Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

WEB-BASED SIMULATION OF SYSTEMS DESCRIBED BY PARTIAL DIFFERENTIAL EQUATIONS

Manuel Alfonseca
Juan de Lara

ETS Informatica
Universidad Autonoma Madrid
Madrid, 28049, SPAIN

Hans Vangheluwe

School of Computer Science
McGill University
Montreal, Quebec, H3A 2A7, CANADA

ABSTRACT

This paper describes how to take advantage of Internet services and object technology to solve 2D partial differential equations (PDEs) in a distributed manner. This is accomplished by means of a distributed object oriented continuous simulation language designed by our research group, called *OOCSMP*, and a Java (and C++) generating compiler for this language (called *C-OOL*). We also describe a graphical mesh generator, which produces *OOCSMP* code. The mesh generator may be invoked from the simulation model allowing the generation of domains and meshes during model execution. The simulation of the heating of several moving pieces is shown, in the single machine case, and in the distributed case.

1 INTRODUCTION

The Internet is becoming ever more popular. The number of computers connected to the Internet allows us to take advantage of the combined power of these computers. This enables us to solve more interesting and complex problems (Java Grande 1999). Many disciplines are re-evaluating their strategies and techniques (Page et al. 2000) in view of the services offered by Internet. For simulation, example services are the distributed networked architecture as well as common protocols for computer communication. Thanks to the "single point of access" an Internet web browser offers, it is gaining popularity in the implementation of courseware. To make such courses more dynamic, simulation models can be integrated (de Lara and Alfonseca 2001).

According to (Fishwick 1996), the web offers services that can be exploited by the field of simulation in various ways, such as building and/or executing distributed models (Cubert and Fishwick 1998) , (Alfonseca and de Lara 2000a), (Alfonseca and de Lara 2000b), or providing Web-based tools, such as simulation interpreters, plug-ins, etc. (Schmid 1999) In this paper we will focus on the execution of distributed models.

Since 1997, we have developed advanced simulation tools that simplify the generation of educational courses for the WWW (Alfonseca et al. 1999). For this purpose, we use the continuous simulation language *OOCSMP* (Alfonseca el al. 1997), designed by our group. This language is an object oriented extension of the original *CSMP* (IBM 1972). Other extensions have also been added, such as the capability to solve partial differential equations (de Lara and Alfonseca 1999), the possibility to synchronize multimedia elements with the simulation execution, as well as new primitives to produce distributed simulations.

In previous publications (de Lara and Alfonseca 1999), (Alfonseca and de Lara 2000a) we have presented several approaches to the solution of PDEs using object technology and web services. In *OOCSMP*, the use of object technology simplifies the domain creation and discretization: generic domains and meshes can be defined within classes, and parametrized in objects. It is also possible to transform those meshes once created, by means of displacement, rotation, scale, etc. operators. The interaction between subsystems (modeled as objects) of the global system (modeled as a collection of objects) is expressed in a natural and extensible way. The system allows the addition of new objects during the simulation execution in the single-machine case.

With the use of web services, and the *OOCSMP* distribution capabilities, it is possible to place each object contributing to the solution of a PDE in a different machine, speeding up the simulation considerably. This distribution approach is quite different (and perhaps more natural) from the distribution scheme used in classical continuous simulation (Monsef 1997), consisting of the parallelization of the algorithms used to solve the model's system of equations. Besides, it is not possible for us to use fine grain parallelism, due to high (and impredictable) latency in the Internet. Schemes where each processor is assigned one element of the matrix resulting from discretization and communicates constantly with the processors of the neighboring elements are not possible in our case. We must try to exploit large grain parallelism.

In spite of this high latency, the use of Internet as the interconnecting network has some advantages:

- it is an existing infrastructure,
- it is highly scalable,
- it offers common and well established protocols and communication mechanisms, and
- the use of Java makes the system highly portable.

Other continuous simulation systems focus on distribution for collaborative and distributed modeling (Cubert and Fishwick 1998) rather than on simulation.

In this paper, we present some techniques to solve PDEs using object technologies. The techniques are illustrated by the solution of a problem: the heating of two moving pieces. The configuration of each piece can be modified at run time. The distribution capabilities of our simulation language are also used to produce distributed Java code for the problem. The code generation takes into account that sometimes the heat in both pieces can be computed simultaneously (if they are separated), while sometimes they must be calculated as one whole in a single processor (if they are in contact).

