AG-IVE: An Agent Based Solution to Constructing Interactive Simulation Systems

Z. Zhao, R.G. Belleman, G.D. van Albada, P.M.A. Sloot

Section Computational Science
Faculty of Science, Universiteit van Amsterdam
Kruislaan 403, 1098SJ Amsterdam, The Netherlands
{zhiming | robbel | dick | sloot}@science.uva.nl
http://www.wins.uva.nl/research/scs/

Abstract. An Interactive Simulation System (ISS) allows a user to interactively explore simulation results and modify the parameters of the simulation at run-time. An ISS is commonly implemented as a distributed system. Integrating distributed modules into one system requires certain control components to be added in each module. When interaction scenarios are complicated, these control components often become large and complex, and are often limited in their reusability. To make the integration more flexible and the solution more reusable, we isolated these control components out of the system's modules and implemented them as an agent framework. In this paper we will describe the architecture of this agent framework, and discuss how they flexibly integrate distributed modules and provide interaction support.

1. Introduction

An Interactive Simulation System (ISS) is a system that combines simulation and interactive visualisation and allows the users to explore the problem space and steer the execution paths of the simulator at runtime. For a complex simulation, putting a human expert in the simulation loop can reduce the parameter space. This can significantly shorten the experimental cycle and improve the efficiency of resource usage. During the last decade, interactive simulation systems have become an important research topic in the area of computational science [1,2,3,4].

In general, a minimum ISS has three basic modules: simulation, visualisation and interaction. The simulator computes and transfers data to the visualisation and interaction modules regularly; the simulation parameters can be manipulated in the interaction module. Simulators and visualisation programs often are inherently very complex. They are often developed on different platforms and with different software architectures, connected using e.g. a *Client-Server* paradigm [8]. Following the classification of paradigms for developing parallel and distributed simulation in [4], we can distinguish two principal approaches to constructing ISSs [5,6,7]. The first approach starts with building simulators using a high performance simulation engine, and then equips them with interaction capabilities. SPEEDS [5] is an example. The other uses a

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2329, pp. 693–703, 2002.

run time infrastructure software to interconnect federated modules. Protocols such as Distributed Interactive Simulation (DIS) [6] and High Level Architecture (HLA) [7] support this approach. The first approach often achieves a higher efficiency because of the specific development environment for the problem domain. The second can easily interconnect modules across platforms and places fewer restrictions on the internal implementation of individual simulators. It can exhibit more flexibility and reusability of components than the first.

In an ISS, all modules have to work together under certain constraints. In our work, we use *scenarios* to specify the relation between the required behaviours of the modules. Updating and distribution of the simulation states is one of the basic actions in an ISS. Depending on the interaction context in each scenario, the simulation state needs to be updated in different ways. Often, these constraints are incorporated into the simulator and visualisation modules. This is straightforward for the system design, but makes the system modules difficult to reuse. Separately implementing them in Intelligent Agents (IA) yields a more generic and portable solution.

In earlier publications [12,13], we have described the Virtual Environment (VE), as a device for scientific visualisation and user interaction. The Intelligent Virtual Environment (IVE) integrates simulators and interactive visualisation programs in a VE, and adds interactive simulation capabilities. In the IVE, agents are used to interconnect simulators and interactive visualisation programs; they are employed to customise the inter-module communication and provide support for module co-ordination and interaction. We implement these agents as a reusable framework. In this paper, we will first introduce the architecture of this AGent based Intelligent Virtual Environment (AG-IVE), next we discuss the issues of simulation state updates and switches between system scenarios. After that, we will present experimental results on the performance of a prototype of AG-IVE.

2. Intelligent Agents and Module Integration

Intelligent Agents and Agent oriented techniques have become a modern AI approach and have been applied in many problem domains, although many researchers do not agree on apparently trivial terminology details [14]. Depending on the problem domain and system design, agents vary in their functions. A simple reflex agent is only able to respond to events from its external world, a goal-oriented agent can make plans to organise its actions.

