Efficient visualization of 3D medical scenes for remote interactive applications

Johan Montagnat, Eduardo Davila, Isabelle Magnin

Efficient visualization of 3D medical scenes for remote interactive applications

Johan Montagnat, Eduardo Davila, Isabelle E. Magnin
CREATIS, CNRS
INSA, Bât. B. Pascal, 20 av. A. Einstein, 69621 Villeurbanne Cedex, France
http://www.creatis.insa-lyon.fr/

Abstract

In this paper, we propose a software framework for developing interactive medical applications that execute on remote servers. The 3D visualization interface of the application can be disconnected from the computation core to allow rendering of the medical scene on the user display while the heavy computations are executed on a remote workstation or a computational grid. We introduce a software architecture designed for reducing the communication overhead between the remote and the local processes and for easing software development by offering transparent management of communications. Execution of a 2D and a 3D segmentation applications are demonstrated and performances are evaluated.

1. Context

Medical images produced by medical acquisition devices in hospitals and radiology centers represent tremendous amount of data. The annual production of digital images in Europe or the USA is estimated to thousands of petabytes. Grid technologies [8] offer a support to archive and process huge datasets. Automatic image analysis techniques are therefore increasingly needed to index, store, and query medical data. Grid-enabled medical application are promising as grids will allow to assemble very large medical databases with data spread over many acquisition sites. Moreover, grids provide a very large computing power suited to process large 3D or sequence of 3D images and to deal with compute intensive image analysis algorithms. Grid technologies already raised a large interest reflected by many research works on efficient scheduling [3], data management [5], and providing information in distributed systems. They are likely to play a key role in a near future for processing large amounts of data using a limited set of shared resources [4]. The new network generation with increased bandwidth, dedicated services, and guaranteed quality of services will boost management of large and distributed collections of data. However, remote execution of interactive applications require costly specific software development. This work proposes a software architecture to efficiently process, visualize, and interact with remote 3D medical images.

Due to the sensitive nature of medical data, medical image analysis algorithms often need human supervision to:

- solve responsibility issues when taking decisions based on data analysis results;
- allow a user intervention when the automated processing is inaccurate or erroneous.

Therefore, many interactive medical applications have been developed offering maximum automated assistance to free the user from tedious and error prone tasks while allowing a specialist to control the algorithm output. For instance, image segmentation and computer assisted diagnosis tools usually require user supervision. Other applications such as medical intervention simulation, augmented reality, and telemedicine also require user interaction and large computing power by nature.

Although grids are promising for many medical applications [14], interactive applications executions on remote systems are made difficult by the need to provide a user interface decoupled from the computing application. In this paper we discuss a software architecture designed to execute remote complex medical application requiring visualization of 3D scenes and interaction with the user. The main objectives of this work is to:

- Provide a multi-purpose, flexible, and transparent solution easing the development of various remote interactive applications without requiring the user to know about the internals of multi-processes programing and network transmission.
- Ensure efficient transmission of graphic data to allow the rendering of complex 3D medical scenes efficiently.
- Allow remote execution of compute intensive applications on remote workstations or grid infrastructures dedicated to high performance computing.

In this regard, X client/server protocol is not suitable. Indeed, X transmission of graphic bitmaps is network intensive (compared to the transmission of minimal geometric information needed to describe medical models) and is not adapted for a high degree of interactivity. Moreover, 3D

graphic visualization is today usually achieved locally using specialized rendering hardware available on most recent computers while remote computing facilities might not provide any specific hardware.

2. Interactive medical applications

Interactive applications follow the simple loop depicted in figure 1. The application is made of an iterative algorithm that progresses by small steps. At each step, the application collects some input from the user (through the mouse or another specialized device), and take into account this input to process the following iteration. After one iteration, the algorithm state is updated and some feedback (usually through visualization) is returned to the user to inform her of the current application progress. Collecting user input and sending feedback might not necessarily happen every algorithm iteration depending on the application.

