increased patient comfort, higher cost-effectiveness, no complications of endoluminal instrumentation, field-of-view extending

‡*Endoscopy* is a diagnostic procedure where an instrument is used to visualize the interior of a hollow organ.

beyond the surface). In addition, simulated endoscopy can be used in virtual spaces that cannot at all, or only after violation of normal anatomical structures, be reached by mechanical (endo-)scopy.

From a clinical perspective, there is a demand in community hospitals to make an environment available suitable for the interactive rendering and interactive matching of, and switching between the above described data sets with an emphasis on simulated endoscopy. The VRE environment provides various such methods for the visualization of medical scans, including volume rendering, surface rendering and interactive clipping and surface mapping techniques. Mechanisms have been added that allow the VRE environment to be run in a CAVE or on an ImmersaDesk. The ImmersaDesk allows the VRE environment to be used in the radiology department.

VRE+

VRE+ extends VRE with methods that allow dynamic exploration. Various methods are added to visualize the results of a simulation while it is running. An intelligent agent system is integrated that constantly monitors a user's actions and provides feedback to the user. Currently, we have implemented a speech recognition agent which enables users to control the environment using hands and voice simultaneously. A second agent monitors the position of the user when using the GEOPROVE probing system and provides feedback on the accuracy of the measurements. Another agent monitors the state of the simulation environment and provides feedback on the state of the simulation.

For the planning part, the VRE+ environment is extended with means to "draw" a surgical procedure using a "grid editor", as will be described shortly.

Fluid flow simulation: the lattice-Boltzmann method

The lattice-Boltzmann method (LBM) is a mesoscopic approach for simulating fluid flow based on the kinetic Boltzmann equation [41]. In this method fluid is modeled by particles moving on a regular lattice. At each time step, particles propagate to neighbouring lattice points and re-distribute their velocities in a local collision phase. This inherent spatial and temporal locality of the update rules makes this method ideal for parallel computing [42]. During recent years, LBM has been successfully used for simulating many complex fluid-dynamical problems, such as suspension flows, multi-phase flows, and fluid flow in porous media [43]. All these problems are quite difficult to simulate by conventional methods [44, 45].

The data structures required by LBM (cartesian grids) bare a great resemblance to the grids that come out of CT and MRI scanners. As a result, the amount of preprocessing can be kept to a minimum which reduces the risk of introducing errors due to data structure conversions. In addition, LBM has the benefit over other fluid flow simulation methods that flow around (or through) irregular geometries (like a vascular structure) can be simulated relatively easy. Yet another advantage of LBM is the possibility to calculate the shear stress on the arteries directly from the densities of the particle distributions [46]. This may be beneficial in cases where we want to interfere with the simulation while the velocity and the stress field are still developing, thus supporting fast data-updating given a proposed change in simulation parameters from the interaction modules.

Lattice-Boltzmann grid generation and editing

As mentioned earlier, the basic structure of the grids used in LBM bare great resemblance to the medical scans obtained from a patient. To convert the medical scans into LBM grids, the raw data from the medical scanner is first segmented so that only the arterial structures of interest remain in the data set. The contrast fluid injected into the patient in an MRA scan provides sufficient contrast in the vascular structures to do this quite efficiently. The segmented data set is then converted into a grid that can be used in LBM; boundary nodes, inlet nodes and outlet nodes are added to the grid using a variety of image processing techniques.

A surgical procedure is simulated through the use of a 3D grid editor. This system allows a user to interactively add and/or delete areas in the LBM grid corresponding to the procedure that is simulated. Similar grid generation techniques as described above are used to ensure the grids comply to the demands imposed by LBM.

Middleware

The different components involved in our interactive simulation system are shown in Fig. 10. As can be seen from this figure, the visualization & exploration system runs on a different system (a CAVE) than the simulation system (a cluster of Linux nodes). HLA is used as a middleware layer to connect these components together. By using HLA, the different components can run asynchronously while spatial and temporal effects can be controlled. In addition, HLA provides attribute ownership management and does efficient data distribution between heterogeneous (if needed) systems.