Rather than using Agent technology for solving specific problems, we aim to use it as an integrating concept, enhancing flexibility, portability and code re-use.

2.1 Agent Paradigm: AGent Based Intelligent Virtual Environment (AG-IVE)

AG-IVE is an agent-based solution to interconnecting simulators and interactive visualisation programs that tries to combine the advantages of both development approaches mentioned in the introduction. It aims to re-use available simulation kernels and visualisation programs and to integrate them in a portable way. In AG-IVE,

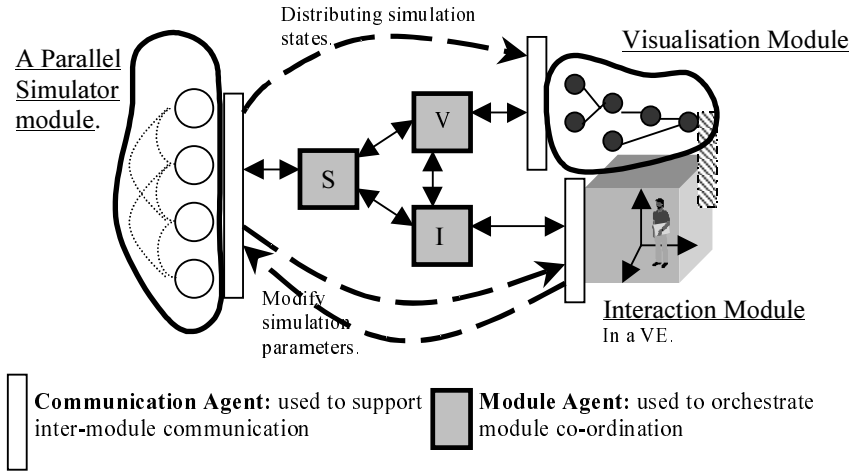

Fig. 1. An example of AG-IVE when it is used to interconnect a simulator and an interactive visualisation module

agents are used to federalise simulators and visualisation programs. A global run time infrastructure is employed to inter-connect them. Currently, the Run Time Infrastructure (RTI) [7] of the High Level Architecture (HLA) is used.

In AG-IVE, two types of agents are defined: "Communication Agents" (ComA) and "Module Agents" (MA). The ComAs hide the implementation details of the simulators and interactive visualisation programs, and provide them with an interface to plug into the global run time environment and interact with other modules. The MAs orchestrate the execution of modules (simulation, visualisation and interaction), and control scenario transitions of the whole system by co-operating with other agents. Fig. 1 shows how these agents can be used to integrate a system.

The example shows a parallel simulator and an interactive visualisation module. The visualisation module presents simulation results in a VE where a user can interactively study the data space. ComAs interface the modules to the RTI and transfer data between them (thick dashed lines). MAs co-operate with each other and the ComAs to control the behaviour of the system (double headed arrows).

2.2 Communication Agents

In AG-IVE, the global run time infrastructure is only used for communication between the modules; the simulators and interactive visualisation programs internally still use their original communication mechanisms. In each module, the processes needing inter-module communication are equipped with a ComA.

To minimise the influence on the execution of the host process and to make the integration easier, a ComA is implemented as a separate. The ComA is designed as an event processing architecture that contains four basic components: *a task generator, a task list, a task interpreter* and *a shared data object space* (Fig. 2). In a ComA, a task is the description of certain actions that the ComA should carry out. Only the task