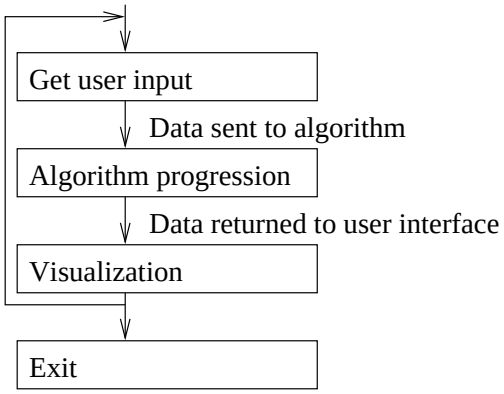

Figure 1. Interactive application general structure.

In particular, medical applications often manipulate 3D data. Most modern medical image acquisition devices (Magnetic Resonance Imaging, Computed Tomography scanners, Nuclear Medicine scanners, Ultrasound devices, etc [1]) are capable of producing 3D images with very detailed informations on the imaged body. To analyze such images, 3D geometrical and anatomical models have been developed [13, 16, 9].

Visualization of 3D medical images in not straightforward since a user cannot at a glance visualize the interior of a 3D volume. Different visualization techniques exist that transform the original volume in a representable data. This might be a single slice extracted out of the volume or a synthesized representation as may be obtained by surface or volume rendering techniques for example [2]. Medical applications usually display additional geometrical representations of some internal modeling of the data. This information is often much more compact than the complete 3D data set. Therefore, rather few information need to be exchanged between the computing machine and the graphic interface.

In the literature, many works dealing with remote data visualization are reported. Many work concern post-processed data visualization [12], while other consider interaction with the data [2]. Some of them offer asynchronous objects management and real-time constraints [17, 11]. However, few of them address the specific needs of medical image processing applications [7, 15], and optimized computational grids for solving medical problems is still an emerging field.

3. Remote 3D visualization and interaction framework

3.1. Porting interactive applications to a remote platform

Our implementation is based on C++ libraries developed in our laboratory, although the proposed framework does not depend on this specific language. Our libraries provide basic objects to describe 2D and 3D medical scenes made of a set of graphical objects and visualization facilities to display all these graphic objects in an application window. Medical scene examples are shown in figures 3 and 4. The user can interact with the 3D scene by different means through the mouse. She can (i) change the display parameters, (ii) select an active graphic object that she can (iii) move (translate, rotate, and rescale) or on which she can (iv) apply application-dependent actions. In addition, an object-dependent menu is proposed for object specific manipulations. We will refer to a *stand-alone* application to mention a classic application designed to run on the local graphic system using these libraries by opposition to a *remote* application that is executed on a remote server. Our goal is to make as few modifications as possible in the user code to move from a stand-alone to a remote application.

In order to execute interactively, a remote application needs to create an interface window on the user local machine. Figure 2 illustrates the remote execution procedure. A graphic daemon is running on the user local machine and waits for incoming connection requests. From its command line interface, the user can connect to a remote machine or use some grid middleware to start the remote application on the desired target (1). At execution time, the remote application needs to know (through a command line parameter) the IP address of the local machine on which the interface should be displayed. It connects on a predefined port of the local machine (2), activating a local process that will deal with the user interface (3). It can then start the main computation loop (4), identical to figure 1 except that all user input is transported from the user interface to the remote process and all visualization data are sent the opposite direction through a communication channel.

In order to be transparent from the user point of view, the same code should be executable either stand-alone (on the user machine, avoiding communication overhead) or remotely, depending on the context. To allow this, the library decouples data structures from graphic representations: to