The paper is organized as follows : Section 2 gives a brief introduction to the *OOCSMP* language, the *C-OOL* compiler and the mesh generator; section 3 describes the procedure to solve PDEs; section 4 presents the distribution capabilities of *OOCSMP*; section 5 presents the solution of the example, in the single machine case; section 6 extends the model to make it distributed. Finally, section 7 presents the conclusions and the future work.

2 *OOCSMP*, *C-OOL* AND *MGEN*

During the past four years, we have developed several extensions to an old continuous simulation language called *CSMP*. The new language has been baptized *OOCSMP*. Some of its characteristics are:

- it is Object-Oriented,
- it supports an algebra of vectors and matrixes,
- it can handle discrete events,
- it can synchronize multimedia elements with the simulation execution,
- it is able to solve second order PDEs and includes a mesh generator,
- it has primitives to generate distributed simulations,
- it has instructions to build a web page where the experimentation environment (including simulator, controls, result presentations, …) will be palced, and
- several output forms are available, and can be mixed in a single problem.

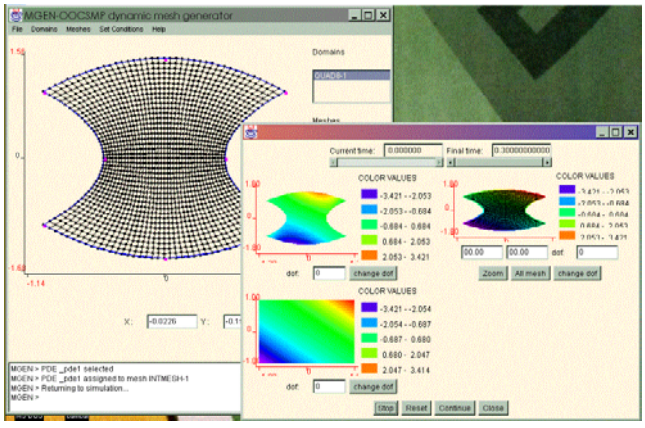

Figure 1: Using *MGEN* During a Simulation Execution

The compiler is named *C-OOL* (a Compiler for the *OOCSMP* Language) and generates three different languages: Plain C++, C++ using the Amulet library (Myers 1997) and Java.

For the three cases, *C-OOL* produces a fully configurable user interface which allows the user to experiment with the problem and answer *"what if...?"* questions. The compiler and the language have been mostly used to produce educational web courses based on simulation. In this case, the compiler generates Java code (*applets*) and HTML pages. The courses can be found at:

`<http://www.ii.uam.es/~jlara/investigacion>`

C-OOL is also able to generate documentation for the simulation models in the form of HTML pages.

MGEN is a graphical mesh generator (programmed in Java) that generates *OOCSMP* code for the mesh, domains, and conditions. *MGEN* can also read *OOCSMP* code containing a description of the problem, and makes it possible to change it graphically. The generator can also be integrated with the model, as another *OOCSMP* input/output panel. In this way, one can pause the simulation, change some mesh parameters, and continue with the execution. Figure 1 shows the use of *MGEN*.

3 THE PROCEDURE TO SOLVE PDEs WITH *OOCSMP*

OOCSMP is able to solve PDEs and systems of PDEs of second order in one or two spatial dimensions and time. PDEs are solved in *OOCSMP* using the following 8-step procedure:

1. Declare the basic domains. Domains are declared by means of primitives. In two dimensions, there are four different primitives (quadrilaterals, 8-node-quadrilaterals, circular sectors and triangles).

2. Set the initial and boundary conditions. Initial and boundary (natural, essential or periodic) conditions have to be imposed when declaring the basic domains. These conditions can be provided by means of any *OOCSMP* expression. Conditions can be set over the edges (*EDGE*), the corners (*CORNER*) or individual nodes (*NODE*). They can be specified as lists of elements (one or more elements separated by commas) or as intervals (two elements separated by a colon).
3. Form the grid. Basic domains can be discretized using several meshing techniques, both structured (isoparametric elements and elliptic grid generation (Thompson et al. 1985)) and unstructured (Delaunay triangulation (Hsuan-Cheng 1997)). In the last case, three different constraints can be applied to each triangle generated (they can be combined in an *AND* expression):
   - a maximum area,
   - a maximum edge size, and
   - a minimum angle.