CONCLUSIONS AND PERSPECTIVES

This paper presents the results of a research in the field of computing environments which was carried out in the framework of the Dutch Polder computing initiative.

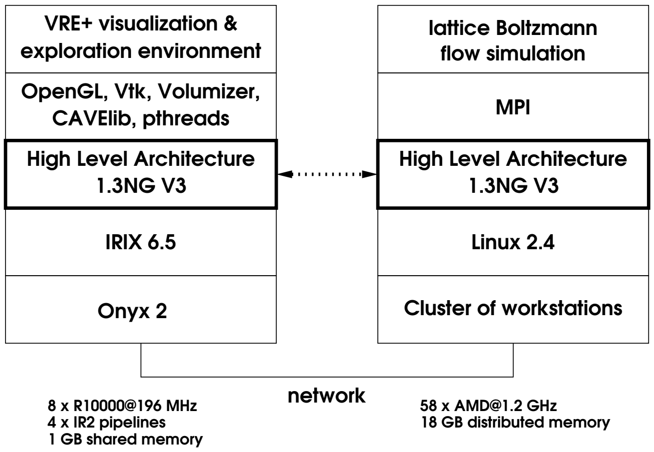

Figure 10. The visualization & exploration environment (on the left, running in a CAVE) and the simulation system communicate via the High Level Architecture (HLA).

In a couple of years, the hardware infrastructure in the form of the Distributed ASCI Supercomputer enabled us to develop an advanced, experimental software environment. This environment integrates a number of programming tools and libraries into a system – a universal problem solving environment.

This PSE makes it possible to run efficiently distributed applications which are characterised by the presence of a human in the interaction loop and by essential role of the visualization.

The experience we have acquired with the application designed for the planning of surgical procedures can be relatively easily extended into on-line decision support system to control flooding or air pollution.

Currently we are going to extend our research into more heterogeneous environment and to make our PSE available on grids where middleware layer will be built on top of Globus.

ACKNOWLEDGEMENTS

The GRAPE-4 special processor boards were kindly made available by Jun Makino of the University of Tokio.

REFERENCES

1. van Halderen AW, Overeinder BJ, Sloot PMA, van Dantzig R, Epema DHJ, Livny M. Hierarchical resource management in the Polder metacomputing initiative. *Parallel Computing* 1998; **24**(12/13):1807–1825.
2. Sloot PMA. Simulation and visualisation in medical diagnosis: perspectives and computational requirements. In *Advanced Infrastructures for Future Healthcare,* Marsh A, Grandinetti L, Kauranne T (eds.). IEEE-EMBS Press, 1999.
3. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan-Kaufmann, 1999.
4. Bal HE, Bhoedjang RAF, Hofman RFH, Jacobs CJH, Kielmann T, Maassen J, van Nieuwpoort R, Romein J, Renambot L, Rühl T, Veldema V, Verstoep K, Baggio A, Ballintijn G, Kuz I, Pierre G, van Steen M, Tanenbaum AS, Doornbos G, Germans D, Spoelder H, Baerends EJ, van Gisbergen S, Afsarmanesh H, van Albada GD, Belloum ASZ, Dubbeldam D, Hendrikse ZW, Hertzberger LO, Hoekstra AG, Iskra KA, Kandhai BD, Koelma DC, van der Linden F, Overeinder BJ, Sloot PMA, Spinnato PF, Epema DHJ, van Gemund A, Jonker PP, Radulescu A, van Reeuwijk C, Sips HJ, Knijnenburg PMW, Lew M, Sluiter F, Wolters L, Blom H, van der Steen A. The Distributed ASCI supercomputer project. *ACM Operating Systems Review* 2000; **34**(4):76–96.
5. Ridge D, Becker D, Merkey P, Sterling T. Beowulf: Harnessing the power of parallelism in a pile-of-pcs. In *Proc. of the 1997 IEEE Aerospace Conference,* vol. 2. IEEE, 1997; 79–91.
6. Bodem NJ, Cohen D, Felderman RE, Kulawik AE, Seitz CL, Seizovic JN, Su W. Myrinet: a Gigabit-per-second Local Area Network. *IEEE Micro* 1995; **15**(1):29–36.
7. Iskra KA, van der Linden F, Hendrikse ZW, Overeinder BJ, van Albada GD, Sloot PMA. The implementation of Dynamite – an environment for migrating PVM tasks. *ACM Operating Systems Review* 2000; **34**(3):40–55.
8. Defense Modeling and Simulation Office (DMSO). High Level Architecture (HLA) homepage. http://hla.dmso.mil/ [19 July 2001].
9. Cruz-Neira C, Sandin DJ, DeFanti TA. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. *Computer Graphics* (SIGGRAPH '93 Proceedings) 1993; **27**:135–142.
10. Makino J, Taiji M. *Scientific Simulations with Special-Purpose Computers.* Wiley, 1998.
11. Spinnato PF, van Albada GD, Sloot PMA. Performance of N-body codes on hybrid machines. *Future Generation Computer Systems* 2001; **17**(8):951–959.