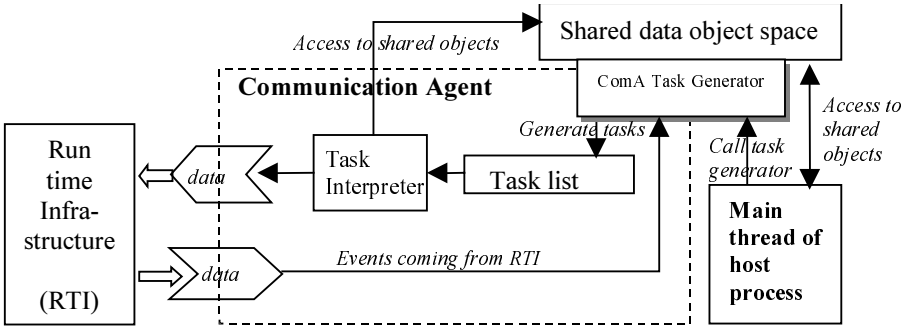

Fig. 2. Architecture of a communication agent

generator can create tasks; it can be called by either the host process or the ComA itself. For instance, the host process can cause the ComA to send data packages to the RTI, and the ComA can receive messages from other agents and react to them by creating certain tasks. Depending on the priority and the time tamp associated with each task, the task interpreter schedules tasks and executes them. This execution style hides the implementation details of the ComA from the host process, and keeps the internal functions of a ComA from being called directly.

In AG-IVE, a shared object space is used to exchange information between host process and the ComA. In the shared object space, all information is encapsulated as data objects. The structures of the data objects are defined as data classes. These are fully described in a globally accessible "Scenario Description file".

Internally, a ComA pre-defines a special kind of data class, the so called *Monitor*. During system creation, the host process can transfer access rights of its parameters to ComA by registering them as data items in the Monitor object. Therefore, the ComA can manipulate the execution of the host process via the Monitor.

Adding a ComA to a process inevitably requires certain code level adaptations. To simplify the integration and reduce the influence on the original software architecture, the interface for accessing shared objects and creating tasks of ComAs has been standardised. In order to minimise the influence on system performance, the function of a ComA is limited to transferring data and responding to external events.

The interface of each ComA, such as the set of events to which it can respond and the set of monitor parameters that it can use to manipulate its host process, is described in the Scenario Description file. From this, an MA can know how to manipulate the ComA.

2.3 Module Agents

Each module is associated with one MA, which interacts with all ComAs in that module. Depending on the type of ISS module, each Module Agent has different functions. Compared with ComAs, Module Agents are more sophisticated. An MA not only can react to the events from its external world like ComAs, but also can use its internal

world model to plan its actions beforehand. For each agent (ComAs and MAs), the external world is a dynamic environment, which includes all the other live agents and the running ISS modules.

Each MA is equipped with a Reasoning Module, which is used to process the observations and make rational action plans. The Reasoning Module utilises first-order logic to represent knowledge. The Reasoning Module has following functions:

1. Maintaining the internal world model. All the information gathered from the external world will be transferred to the Reasoning Module as Prolog terms.
2. Responding to queries. The other agents can lean the current status of a module by asking the MA of that module.
3. Making plans for given goals. In each scenario, MA has a long-term goal like optimising the execution of its module. The long-term goal is normally achieved by a number of sub-goals.

RTI is the only channel for cross module and inter-agent communication. To communicate with the other agents via the RTI, each MA has a ComA. MAs perceive the dynamic external world by observing and analysing the packages transferred over the RTI. ComAs and MAs co-operate with each other to realise functions like scenario control and state update.

2.4 Interaction and Scenario Switches

ISSs allow the user to explore the computational results and to interactively manipulate the parameters of the simulation. In AG-IVE, scenarios represent the contexts of certain interactions and their required constraints on system modules. Each scenario has one or more basic interaction objectives, for instance exploring simulation results, modifying simulation parameters or probing information. In different systems, the scenarios of the interaction may be different. In AG-IVE, the following basic interaction scenarios are distinguished.

1. Obtaining data from the simulator (sample, query);
2. Explore the visualisation of the data obtained from the simulation;
3. Adjust the simulation parameters (computational steering);
4. "VCR" control of the simulation progress (re-start, rollback, pause and resume the simulation execution).