   In the Delaunay triangulation it is also possible to generate quadrilaterals, by dividing each triangle into three quadrilaterals (with a common vertex in the triangle baricenter).
   Meshes can be smoothed using Laplacian smoothing (Field 1988).
4. Several meshes can be concatenated to produce more complex meshes, even if they have been generated with different techniques. Geometrical transformation operations can be applied to domains or meshes : move (*MOVE* method), rotation (*ROTATE* method) and scale (*SCALE* method).
5. Declare the PDE(s). In this step, the partial differential equations to be solved have to be declared. One equation is declared for each sub-mesh in the total mesh. The solution method for the PDE has to be chosen at this stage. In *OOCSMP* we have implemented several schemes of the finite differences method (Strikwerda 1989):
   - Explicit schemes, such as classical *CTCS*, *FTCS*, etc. (configurable, but the system can choose the better method for the problem) or the Du Forte-Frankel method.
   - Implicit schemes, such as ADI or Crank-Nicolson.

   The finite element method (Zienkiewicz and Taylor 1989) can also be used. For the finite difference case, meshes are restricted to the isoparametric type, but grid lines are not restricted to be parallel to the X and Y axes. If the system detects that they are, the solving algorithm is much faster, because no transformation is needed from the physical domain to the computational domain (a 2x2 rectangle). Finite element meshes are restricted to have the same type of simplexes in each submesh.
   The Java and C++ libraries implementing the methods have been coded to allow easy variation of the schemes, as new schemes can inherit common algorithms from the existing ones and change only a few details. For example, the class implementing the Du Forte-Frankel method is a subclass of the class implementing classical explicit methods, where we have only to change the methods that indicate how to discretize some derivatives.
6. Assign the equation(s) to the mesh. At this stage, each equation has to be assigned to the appropriate mesh. Different equations can be assigned to each sub-mesh. They may be the same equations with different solution methods. If this is the case, care must be taken, because perturbations can occur in the borders of concatenated meshes.
7. Solve. Method *STEP* has to be invoked for the total mesh.
8. Select the output form. Several ouput forms are available in *OOCSMP* (in the Java case) to view PDE solutions, such as :
   - maps of isosurfaces,
   - plots to view the nodes of the grid, with the initial conditions,
   - three dimensional plots, and
   - plots for vectors.

   All these outputs refresh the solution at *PLdelta* intervals. Several output forms can be combined in a single problem.

A library of components with typical PDEs has been developed. The components are *OOCSMP* classes containing a discretized basic domain. Classes can be parametrized when declaring objects. Typical parameters are: the number of elements in the grid, the constrains if Delaunay triangulation is chosen, or the coefficients of the PDE. Component grids can be concatenated.

4 DISTRIBUTION IN *OOCSMP*

Distribution has been implemented in *OOCSMP* using the *rmi* Java package (Berg and Fritzinger 1998). For this reason, it is only available when generating Java code.

Distribution has been implemented at the object level: it is possible to choose the machine where every object taking part in the simulation model is going to be placed. Several objects can be placed in the same machine. Objects can also be replicated in all the machines of the distribution

scheme. This allows us to emulate object migration by activating/passivating objects.

This distribution scheme is a natural way (although not the most efficient) of describing distribution in object-oriented simulation models. It fits the distributed nature of the Internet, and follows the current vision (Fishwick 1998), (Page and Opper 1999a) of a web populated with digital objects (simulation models). Thereby each object resides in a different location, and modeling is done in a composable, distributed fashion.

It is useful to identify clusters of objects that communicate mostly inside the cluster. We have applied this scheme successfully to ecological simulations (Alfonseca and de Lara 2000b), where species populations are represented as objects, and ecosystems as collections of these objects. Each ecosystem is placed in a different machine, and each species only interacts with those in the same machine, except for migration.

Each machine (IP address) taking part in the simulation is assigned a label. This information is usually placed in a separate file. Labels will be used in all the subsequent references to machines. Thus, changes in the distribution scheme do not imply a modification of the main model, only a recompilation. It is also possible to assign the same IP address to different labels.