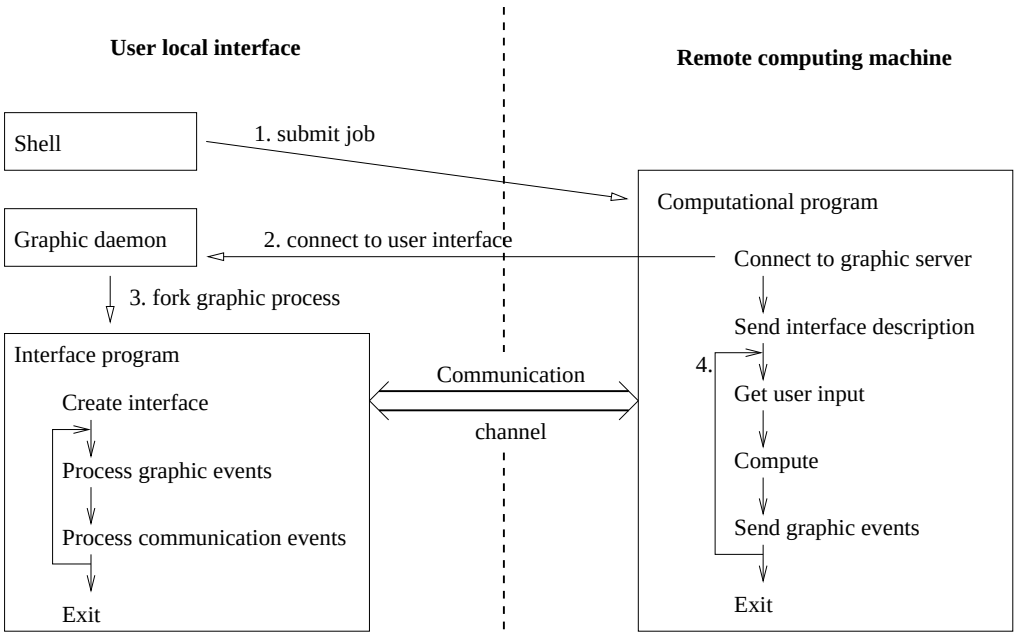

Figure 2. Remote interactive applications overview (see text).

each component of the scene (medical object, model, etc) is associated two objects. The data structure object will remain on the computing host. The graphic object does not hold any data except graphic representation parameters. It is using the geometric information contained in the data object for rendering. Each graphic object redefines two redraw methods: one remote method is sending data to be displayed and on local method is getting the data and actually performs the redraw. In case of stand-alone application, both methods are executed sequentially on the same machine and a stack is used to avoid sending data through the network loopback interface. In case of remote execution, two instances of each graphic object composing the scene are created. The first instance lies on the computing host and is responsible for sending data to the second instance, that lies on the local host, each time a redraw event is received.

3.2. Communications

To obtain a high interactivity level, *e.g.* including real-time requirements such as needed for surgery simulation for instance, communications between the program and the user interface should be as efficient as possible. For some critical applications, a guaranteed network quality of service may even be mandatory. Both user feedback and visualization information should be exchanged between the remote and the local process. Usually, visualization (3D objects) represent much more data than user feedback (mouse clicks) and it is the most critical part in terms of performance.

A two levels communication protocol has been designed in our framework. A higher level interface protocol allows the user to create or delete interfaces widgets and graphic objects. A lower level graphic protocol is designed to send 3D data that compose the graphic scene to the local machine.

3.3. Communication channels

In our implementation we are using direct process communication through UNIX sockets for optimal performance. A C++ object encapsulates the communication channels and performs optimization such as packing small message into a long binary chain that is transmitted using as few packets as possible. On the fly compression could be used to improve performances further. The communication layer is hidden from the user through standard method calls. The system dynamically determine whether it is running stand-alone or remotely and either execute local code or send a message to the graphic daemon for the corresponding code to be executed on the local machine.

3.4. Interface protocol

The interface protocol defines messages to manage interface creation, user interaction, and program termination. Messages sent from the remote to the local machine include TERMINATE (disconnect and terminate interface program), EVALUATE (evaluate transmitted script), CREATE (create a new graphic object), DELETE (delete a graphic object), and REDRAW (cause a redraw of the 3D scene). Conversely, messages sent from the local to the computing machine include TERMINATE (disconnect and terminate computation program), READ (read new data from a file/stream), WRITE (write data to a file/stream),

START (start computation loop), STOP (stop computation loop). the interface communicators on the local and the remote machines periodically poll the incoming messages to respond these commands.

3.5. Graphic protocol

Whenever a viewer window needs to be refreshed (because the viewing parameters changed, the graphic window needs to be redrawn, or the 3D scene has been updated), the local process sends redraw events that cause a redraw procedure of all graphic objects to be called. The redraw procedure needs to retrieve the data content from the remote machine for each graphic object. Therefore, each pair of graphic object instances use pre-defined methods to exchange geometric data (scalar values, vectors...). In case of stand-alone execution, these methods use an internal stack to avoid network communication through the local loopback interface. In case of remote execution, these methods send the graphics information from the remote process to the local machine. The graphic communication protocol includes an INIT message that initiates data transmission from the local side. The remote graphic object responds by a set of SEND messages transmitting the geometric information needed to build the 3D scene. The local graphic object use a set of GET methods to retrieve the messages sent in a known order. Data transmission ends with implicit mutual consent.