To achieve the objective of a scenario, each system module has to run under certain constraints. These are adapted when scenarios are switched. The constraints often involve issues such as module synchronisation, intermediate state storage and data distribution. In AG-IVE a Scenario Description file specifies the capability of agents and their desired behaviour in each specific scenario. At run time, each MA reads information from the description file and adjusts the behaviour of ComAs in its module to fit the requirements of a scenario. The Federation File [7] needed by the HLA RTI is also generated from the scenario description file.

Switches between scenarios are often triggered by different mechanisms such as predefined interaction routes or by the user. In AG-IVE, the interaction MA monitors the user's behaviour during interaction and tries to identify the switching of scenarios.

After a scenario has switched the interaction MA broadcasts the new scenario to all the other MAs to inform them to adapt the behaviour of their ComAs.

2.5 Optimisation of State Updates

The simulator is required to export its states selectively and to distribute them efficiently to the visualisation and interaction modules. The visualisation module also needs to update in a rational manner, so that the user can obtain clear insights into the evolution of the simulation and its problem space. The content that will be transferred for visualisation and interaction normally is a subset of the state of the simulation, which can be mapped onto subsets of the local states in each simulation process

The update of simulation states has to address issues such as speed difference between simulators and visualisation, the user's perceptual delay, and the switch between different scenarios [10]. In different interaction contexts and running conditions, the way to export and distribute simulation states will not be the same. In AG-IVE, agents are used to optimise the state updates in each module.

Each MA monitors the running status of its host module, obtaining the internal information of each individual process by communicating with the ComA. The MAs share this information with each other and estimate the communication delay over the network by using the information from the other agents. Finally, the simulation MA can decide the best update rate for the simulation to export states.

Apart from making decisions with respect to resource utilisation, optimising the update of the simulation state also involves issues like efficiently distributing data objects, and storing intermediate simulation states during interaction. In the rest of this section, we will briefly introduce the considerations on these issues in AG-IVE.

For complex parallel simulators, the vectors of the simulation states are often very large, in the order of tens of megabytes. Efficiently distributing large size data objects is an important problem in distributed interactive simulation systems. In AG-IVE, the services from the lower level RTI are employed for the basic communication support. Based on these RTI services, AG-IVE provides a number of optional techniques to decrease the size of data such as: compression, data sub-sampling and segmentation. These techniques are implemented as *filters* of data objects, they can be used to process data objects before they are sent or after they are received. For this purpose, AG-IVE explicitly defines data items such as dimensions, processing methods and mapping algorithms in the simulation state data class. These techniques can significantly shorten the data transmission delay, and improve the update rate, but as most of them are lossy, they can be activated and de-activated at runtime.

2.6 Implementation and Current Status

AG-IVE has only been tested with parallel simulators that run synchronously and in time driven mode. The interface of RTI is based on the specification of Version 1.3. The visualisation programs are executed on either VEs or normal desktops. The Reasoning Modules of the MAs are implemented by Amzi Logical Server [19].

AG-IVE provides basic templates for integrating ComA and a host program. The ComA can be added as a separate thread or in the same thread as the original process. We have used this method in a fluid flow simulator and a visualisation program, and the basic integration only took a few hours. To make the construction of the scenario description file easier, a tool that can assist a user to specify agent actions and the constraints between them is under development.

3. Test Case: Intelligent Virtual Environment (IVE)

Currently the IVE is being used in a medical application. In surgery, operations such as vascular reconstruction can often be performed in different ways. The best treatment is not always obvious, or may be hard to discern because of complicated or multi-level diseases of the patient. Verifying an operation plan is a difficult task, even for expert surgeons. Computer simulation may help a surgeon to validate his treatment, but it is almost impossible to let a computer simulate all situations, because of the huge number of possible solutions. What this application tries to do is to put a human expert into the simulation cycle, and let him apply his expertise to confine the problem space. In this system, a simulation program simulates the patient's blood flow on a parallel computer system. A visualisation module presents the simulated results together with the body's geometrical information obtained from a medical scanner (such as CT or MRI). The visualisation and interaction modules run in an immersive virtual environment (CAVE) [15] where a user can study the results of the given treatment, and modify it if necessary. These systems are connected through a high performance network.