When invoking the object constructor, it is necessary to indicate in which machine the object will be created (by means of the machine label). If a label is not given, the object is replicated in all the machines. Global variables are likewise replicated.

Different integration schemes can be used in each machine. The output forms can also be different, and it is possible to monitor remote objects.

In this way, a single model is needed, but it has to be compiled for every machine in the simulation. By default, the compiler puts synchronization points where necessary, and keeps the simulation time synchronized (if there is some dependency between objects), by adding synchronization points at the end of every simulation loop. Semaphores can be omitted by means of compiler options.

As each machine executes a main simulation loop, the computation control is completely distributed among processors. The compiler generates customized Java code for each machine, in such a way that :

- Single calls to methods invoked on local objects are executed.
- Each machine only executes local calls on its local objects. It sets semaphores to keep execution globally synchronized.
- Methods or attributes on remote objects, whose results are needed to compute local variables, are accessed via remote method invocation, but this is transparent to the user.

All object access (local and remote) is regulated via semaphores. If the object is local, the semaphore is called in the local machine. If it is remote, it is accessed in the remote machine. Semaphores control both simulation time synchronization and object dependencies, and are associated with a special attribute in each class. Each invocation of a method of one object increases that variable , which is reset at the beginning of the simulation loop. In this way, it is easy for the compiler to produce a parallel simulation equivalent to the serial one (in the sense that it gives the same result). Semaphores take into account the request and local times and the number of previous accesses to an object to obtain a better synchronization.

Instructions involving assignments to global variables, or calls to global functions or procedures are replicated in all the machines. This calculation replication is not a waste of time, because the latency is high. Sometimes calculation replication is a better choice than communication.

5 A STAND-ALONE SAMPLE PROBLEM

In this section, we present a problem. It is a more complex version of the one described at (Alfonseca and de Lara 2000a). Suppose we want to simulate the heating of two L-shaped pieces. The heat supplied follows the equation: $e^{2t} \sin(x+y) \cosh(x+y)$. A scheme of this problem is shown in Figure 2. The heat supplied is shown as a map of isosurfaces, the blue color (lower left corner) representing the cooler temperatures, the red color (upper right corner) the higher temperatures. The pieces are moving, approach until they collide, stand for a while, and then separate. This means that sometimes it is possible to integrate both pieces separately, while sometimes the calculation has to be accomplished in a single mesh.

We can not take advantage of the symmetry of the problem to reduce it to the calculation of only one piece, as the boundary conditions are not symmetric, and the pieces

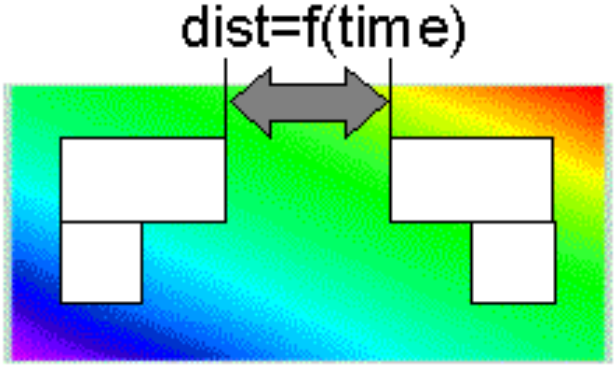

Figure 2: Scheme of the Problem

move and join at some point in time. But we can generalize the geometry of the pieces by encapsulating all their behavior inside a class (called *LPiece*). This class has information about the domain, the mesh, the equation to be solved and the conditions. The domain is the concatenation of two quadrilaterals of size 2x1 and 1x1. *LPiece* objects take as parameters the displacement (in X and Y) to be applied to the each quadrilateral, and the conductivity coefficients. It is easy to obtain pieces with a different geometry, by changing the parameters of the constructor. This can be done at run time.