3.6. Remote interface and visualization

In our framework, the interface program is generic and does not depend on a precise application. Technically, the interface may be described by different means such as using java byte code sent through the communication channel. In our implementation we rely on a platform-independent windowing system (wxWindows[1] wrapped in the python[2] scripting language). The script is sent through the channel and executed on the local machine. This solution proves to be very flexible: a program can modify its graphical interface dynamically depending on the algorithm evolution and the user interactions. There is not limitation on the generated interface induced by the system.

When an INIT message is received by a graphic object on the remote side, it sends all graphic information needed to redisplay that object. The following pseudo-code illustrates the redraw methods of a polygon made of a list of `n` vertices represented as vectors:

```
void Polyon::localRedraw() {
  int n = getInteger();
  Vector v1 = getVector();
  for(int i = 1; i < n; i++) {
    Vector v2 = getVector();
    draw(v1, v2); v2 = v1;
  }
}
void Polyon::remoteRedraw() {
  int n = data->getNbVertices();
  sendInteger(n);
  for(int i = 0; i < n; i++)
    sendVector(data->getVertex(i));
}
```

The INIT message precedes the entrance into the local redraw code and causes the remote redraw code to start. Only SEND messages from the remote to the local machine are then needed until all data have been exchanged and the communication ends by mutual consent. From the user point of view, message exchanges are hidden in the get/send function calls that are defined for every primitive types or array of primitive types. On a stand-alone execution, both local and remote codes are executed on the same machine. The remote code is first executed. To avoid the overhead of a communication channel, the send methods are redefined to push the addresses of sent data onto a stack. The local code then pops the data out of the stack during the get methods. Using a stack of pointers avoids useless memory copies.

When executing a remote application, two parallel processes are running on two machines: the computation process that periodically sends redraw event and the graphic process that receives and treats redraw events. Depending on the relative cost of the computation loops and the redisplay process, the two process can easily get desynchronized. If the computation process is slower (the usual case for costly applications that require remote execution), the graphic process is just idle between two redraw events. However, if the computation process is faster, it can overflow the graphic process with redraw events and spend too much time in network transmission of the data. To avoid that, our graphic process rejects incoming redraw events when it is already busy by processing such an event. The computing process can then skip data transmission and iterate several times before the 3D scene is refreshed. It is also possible to enforce a minimum redisplay rate, at the cost of slowing down the computation process, in cases where the computations would evolve too fast for the user to get a chance to interact.

3.7. Transparency from the user point of view

The proposed system was designed to be as transparent as possible from the user point of view, while remaining efficient. Indeed, most messages are completely hidden by the graphic system and the user programs call methods that cause the messages to be exchanged. In case of a stand-alone execution, the behavior of the system changes transparently for efficiency reasons avoiding useless message exchanges.

[1] `http://www.wxwindows.org`
[2] `http://www.python.org`

4. Results and discussion

Interactive model-based segmentation algorithms have been ported on our platform for testing. Segmentation algorithms are used to identify structures of interest such as organs in an input image. Model-based segmentation algorithms use a geometrical or statistical model of the organ(s) of interest. A generic model with the *a priori* shape of the organ is embedded into the image. To this model is associated an energy (a real value), trade-off between the regularity of the model (internal energy) and the good fit of the model with the data (external energy). Through an energy minimization procedure, the model can iteratively deform to better fit the image content [16]. At each iteration, the current shape of the model is updated and displayed on the user screen together with the input image. The user can interact by grabbing the model in areas where the automatic convergence is not satisfactory. This algorithm is known in 2D as a *snake* [10], and has been extend in 3D to deformable surfaces (such as *simplex meshes* [6]).