12. Bhoedjang RAF, Rühl T, Bal HE. Design issues for user-level network interface protocols on Myrinet. *IEEE Computer* 1998; **31**(11):53–60.
13. Bhoedjang RAF, Rühl T, Bal HE. Efficient multicast on Myrinet using link-level flow control. In *Proc. of the 1998 Int. Conf. on Parallel Processing.* Minneapolis, MN, 1998; 381–390.
14. Bhoedjang RAF, Verstoep K, Rühl T, Bal HE, Hofman RFH. Evaluating design alternatives for reliable communication on high-speed networks. In *Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-9).* Cambridge, MA, 2000; 71–81.
15. Bal HE, Bhoedjang R, Hofman R, Jacobs C, Langendoen K, Rühl T, Kaashoek MF. Performance evaluation of the Orca shared object system. *ACM Transactions on Computer Systems* 1998; **16**(1):1–40.
16. Epema DHJ, Livny M, van Dantzig R, Evers X, Pruyne J. A worldwide flock of Condors: load sharing among workstation clusters. *Future Generation Computer Systems* 1996; **12**(1):53–65.
17. Santoso J, van Albada GD, Sloot PMA, Nazief BAA. Simulation of hierarchical resource management for meta-computing systems. *International Journal of Foundations of Computer Science (IJFCS)* 2001; (accepted).
18. Pruyne J, Livny M. Managing checkpoints for parallel programs. In *Proc. of IPPS 2nd Workshop on Job Scheduling Strategies for Parallel Processing,* LNCS vol. 1162. Springer-Verlag, 1996; 140–154.
19. Iskra KA, Hendrikse ZW, van Albada GD, Overeinder BJ, Sloot PMA, Gehring J. Experiments with migration of message passing tasks. In *Proc. of the 1st IEEE/ACM Int. Workshop on Grid Computing (GRID 2000),* LNCS vol. 1971. Springer-Verlag, 2000; 203–213.
20. Software by SCS.
    http://www.science.uva.nl/research/scs/Software/ [16 July 2001].
21. Robinson J, Russ SH, Flachs B, Heckel B. A task migration implementation of the Message Passing Interface. In *Proc. of the 5th IEEE International Symposium on High Performance Distributed Computing.* IEEE CS, 1996; 61–68.
22. Bubak M, Zbik D, van Albada GD, Iskra KA, Sloot PMA. Portable library of migratable sockets. In *Proc. of the SGI Users' Conference 2000.* Kraków, Poland, 2000; 175–184.
23. Chen JX, Rine D, Simon HD. Advancing interactive visualization and computational steering. *IEEE Computational Science & Engineering* 1996; **3**(4):13–17.
24. Johnson CR, Parker SG. Applications in computational medicine using SCIRun: A computational steering programming environment. In *Supercomputer '95.* Springer-Verlag, 1995; 2–19.

25. Roy TM, Cruz-Neira C, DeFanti TA, Sandin DJ. Steering a high performance computing application from a virtual environment. *Presence: Teleoperators and Virtual Environments* 1995; **4**(2):121–129.