The fluid flow simulator [16] uses MPI [9] as its internal communication interface, and runs on a PC cluster [11]. The interactive simulation program, VRE (Virtual Radiology Explorer) runs in the CAVE environment, and uses shared memory to exchange information between visualisation and interaction processes. RTI Next Generation 1.3 Version 4 is used as the global run time infrastructure.

3.1 Package Size and the State Transfer Delay

In data distribution, the size of packages significantly influences the delay in their transmission. We use the term *transfer delay of the state objects* to represent the time from when one module starts sending objects until the other one finishes receiving them. We have measured the transfer delay of ComAs between two workstations that are not in the same cluster. Both workstations use a Pentium III processor and Linux Redhat 7.2 system; they are connected via a 100 Mbits/s network. Fig. 3 shows the average of 15 different measurements. The latency of the ComA is about 0.02s, its throughput is 47 Mbits/s. The throughput is half of the bandwidth, but it is very close to 50 Mbits/s; the performance we measured with a TCP socket program which makes use of the CAVERN library [18]. Internally RTI also uses sockets as its communication mechanism and we can see that the additional services provided by RTI have no large effect on communication, except for an additional latency for small messages.

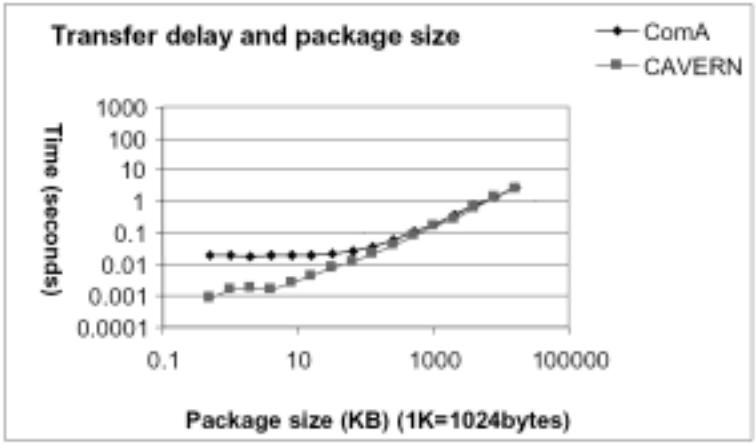

Fig. 3. Transfer delay versus package size for a ComA based method and for CAVERN

From Fig. 3 we also can see that the transfer delay increases rapidly when the package size exceeds 1 Mbyte. In order to reduce the size of packages, a number of techniques have been applied in AG-IVE, such as using general data compressing method like LZW [17] and sub-sampling simulation states. During experiments, we have seen there is not a single method that is effective for all types of simulator. For instance, for the fluid flow simulation, general compressing methods have not exhibited good performance. Normally, only less than 20% reduction of the whole simulation results can be achieved, and it needs more than 15 seconds to compress. Sub-sampling data content in simulation states will cause information loss, and requires certain code adaptation in the visualisation program. To select a proper option for distributing data objects, MAs can monitor all packages transferred between modules. MAs perform certain experiments at run time, and record the performance of each optional technique. Depending on the situation, MAs dynamically select the best option for ComA to transfer data. The user can also interactively enable or disable these functions.