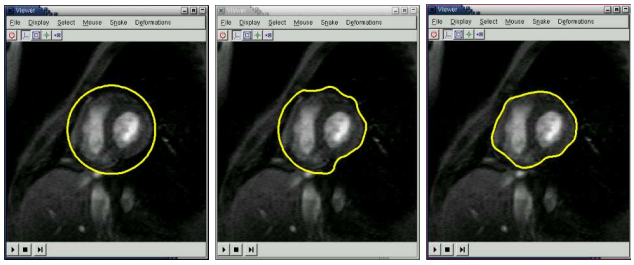

Figure 3. Snake-based 2D segmentation. Left: initialization. Middle: automatic result. Right: supervised result.

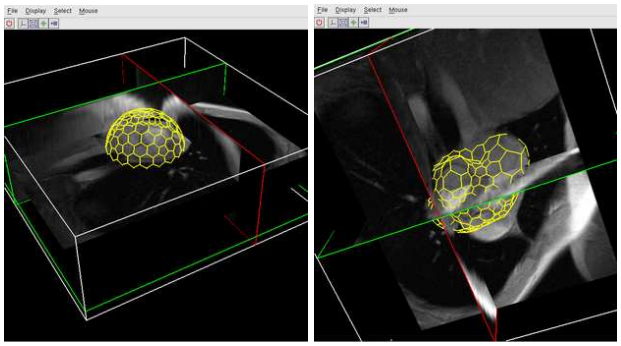

Figure 4. Simplex model-based 3D segmentation. Left: initialization. Right: supervised result.

Figure 3 shows a 2D deformable contour used to segment the heart left ventricle contour from a cardiac Magnetic Resonance slice of the thorax. The initial contour (a simple circle) is overlaid on an MR slice on the left. The result of automatic segmentation is shown in the center, while the result obtained by user interaction (a few mouse clicks while the model is being deformed) is shown on the right. Similarly, figure 4 shows a deformable surface used to segment the heart left ventricle from a Magnetic Resonance volume. On the left is shown the surface initialization overlaid on a 3D medical volume represented by 3 orthogonal slices and on the right is shown the result of supervised segmentation.

Figures 5 and 6 show several performance measurements of the remote execution of 2D and 3D interactive model-based segmentation algorithms ported on our platform.

In first rows of figures 5 and 6 are reported the total number of bytes exchanged during a segmentation task and the corresponding network usage to measure the network stressing. Application-related measurements are also given: the number of iterations per seconds is dependent on the algorithm efficiency. The number of iterations per redraw depends on the relative time needed to make one computation loop and one scene redraw. The total execution time gives an application performance indicator for a fixed number of iteration (given amount of computation). Each measure is reported in 3 cases (columns): (1) the stand-alone application that do not need network transmission of the data, (2) the remote application running the graphic daemon on the same host than the computing process, and (3) the remote application running the graphic daemon and the computing process on two different hosts connected over an IP network (with a measured 200Kbits/s capacity).

To interpret these result, note that the stand-alone application only involves one process. Therefore, the algorithm evolution and the graphic rendering are performed sequentially, with one rendering for each algorithm iteration. Conversely, in the remote application case, two concurrent processes are running (either on the same machine in the local case or on two different machines in the remote case). Due to the relative cost of the rendering compared to a single computation iteration with the fast models used in this testing, this is penalizing for the stand-alone application while the remote application performs several iterations for each rendering. This results in a longer execution time of the stand-alone application and a much larger number of bytes sent to the renderer. Past the initialization stage, where the background image is sent, the number of bytes transmitted for rendering is only dependent on the model size (the number of vertices: *i.e.* 50 vertices for the snake and 500 vertices for the simplex surface in this example) and is fairly low (ten's of bytes per second in the 2D case and thousand of bytes per second in the 3D case) even without compression (for higher resolution models, data compression could be valuable though). This results in a very fluid interaction and makes possible the remote execution of the supervised segmentation application.

Using the X display client/server mechanism for executing the same applications necessitates to transmit the complete interface window bitmap picture at each redraw event. This results in so poor performances that the 2D applications can hardly be considered as interactive anymore and

the 3D application just could not really be executed interactively on a remote machine.

2D snake	stand-alone	local	remote
number of bytes sent	1243	55	45
throughput (Mbits/s)	0.053	0.012	0.008
iterations / second	42.56	209.22	186.86
iterations / redraw	1	51.88	64.64
execution time	4.70	0.96	1.07

Figure 5. 2D remote segmentation performances.