The following listing shows the complete *OOCSMP* model.

```
  DATA EPS:= 1e-4, ACER:= 1
  DATA JOINED:= 0, MAXJOINTIME:= 6
  APPHEAT X, Y
    APPHEAT := EXP(2.0*TIME)*SIN(X+Y)*CH(X+Y))

  CLASS LPiece
  {
    * Define class parameters, displacement
    * of the piece components
    DATA trx1, try1, trx2, try2
    DATA Kx, Ky

    DOMAIN qd1:= QUADRILATERAL(0,0,2,0,2,1,0,1,
                    INITIAL(0),ESSENTIAL(EDGE(1:4),
                    APPHEAT(X,Y))

    DOMAIN qd2:= QUADRILATERAL(0,-1,1,-1,1,0,0,0,
                    INITIAL(0),ESSENTIAL(EDGE(1:4),
                    APPHEAT(X,Y))
    qd1.MOVE (trx1, try1)
    qd2.MOVE (trx2, try2)

    * Mesh the domains
    MESH m1:=ISOPARAMETRIC(qd1,QUADRILAT4,
                           ELEMENTS(20,10))
    MESH m2:=ISOPARAMETRIC(qd2,QUADRILAT4,
                           ELEMENTS(10,10))

    * Create 2 PDEs
    PDE H2da(0,0,1,1,-Kx,1,-Ky,0,0,0,0,0,0,0,0,FEM)
    PDE H2db(0,0,1,1,-Kx,1,-Ky,0,0,0,0,0,0,0,0,FEM)

    m1.CONCAT(m2)
    m1.setPDE(H2da)
    m2.setPDE(H2db)

    DYNAMIC x
      m1.MOVE(x,0)
      m1.STEP()
  }

  LPiece p1 (0, 0, 0, 0, 3.0, 3.0 )
  LPiece p2 (3, 0, 4, 0, 3.5, 4.0 )

doJOIN
  p1.m1.CONCAT(p2.m1)
  ACER *= -1
  JOINED := MAXJOINTIME

doMOVE
  JOINED -= 1
  p1.STEP( ACER*0.05 )
  p2.STEP( -ACER*0.05)

doDETACH
  p1.m1.DETACH(p2.m1)
  doMOVE

DYNAMIC
  DIST := ABS(p1.m1.RIGHTX-p2.m2.LEFTX)
  JOINED-=1
  INSW ((DIST+EPS)*(EPS-DIST),
        FCNSW(JOINED,doJOIN, ,),
        doMOVE)
  FCNSW(JOINED, , doDETACH, p1.m1.STEP() )
  INSW (DIST-2, , ACER*=-1)

TIMER FINTIM:= 20, delta:= 0.1, PLdelta:= 0.1
GRIDPLOT [C], p1.m1, p2.m1
ISOPLOT  [E], p1.m1, p2.m1
```

Listing 1: The Single Machine Model

Both pieces move parallel to the X-axis. When their distance is less than a given small interval, both pieces are concatenated, otherwise, they approach or separate. The distance is computed by means of the *RIGHTX* and *LEFTX* predefined methods, that return the rightmost and leftmost X coordinate. These methods can be invoked on meshes or on domains. In *OOCSMP*, if a method doesn't receive parameters, it can be invoked with or without the trailing parentheses. In this way, one need not worry about whether the requested information is implemented as a method or as an attribute.

When both pieces collide, the resolution method for the first mesh has to handle the resolution of both meshes. In order to accomplish this, the method has to reconfigure its internal vectors, recalculate Jacobians, change the boundary conditions in the resulting inner borders, copy the solutions from the second mesh, etc. This is a different situation from the concatenation that occurs in the declarative section of the *LPiece* objects, which occurs before the solvers have been created, and the compiler knows that it has to handle the solution of both sub-meshes. This is a more efficient concatenation.

When both pieces are joined, only one *STEP* message has to be sent to the first mesh in order to solve the equation in the whole domain. When the meshes separate, two *STEP* messages have to be passed to both meshes.

After some time before the concatenation, both meshes detach. The *DETACH* predefined method reconfigures both solvers, changes the boundary conditions at the bond borders, etc. This situation is treated as a discrete event (handled by the *FCNSW* method). Blocks *INSW* and *FCNSW* are the *OOCSMP* discrete event handlers. They are overloaded in such a way that they can act as "event handlers" (as in this program) or as normal *OOCSMP* blocks (when they appear to the right of an assignment).

Finally, we choose two graphical output forms, one to display the nodes of both grids (*GRIDPLOT* instruction) and another to view the solution of the PDEs (*ISOPLOT* instruction) in the form of a map of isosurfaces. The latter is able to distinguish if both meshes are separated or they have been concatenated (plotting only one or both of them).