26. Bryson S. Virtual reality in scientific visualization. *Communications of the ACM* 1996; **39**(5):62–71.

27. Belleman RG, Kaandorp JA, Sloot PMA. A virtual environment for the exploration of diffusion and flow phenomena in complex geometries. *Future Generation Computer Systems* 1998; **14**(3-4):209–214.

28. Belleman RG, Sloot PMA. The design of Dynamic Exploration Environments for computational steering simulations In *Proc. of the SGI Users' Conference 2000.* Kraków, Poland, 2000; 57-74.

29. Overeinder BJ, Sloot PMA. Extensions to time warp parallel simulation for spatially decomposed applications. In *Proc. of the 4th United Kingdom Simulation Society Conference (UKSim 99).* Cambridge, UK, 1999; 67–73.

30. Elvins TT. Virtually lost in virtual worlds – wayfinding without a cognitive map. *Computer Graphics* 1997; **31**(3):15–17.

31. Belleman RG, Kaandorp JA, Dijkman D, Sloot PMA. GEOPROVE: Geometric probes for virtual environments. In *High-Performance Computing and Networking (HPCN'99),* LNCS vol. 1593. Springer-Verlag, 1999; 817–827.

32. Hayes-Roth B. An architecture for adaptive intelligent systems. *Artificial Intelligence* (special issue on agents and interactivity) 1995; **72**:329–365.

33. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *Intl. J. Supercomputer Applications* 1997; **11**(2):115–128.

34. *High Level Architecture Run Time Infrastructure Programmer's Guide* (1.3 version 7). Defense Modeling and Simulation Office (DMSO), Department of Defense, US, 1999.

35. Yucel EK, Anderson CM, Edelman RR, Grist TM, Baum RA, Manning WJ, Culebras A, Pearce W. Magnetic resonance angiography (update on applications for extracranial arteries). *Circulation* 1999; **100**:2284–2301.

36. Ku DN. Blood flow in arteries. *Annu. Rev. Fluid Mech.* 1997; **29**:399–434.

37. Taylor CA, Hughes TJR, Zarins CK. Finite element modeling of three-dimensional pulsatile flow in the abdominal aorta: Relevance to atherosclerosis. *Annals of Biomedical Engineering* 1998; **26**:975–987.

38. Durnford L. Virtual reality: More than just a game. *Radio Netherlands Wereldomroep*, May 14th 1999. http://www.rnw.nl/science/html/virtualreality990514.html [19 July 2001].

39. Versweyveld L. Exploring the medical applications of virtual reality techniques in the Dutch CAVE. *Virtual Medical Worlds*, March 2nd 1998. http://www.hoise.com/vmw/articles/LV-VM-04-98-13.html [19 July 2001].

40. Surfnet: Gigaport homepage.
http://www.gigaport.nl/en/en_index.html [19 July 2001].

41. Chen S, Doolen GD. Lattice Boltzmann method for fluid flows. *Annu. Rev. Fluid Mech.* 1998; **30**:329–364.

42. Kandhai D, Koponen A, Hoekstra AG, Kataja M, Timonen J, Sloot PMA. Lattice Boltzmann hydrodynamics on parallel systems. *Computer Physics Communications* 1998; **111**(1-3):14–26.

43. Koponen A, Kandhai D, Hellen E, Alava M, Hoekstra A, Kataja M, Niskanen K, Sloot P, Timonen J. Permeability of three-dimensional random fiber webs. *Physical Review Letters* 1998; **80**(4):716–719.

44. Kandhai D. *Large Scale Lattice-Boltzmann Simulations (Computational Methods and Applications).* PhD thesis, Universiteit van Amsterdam, Amsterdam, the Netherlands, 1999.

45. Kandhai D, Vidal D, Hoekstra A, Hoefsloot H, Iedema P, Sloot PMA. Lattice-Boltzmann and finite element simulations of fluid flow in a SMRX mixer. *Int. J. Numer. Meth. Fluids* 1999; **31**:1019–1033.

46. Artoli AM, Kandhai D, Hoefsloot HCJ, Hoekstra AG, Sloot PMA. The stress tensor in lattice Boltzmann simulations. Submitted for publication, 2001.