3D simplex	stand-alone	local	remote
number of bytes sent	14667	3529	2473
throughput (Mbits/s)	0.288	0.097	0.042
iterations / second	19	27.56	17.03
iterations / redraw	1	5.8	8.12
execution time	10.2	7.26	11.74

Figure 6. 3D remote segmentation performances.

5. Conclusion

In this paper we presented a 3D object visualization interface for interacting with remote medical applications. The framework is generic and can adapt to many different applications, even outside the medical field. The system is both efficient and transparent from the user point of view. It intends to ease the creation of interactive medical image processing algorithms on remote platforms such as grids.

Efficiency is needed to insure maximal performance in application for which interactivity may be critical (real time constraints may be needed). One purpose of remote execution is to deliver additional computing power to the user. The graphical system should therefore not compensate for the computational improvement.

It is mandatory that the system remains easy to use and as transparent as possible for the user in order to ease new algorithm developments. Our system can run stand-alone when no remote server is available without code modification or recompilation. The message exchanges are hidden from the user.

Acknowledgments

This work is partly supported by the European DataGrid IST project (http://www.edg.org/), the French ministry ACI-GRID project (http://www-sop.inria.fr/aci/grid/public/), and the ECOS Nord Committee (action C03S02).

References

[1] R. Acharya, R. Wasserman, J. Sevens, and C. Hinojosa. Biomedical Imaging Modalities: a Tutorial. *Computerized Medical Imaging and Graphics*, 19(1):3–25, 1995.

[2] C. Bajaj, V. Anupam, D. Schikore, and M. Schikore. Distributed and Collaborative Volume Visualization. *IEEE Computer*, 27(7):37–43, July 1994.

[3] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level Scheduling on Distributed Heterogeneous Networks. In *Supercomputing*, Pittsburgh, PA, USA, Nov. 1996.

[4] V. Breton, R. Medina, and J. Montagnat. DataGrid, Prototype of a Biomedical Grid. *Methods MIMST*, 42(2), 2003.

[5] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientifi c datasets. *Journal of Network and Computer Applications*, 23(3):187–200, July 2000.

[6] H. Delingette. General Object Reconstruction based on Simplex Meshes. *International Journal of Computer Vision*, 32(2):111–146, 1999.

[7] H. Delingette, E. Bardinet, D. Rey, J.-D. Lemarchal, J. Montagnat, S. Ourselin, A. Roche, D. Dormont, J. Yelnik, and N. Ayache. YAV++: a software platform for medical image processing and visualization. In *IEEE International Workshop on Model-Based 3D Image Analysis (IMVIA'01)*, Utrecht, The Netherlands, Oct. 2001.

[8] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, July 1998.

[9] A. Frangi, W. Niessen, and M. Viergever. Three-Dimensional Modeling for Functional Analysis of Cardiac Images: A Review. *IEEE Transactions on Medical Imaging*, 20(1):2–25, 2001.

[10] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. In *International Conference on Computer Vision (ICCV'87)*, pages 259–268. IEEE, 1987.

[11] K. Kim. APIs for Real-Time Distributed Object Programming. *IEEE Computer*, 33(6):72–80, June 2000.

[12] W. Lefer and J.-M. Pierson. A Thin Client Archietcture for Data Visualization on the World Wide Web. In *proceedings of the International Conference on Visual Computing*, Goa, India, Feb. 1999.

[13] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91–108, 1996.

[14] J. Montagnat, V. Breton, and I. Magnin. Using grid technologies to face medical image analysis challenges. In *Biogrid'03, proceedings of the IEEE CCGrid03*, Tokyo, Japan, May 2003.

[15] J. Montagnat, E. Davila, and I. Magnin. 3D objects visualization for remote interactive medical applications. In *3D Data Processing, Visualization, Transmission*, June 2002. accepted.

[16] J. Montagnat, H. Delingette, and N. Ayache. A review of deformable surfaces: topology, geometry and deformation. *Image and Vision Computing*, 19(14):1023–1040, Dec. 2001.

[17] C. Schmidt and F. Kuhns. An overview of the Real-Time CORBA Specifi cation. *IEEE Computer*, 33(6):56–63, June 2000.