3.3 ComAs and Their Penalty on System Performance

As a part of the ISS process, a ComA will require computing power from its host process. To evaluate the performance of the ComA and its influence on the original ISS processes, we have built four test configurations. The first and the second integrate ComA as separate threads but use user level scheduling vs. kernel level scheduling. In the third the ComA executes in the same thread as the simulator. The fourth one uses the CAVERN library to implement the communication between simulator and visualisation. In the first two cases, ComA and the simulation kernel run in parallel. In the last two cases, the computing of simulation kernel will be blocked during data export and distribution. All use the same simulator. Every time step, the simulation kernel exports the whole simulation state once, which is about 18 Mbytes. All these tests were executed using a single processor, two and four processors respectively. Each case has been measured 10 times, the average of each time step has been

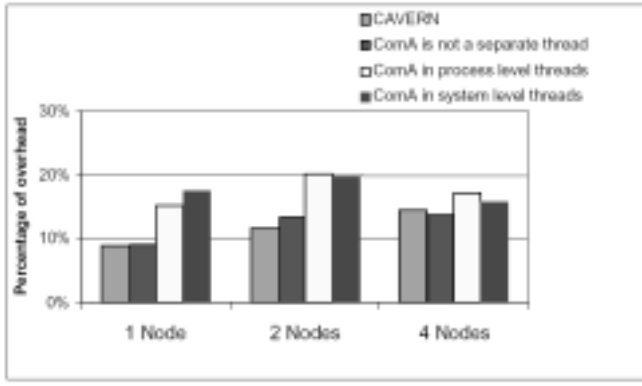

Fig. 4 Performance penalty of each test bed

compared to the original simulator. We use the ratio between the additional computing time required by these test runs and the time cost in the original simulator to obtain the penalty on system performance. Fig. 4 shows results. Using RTI, the local call back functions in each ComA only can be activated when they are ticked. When running in multithread style, in order to react on RTI events in real time, the tick function has also been called regularly even when a ComA has no task to process. That makes the system suffer higher performance penalty than the single thread integration (see Fig. 4). From the comparison with the CAVERN test case, we can say that the performance penalty of ComA is not prohibitive. As future work, we will look for more efficient scheduling strategies and improve the performance of ComAs when they are integrated as separate multithreads.

When more nodes are employed in computing, the local simulation state in each process becomes smaller, which accelerates computation. From the measurement, we found the difference between these four test beds becomes small when using more nodes. But on the other hand, the ComA in the visualisation module has to spend more time to collect and combine all sub-states. Fig. 5 shows the comparison of simulation states collection time between using ComA and using CAVERN. When using ComAs,

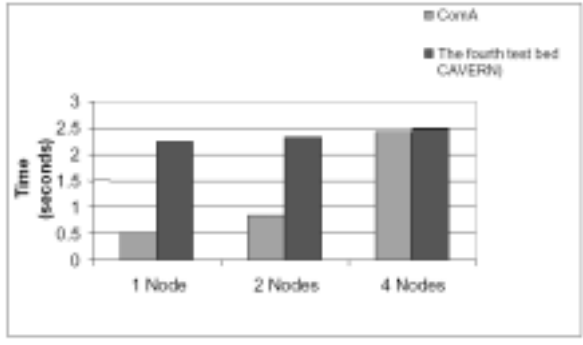

Fig. 5. Simulation sub-state collection times in the visualisation module

the RTI handles data distribution using broadcast; but when using CAVERN, data collection is done by querying data from the data channels which have been created when the system started. Because of this, we can see, from the Fig. 5, that when more ComAs are sending packages, the time cost for ComA to collect all data objects may become very large.

4. Conclusions and Future Work

As a research topic in both academic research and industrial applications, Intelligent Agents have been used in many areas. For us, developing one or two agents is not the final goal. The goal of our research is to find out how to build complex distributed interactive simulation systems and if Intelligent Agents really are the best solution. The AG-IVE project is still in progress. From the experiments that we have done, we can at least draw the following conclusions:

First, we can reuse available simulator and visualisation programs and interconnect them into one system. This is a practical way to develop interactive simulation system. Agents allow us to place essential control functionality outside the IVE modules, and thus to decrease their complexity. This makes the system more modular and increases the reusability of the modules. Second, using a Scenario Description file to explicitly represent shared object interface and interaction contexts, can make module integration more flexible and improve reusability. Third, using a run-time environment like the RTI can simplify the integration of distributed modules. The communication interface of HLA federates provides a simple and standard way to exchange information. This makes the development process of each module more independent. Finally, from the performance measurements that we have done, we can say that using Agents will not lead to large performance penalties.