The model is compiled into an applet with *C-OOL* and the screen in Figure 3 is obtained. This picture was taken when both meshes are concatenated. The applet includes several buttons to control the simulation and change the parameters at run time. The applet can be found at: `<http://www.ii.uam.es/~jlara/investigacion/ecomm/moving.html>`.

The *HTML* page containing the applet has also been generated with *C-OOL* using several language extensions discussed in (de Lara and Alfonseca 2001).

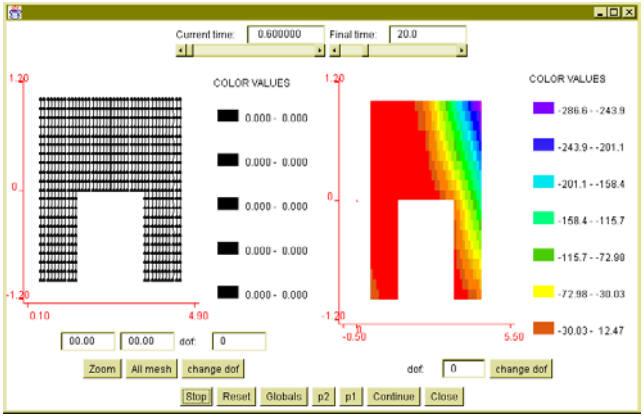

Figure 3: A Moment in the Simulation of the Previous Problem

6 MAKING THE MODEL DISTRIBUTED

This example problem is very suitable for distribution. When the pieces are separated, they can be solved simultaneously. When they are joined, a single computer has to carry out all the computation.

To make the model distributed, the only thing we have to do is to indicate the machine where the objects have to be created. The necessary changes to make the model distributed are shown in listing 2.

```
MACHINE m1 "urano.ii.uam.es"
MACHINE m2 "minerva.ii.uam.es"
...
LPiece p1 (0,0,0,0,3.0,3.0) MACHINE m1
LPiece p2 (3,0,4,0,3.5,4.0) MACHINE m2
...
GRIDPLOT [C], [MACHINE=m1], p1.m1
ISOPLOT  [E], [MACHINE=m1], p1.m1
GRIDPLOT [C], [MACHINE=m2], p2.m1
ISOPLOT  [E], [MACHINE=m2], p2.m1
```

Listing 2: Changes to Make the Model Distributed

The compiler locates the synchronization points automatically. The critical points are :

- The concatenation of both meshes, as all the information about the solution found on the second mesh has to be sent from one computer to the other, and from that point on, only the first computer calculates, while the other is waiting.
- The separation of both meshes, when the solution found for the second mesh in the first computer has to return to the second machine. From this point on, until the next concatenation, both computers can solve the equations separately.

Other synchronization points arise when calculating global variables that need information about distributed objects (global variables, as *DIST, ACER* and *EPS* are replicated in both machines). One of these moments is, for example, the instant at which the distance between pieces is calculated. This is done in the main simulation loop in both machines, and the information about the remote piece is accessed via a remote method. When compiling the *OOCSMP* code, *C-OOL* realizes which remote methods are going to be needed. In our case :

- wrappers to execute *RIGHTX* and *LEFTX* methods on remote meshes,
- a method to access remote mesh *m1* (used when invoking the *CONCAT* method),
- a method to return the calculated solution of *m1* on object *p2* (used when invoking the *DETACH* method).

We have changed the output forms to view only the local objects. When both pieces are concatenated, the solution of both pieces is shown on machine *m1*. Information about physical machine locations (the first two lines) is usually placed in a separate file, and included in the main model. In this way, it is possible to change the distribution scheme, leaving the main model untouched.

The efficiency of the distributed model is much higher than the single machine model, because during most of the simulation time both pieces can be calculated simultaneously.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented several techniques which simplify the solution of partial differential equations using object technology. The objects included in the simulation

model can be distributed in a natural way across the Internet, and executed more efficiently than in a single computer. The transition from single-machine models to distributed models can be performed with minimal effort and change.