In the near future, we will first finalise the basic services of AG-IVE, such as using interaction MA to handle scenario switches and providing interaction support. A user-friendly tool for specifying interaction scenarios will also be developed. After that we will also try to apply AG-IVE to the simulators like discrete event driven systems.

References

1. Franks S., Gomes F., Unger B., Cleary J.: State saving for interactive optimistic simulation; Proceedings of the 1997 workshop on Parallel and distributed simulation, (1997) 72-79
2. Salisbury, M. R: Command and Control Simulation Interface Language (CCSIL): status update. In Proceedings of the twelfth workshop on standards for the interoperability of defence simulations, (1995) 639-649
3. Ross P. Morley, van der Meulen Pieter S., Baltus P.: Getting a grasp on interactive simulation. In Raimund K. Ege, editor, Proceedings of the SCS Multiconference on Object-Oriented Simulation, Anaheim, CA, (1991) 151-157
4. Ferenci S., Perumalla K., Fujimoto R.: An Approach to Federating Parallel Simulators, in the Workshop on Parallel and Distributed Simulation, (2000)

5. Steinman, J.S.: SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete Event Simulation. International Journal on Computer Simulation, (1992) 251-286
6. IEEE Std 1278.1-1995, IEEE Standard for Distributed Interactive Simulation -- Application Protocols. New York, NY: Institute of Electrical and Electronics Engineers, Inc., (1995)
7. Defence Modelling and Simulation Office (DMSO). High Level Architecture (HLA) homepage. http://hla.dmso.mil/, (2001)
8. Perumalla, K.S., Fujimoto, R.M.: Interactive Parallel Simulations with the Jane Framework. Special Issue of Future Generation Computer Systems, Elsevier Science, (2000)
9. Message Passing Interface Forum. MPI-2: A Message-Passing Interface standard. The international Journal of Supercomputer Applications and High Performance Computing, 12 (1-2), (1998)
10. Belleman R.G., Sloot P.M.A.: The Design of Dynamic Exploration Environments for Computational Steering Simulations, in the Proceedings of the SGI Users' Conference 2000, pp. 57-74. Academic Computer Centre CYFRONET AGH, Krakow, Poland, ISBN 83-902363-9-7, (2000)
11. The Beowulf cluster at SARA http://www.sara.nl/beowulf/, (2001)
12. Belleman R.G., Sloot P.M.A.: Simulated vascular reconstruction in a virtual operating theatre, in Computer Assisted Radiology and Surgery (Excerpta Medica, International Congress Series 1230), Elsevier Science B.V., Berlin, Germany, (2001) 938-944
13. Belleman R.G., Kaandorp J.A., Dijkman D., Sloot P.M.A.: GEOPROVE: Geometric Probes for Virtual Environments, in the proceedings of High-Performance Computing and Networking (HPCN Europe '99), Amsterdam, The Netherlands, in series Lecture Notes in Computer Science. Springer-Verlag, Berlin, ISBN 3-540-65821-1, (1999) 817-827
14. Wooldridge, M., Jenings, N.: Intelligent Agents: Theory and Practice, in Knowledge Engineering Review, 10(2); (1995) pp. 115-152
15. Cruz-Neira C., Sandin D.J., DeFanti T.A.: Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. SIGGRAPH '93 Computer Graphics Conference, (1993) 135-142
16. Kandhai B.D.: Large Scale Lattice-Boltzmann Simulations (Computational Methods and Applications), PhD Thesis, Universiteit van Amsterdam, (1999)
17. Ziv J., Lempel A., A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, Vol. 23, No. 3, 337-343
18. CAVERNsoft, http://www.openchannelsoftware.org/projects/CAVERNsoft_G2/, (2001)
19. Amzi Prolog Logic Server, http://www.amzi.com/ (2002)