According to (Page et al. 1999), in a distributed simulation system, the model designer should not have to worry about the details of the technology used by the compiler to produce distributed simulations. This is fully accomplished by our system: the only information that has to be provided to the compiler is the address of the participant computers. The details about the underlying technology (in our case, Java and *rmi*) used by the compiler to produce the distributed simulation is completely hidden from the user.

We are considering to build an environment to simplify the construction of the *OOCSMP* models, as the compiler does not have a graphical interface. With this environment, we would be able to debug the models, even in the case of distributed simulations. The tool would also help to validate the model, to design experiments, and in the construction of web pages, become an authoring tool for the development of simulation web-courses. The tool could also cover distributed and cooperative modeling, making use of distributed model repositories, such as in (Cubert and Fishwick 1998).

Our distribution scheme can be made more efficient by using, for example, more sophisticated forms of computing replication, techniques for hiding latency, data advancement, data cachéing, etc.

We also plan to migrate from *rmi* to *Corba* (Berg and Fritzinger 1998) as the supporting package to handle distribution. *Corba* would allow us to mix Java and C++ code in a single simulation.

An important concept, not yet covered by our work is the notion of interoperability (Page 1998). Interoperation means reuse of components, by using them in new contexts, enabling the addition or deletion of components at simulation run time. This concept is related to the notion of component-based modeling (Page and Opper 1999b) and model reuse. Uniform and well defined interfaces for objects taking part in the simulation will have to be designed. In our example, this would allow us to dynamically add to the simulation new pieces in remote machines (perhaps objects belonging to a different class). This feature has been incorporated to the High Level Architecture (HLA) (Dahmann et al. 1997) (HLA 2001), developed by the U.S. Department of Defense, and established as a standard for distributed simulation applications.

ACKNOWLEDGMENTS

This paper has been sponsored by the Spanish Interdepartmental Commission of Science and Technology (CICYT), project number TEL1999-0181

REFERENCES

Alfonseca,M., Pulido, E., Orosco, R., de Lara, J. 1997. OOCSMP: An Object-Oriented Simulation Language. *Proceedings of the 9th European Simulation Symposium ESS97*, SCS Int., Erlangen, Germany, pp. 44–48.

Alfonseca, M., de Lara, J., Pulido, E. 1999. Semiautomatic Generation of Web Courses by Means of an Object-Oriented Simulation Language. *Special issue of "SIMULATION", Web-Based Simulation*, Vol 73(1):5-12.

Alfonseca, M., de Lara, J. 2000. Distributed Simulation of systems based on Partial Differential Equations at the Internet. *IMACS'2000*. Lausanne, Switzerland.

Alfonseca, M., de Lara, J. 2000. Distributed Simulation of Ecosystems for the Internet. *IMACS'2000*, Lausanne, Switzerland.

Berg,D.J., Fritzinger, S. 1998. Advanced Techniques for Java Developers. *Wiley Computer Publishing*.

Cubert, R.M., Fishwick, P.A. 1998. OOPM: An Object-Oriented Multimodeling and Simulation Application Framework. *SIMULATION*, Vol. 70(6):379-395.

Dahmann, J.S., Fujimoto, R.M., Weatherly, R.M. 1997. The Department of Defense High Level Architecture. *Proceedings of the 1997 Winter Simulation Conference.*

de Lara, J., Alfonseca, M. 1999. Simulation partial differential equations in the World Wide Web. *Proceedings of the EUROMEDIA'99*, SCS Int., Munich, Germany, pp. 45-52.

de Lara, J., Alfonseca, M. 2001. Constructing Simulation-Based Web Documents. *IEEE Multimedia, Special issue on Web Engineering*. January-March. pp. 42-49.

Elmqvist,H., S.E.Mattson. 1997. An Introduction to the Physical Modeling Language Modelica, *Proc. 9th European Simulation Symposium ESS97*, SCS Int., Erlangen, Germany, pp. 110–114. See also `<http://www.modelica.org>`

Field, D.A. 1988. Laplacian smoothing and Delaunay triangulations. *Comm. Applied Numer. Meth.*, 4:709-712.

Fishwick, P.A. 1996. Web-based Simulation: Some personal Observations. In *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, 1996, pp. 772-779.

Fishwick, P.A. 1998. Issues with Web-Publishable Digital Objects. In *Proceedings of SPIE: Enabling Technologies for Simulation Science II*, pp. 136-142.

HLA web page `<http://hla.dmso.mil/>`

Hsuan-Cheng, Lin. 1997. JAVAMESH- A two dimensional triangular mesh generator for finite elements. MS thesis at Pittsburgh University.

IBM Corp. 1972. Continuous System Modelling Program III (CSMP III) and Graphic Feature (CSMP III Graphic Feature) General Information Manual. IBM Canada, Ontario, GH19-7000.

Interim Java Grande Forum Report. Java Grande Forum Technical Report JGF-TR-4, 1999, see : <http://www.javagrande.org/report.htm>.

Monsef, Y. 1997. Modelling and Simulation of Complex Systems. *SCS Int*. Erlangen.

Myers, B. et al. 1997. The Amulet v3.0 reference Manual. *Carnegie Mellon University School of Computer Science Technical Report no. CMU-CS-95-166-R2* and *Human Computer Interaction Institute Technical Report CMU-HCII-95-102-R2*.

Page, E.H. 1998. The rise of web-based simulation : Implications for the High Level Architecture. *Proceedings of the 1998 Winter Simulation Conference*, Washington D.C, USA.

Page, E.H., Opper, J.M. Investigating the Application of Web-Based Simulation Principles within the Architecture for a Next-Generation Computer Generated Forces Model. To appear in: *Future Generation Computer Systems, Elsevier Science Publishing.*

Page, E.H., Nicol, D.M., Balci, O., Fujimoto, R.M., Fishwick, P.A., L'Ecuyer,P., Smith, R. 1999. Panel: Strategic Directions in Simulation Research. *Proceedings of the 1999 Winter Simulation Conference*, pp. 1509–1520.

Page, E.H., Opper, J.M. 1999. Observations on the Complexity of composable simulation. *Proceeding of the 1999 Winter Simulation Conference*, pp. 553-560.

Page E.H. Buss, A., Fishwick, P.A., Healy, K., Nance, R.E., Paul, R.J. Web-Based Simulation: Revolution or Evolution?. ACM Transactions on Modelling and Computer Simulation.

Schmid, Ch. 1999. A Remote Laboratory Using Virtual Reality on the Web. *Special issue of SIMULATION, Web-Based Simulation*, 73(1):13-21.

Thompson, J.F., Warsi, Z.U.A., Mastin, C.W. 1985. Numerical Grid Generation. *Elsevier Science Publishing CO.Inc.* In internet at: <http://WWW.ERC.MsState.Edu/education/gridbook>.

Strikwerda, J.C. 1989. Finite difference schemes and partial differential equations. *Chapman & Hall*; New York.

Zienkiewicz, O.C, Taylor, R.L. 1989. The Finite Element Method, 4th edn, vol. I. *McGraw-Hill*; New York.

AUTHOR BIOGRAPHIES

MANUEL ALFONSECA is the Faculty Subdirector in the Department of Computer Science at the Universidad Autonoma of Madrid (UAM) where he teaches and conducts research. His research interests include computer languages, simulation, complex systems, object-orientation and theoretical computer science. He received his PhD in electrical engineering in 1972 and his MSc in computer science in 1976, both at the Universidad Politecnica of Madrid. He is a member of the SCS, New York Academy of Sciences, IEEE Computer Society, ACM, British APL Association, and Spanish Association of Scientific Journalism. e-mail: <Manuel.Alfonseca@ii.uam.es>

JUAN DE LARA is an associate professor at UAM, where he teaches software engineering. His research interests include Web based simulation, Meta-Modeling, distance learning and social agents. He received his PhD in June 2000, at UAM. He graduated in 1994 with a Top of Class Award as a Technical Engineer in Computer Science. In 1996 he received the honor of Higher Engineer in Computer Science. At present, he is performing postdoctoral research at McGill University with professor Hans Vangheluwe. e-mail: <Juan.Lara@ii.uam.es>

HANS VANGHELUWE has been a Research Coordinator in the Department for Applied Mathematics, Biometrics, and Process Control (BIOMATH) of Ghent University, Belgium for the past decade. He is currently an assistant professor in the School of Computer Science of McGill University in Montreal, Canada where he teaches Modelling and Simulation, Software Design, and Computer Networks. He also heads the Modelling and Simulation research lab. e-mail: <hv@cs.mcgill.ca>